Comparative Analysis of MD5, SHA-256, and BLAKE2 for Tamper Detection in Indonesian Food Composition Data (TKPI)

Thaffariq Azka Rahmat – 18223048[1,2]
Department of Information System and Technology
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10, Bandung 40132, Indonesia
[1]18223048@std.stei.itb.ac.id, [2]thaffariq@gmail.com

***Abstract*—Data integrity in nutritional databases is critical, as even minor undetected modifications to food composition values can have great implications for dietary assessment, public health policy, and consumer trust. Amidst Indonesia's large-scale Free Nutritious Meal Program (Makan Bergizi Gratis), the relevance of this issue has elevated, which relies on national nutrition standards to serve millions of beneficiaries nationwide, underscoring the need for reliable mechanisms to safeguard the integrity of underlying nutritional data. This paper presents the design, implementation, and experimental evaluation of a cryptographic hash-based tamper detection system applied to the Indonesian Food Composition Table (TKPI), comprising 1,346 food records sourced from Panganku (Kementerian Kesehatan RI). Three widely used hash algorithms, consists of MD5, SHA-256, and BLAKE2b, are compared across three dimensions: tamper detection capability, avalanche effect characteristics, and computational performance at varying dataset scales. In addition, a Merkle tree structure is proposed and implemented as a mechanism for efficient localization of tampered records without requiring full re-hashing of the dataset. Experimental results demonstrate that all three algorithms successfully detect single-field modifications, including changes as small as 0.1 gram in nutrient values, with each algorithm producing a completely different digest upon tampering. SHA-256 and BLAKE2b shows near-ideal avalanche behavior (~50% bit change), whereas MD5 shows a higher deviation at 58.6%, consistent with its known cryptographic weaknesses. In terms of performance, all three algorithms achieved comparable throughput in the range of approximately 25,000–53,000 records per second, with relative speed differences depending on dataset size and hardware rather than any single algorithm being consistently superior. The implemented Merkle tree localizes tampered records within 12 traversal steps for a 1,346-node tree, offering great efficiency over linear search. Based on these findings, SHA-256 is recommended for security-sensitive applications due to its balance of security and standardization, while MD5 is discouraged despite its competitive speed, owing to its weaker avalanche characteristics and well-documented collision vulnerabilities.**

Keywords—cryptographic hash function; data integrity; MD5; SHA-256; BLAKE2b; Merkle tree; food composition data; TKPI; tamper detection

I. Introduction

A. Background

Food nutrition composition data is a fundamental instrument in formulating food policies, evaluating consumption adequacy, and developing food products in Indonesia. In Indonesia, the primary reference source for this data is the Indonesian Food Composition Table, which is commonly abbreviated as TKPI and published by the Ministry of Health of the Republic of Indonesia. The 2017 TKPI is an expansion of the 2009 version, containing a compilation of nutritional composition data for Indonesian food derived from research by the Center for Nutrition and Food Research and Development of the Indonesian Ministry of Health, alongside various other sources. Nutrient levels in the TKPI are presented per 100 grams of the edible portion, and this data serves as a reference for healthcare professionals, researchers, educational institutions, and food industry players throughout Indonesia.

The importance of TKPI data reliability is increasingly critical given its current large-scale utilization. The Government of Indonesia, through the National Nutrition Agency (BGN), runs the Free Nutritious Meal Program (MBG). This program is designed as one of the strategic steps toward Golden Indonesia 2045, focusing specifically on tackling stunting and improving public nutrition. The scale of this program is massive: as of January 20, 2026, the number of Nutrition Fulfillment Service Units (SPPG) reached 21,102 units, reaching approximately 59.86 million beneficiaries with a budget realization nearing Rp 18 trillion. The BGN targets the number of SPPGs to reach 32,000 units with 82.9 million beneficiaries and an approved budget of Rp 268 trillion. All menus in the MBG Program must be formulated based on the Recommended Dietary Allowances (AKG) standards set by the Ministry of Health, which in turn refer to food composition databases like the TKPI.

This vast scale of distribution poses great monitoring challenges. Throughout its year of implementation, 177 food poisoning outbreaks caused by MBG were recorded across 127 regencies/cities and 33 provinces, with the total number of MBG poisoning victims from January 2025 to May 2026 reaching 37,270 people. Analysis revealed that the main factor

behind these mass poisonings was the weak implementation of food safety standards resulting from the centralized and rushed execution of the MBG, including weak early detection and field incident response systems. Although the majority of these incidents stemmed from hygiene and food handling issues rather than direct data manipulation, this series of events illustrates a broader systemic vulnerability. It highlights that monitoring and verification mechanisms within the national nutritional data ecosystem, including reference data like the TKPI, still need to be strengthened to reliably support large-scale national programs.

In addition to operational monitoring issues, the TKPI data itself has documented limitations. Most of the data in the 2017 edition of the TKPI is still estimated, obtained through imputed values, which are calculated by matching local food types with similar foods from other references, as well as borrowed values, which represent similar food data taken from other countries without direct testing on local food. In response to these limitations, the National Food Agency in 2023 and 2025 conducted independent laboratory testing on various fresh food commodities to produce more valid analytical values. The results showed differences in values compared to the TKPI for several food types, which could be caused by variations in variety, geographical location, ecosystem, and harvest time of the samples. These findings confirm that food composition data is dynamic and continuously updated. Consequently, any revision or data update should ideally have its authenticity systematically verified instead of relying solely on manual validation.

In this context, data integrity becomes a crucial aspect distinct from data accuracy. While accuracy relates to how correctly a nutritional value reflects the actual food condition, integrity relates to whether the data remains intact and has not undergone unauthorized alterations since it was published or distributed, whether due to technical errors, transmission corruption, or intentional manipulation. As nutritional data like the TKPI is replicated and distributed across various systems, platforms, and stakeholders on a national scale, the risk of undetected data alterations also increases. Therefore, a technical mechanism is required to verify that nutritional data has not undergone any changes since its original source.

Cryptography provides a proven, reliable solution to this problem through cryptographic hash functions. A hash function generates a fixed-size value, known as a digest, from input data. It possesses a unique property where even the slightest change in the input data will produce a completely different digest, a phenomenon known as the avalanche effect. This property makes hash functions an efficient mechanism for detecting data alterations without needing to maintain duplicate copies of the original data. Furthermore, the Merkle tree structure, which is a hierarchical arrangement of hashes, enables the efficient localization of data alterations within large datasets without requiring a linear re-examination of the entire dataset.

B. Problem Formulation

Based on the background described above, this research addresses several key problems. First, it investigates how to effectively design and implement a tampering detection system for the TKPI food composition data by utilizing cryptographic hash functions. Second, it examines the comparative security characteristics, specifically the avalanche effect, alongside the computational performance among MD5, SHA-256, and BLAKE2b algorithms when applied to a nutritional dataset spanning thousands of records. Third, it explores how a Merkle tree structure can be employed to efficiently localize tampered records without necessitating an entire re-hashing process.

C. Research Objectives

The primary objective of this study is to design and implement a tampering detection system based on cryptographic hash functions for the Indonesian food composition dataset. Additionally, this research aims to experimentally compare the performance and security characteristics of the MD5, SHA-256, and BLAKE2b algorithms. Furthermore, it seeks to implement a Merkle tree structure to serve as an efficient tampering localization mechanism. Ultimately, these findings are intended to provide comprehensive recommendations on suitable algorithms for practical application within the framework of national nutritional data monitoring.

D. Research Scope and Limitations

This research is bound by several scopes and limitations to maintain its focus. The dataset used is strictly limited to the Indonesian food composition data sourced from Panganku by the Ministry of Health of the Republic of Indonesia, which has been curated into a machine-readable format consisting of 1,346 records with four nutritional value attributes, namely calories, protein, fat, and carbohydrates. Furthermore, this study focuses exclusively on data integrity detection mechanisms, meaning it does not address physical food safety issues such as hygiene or microbiological contamination. The tampering simulations are conducted solely within a controlled experimental environment rather than on actual production systems. Finally, while the Free Nutritious Meal Program (MBG) is utilized to provide context and demonstrate the scale and urgency of the problem, it does not serve as the data source processed during the experiments.

II. Theoretical Framework

A. Cryptographic Hash Functions

A hash function is defined as a function that maps a bit string of arbitrary length to a fixed-length bit string, with the output referred to as a message digest, hash code, or hash value and used to detect whether the original message has been altered. For a hash function to be considered cryptographically secure, it must satisfy three properties. Collision resistance requires that it be computationally infeasible to find any two distinct inputs that map to the same output. Preimage resistance requires that, given a randomly chosen output, it be computationally infeasible to find any input that maps to that output. Second preimage resistance requires that, given one input value, it be computationally infeasible to find a second, distinct input value producing the same output as the first.

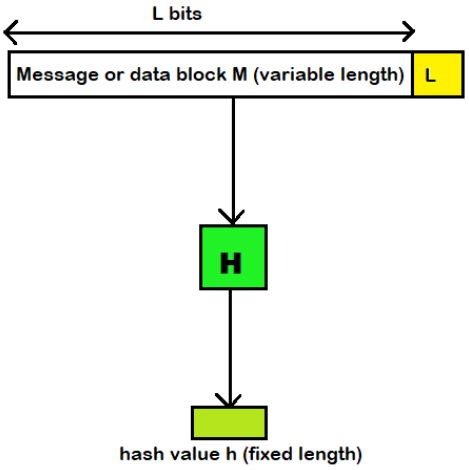

Fig. 1. Generic Hash Function
(Source: GeeksforGeeks)

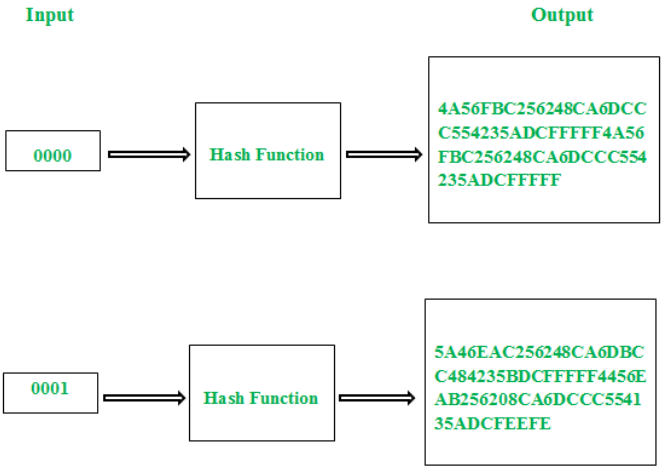

Fig. 2. Avalanche Effect
(Source: GeeksforGeeks)

In this study, three hash algorithms are implemented and compared; each is defined in its own formal specification document, as described below.

MD5 takes as input a message of arbitrary length and produces a 128-bit message digest, or "fingerprint," of that input; the algorithm is designed so that producing two messages with the same digest is conjectured to be computationally infeasible. MD5 is included in this study as a performance and avalanche-effect baseline because of its continued widespread legacy use despite its known collision weaknesses.

SHA-256 is one of several approved secure hash algorithms used to compute a condensed representation of electronic data, producing a 256-bit message digest. It is a federally approved standard for U.S. government cryptographic applications and functions as the de facto baseline algorithm for security-critical use cases.

BLAKE2b is optimized for 64-bit platforms and can produce digests of any size between 1 and 64 bytes, or up to 512 bits. It was designed as a faster alternative to the SHA family while maintaining a comparable security margin, a property that motivates its inclusion in the performance comparison carried out in this study.

B. Avalanche Effect

The avalanche effect refers to the property of a cryptographic hash function whereby a small change in the input, such as flipping a single bit, produces a large and seemingly unpredictable change in the output digest. In an ideal hash function, such a change causes approximately half of the output bits to flip. This property follows naturally from preimage and collision resistance, since outputs must appear unpredictable and uncorrelated with their inputs, which forces even minimal input changes to propagate broadly throughout the digest. The avalanche effect is directly relevant to the tamper detection objective of this study because it implies that even a minor and deliberately subtle alteration to a single nutritional value in a TKPI record should produce a digest bearing no discernible resemblance to the original.

C. Merkle Tree

The Merkle tree, also referred to as a hash tree, is a method for efficiently verifying and authenticating large sets of data. Structurally, a Merkle tree is a tree in which every leaf node is labeled with the cryptographic hash of a data block and every non-leaf node is labeled with the cryptographic hash of the concatenation of its child nodes' labels, a process that continues until a single hash is obtained at the top of the tree; this top-level hash is known as the Merkle root.

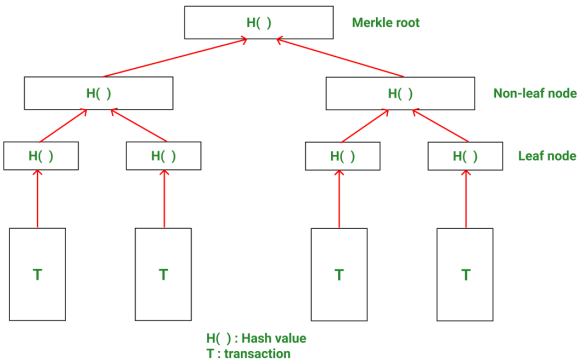

Fig. 3. Merkle Tree Structure
(Source: GeeksforGeeks)

The root represents a compact, fixed-size commitment to the entirety of the underlying dataset. Any modification to a single data block changes its leaf hash, and that change propagates upward until it changes the root, which allows tampering to be detected by comparing only the root values of two dataset versions rather than every individual record.

A key practical property of this structure is that locating which leaf differs between two trees requires processing an amount of data proportional to the logarithm of the number of leaves rather than to the total number of leaves. This logarithmic property forms the basis for the localization mechanism implemented in this study, in which a Merkle tree built over the hashed TKPI records is used to identify the

specific tampered food record without re-hashing the entire dataset from scratch.

D. Data Canonicalization

Because a hash function produces a digest that depends on the exact byte sequence of its input, applying it to structured tabular data such as TKPI records requires that each record first be transformed into a single, deterministic string representation prior to hashing. This process is commonly referred to as canonicalization.

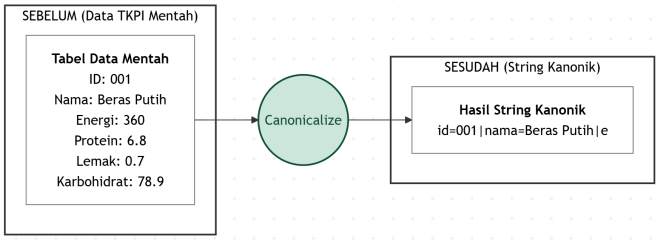

Fig. 4. Data Canonicalization Diagram

Without a fixed field order and consistent value formatting, two records that are semantically identical could be serialized differently, for example due to differing column order during data export, and would consequently produce different digests, leading to false-positive tampering alerts that do not reflect any actual change in the underlying data. In this study, canonicalization is implemented by concatenating each relevant attribute of a food record (identifier, name, and the four nutritional values) into a key-value formatted string using a fixed delimiter and a fixed column order, prior to hashing with each of the three algorithms described above.

E. Data Integrity versus Data Accuracy

A conceptual distinction must be drawn between data integrity and data accuracy, since the two are frequently conflated despite addressing different concerns. Data accuracy concerns whether a recorded nutritional value correctly reflects the true composition of a food item, a property that depends on sampling methodology, laboratory measurement, and analytical technique. Data integrity, in contrast, concerns whether a record, once published, remains unchanged as it is copied, transmitted, or stored across systems, regardless of whether the originally recorded value was itself accurate. The hash-based mechanism developed in this study addresses only the latter concern. It can confirm with very high probability that a record has not been silently altered after the fact, but it makes no claim regarding the correctness of the original recorded value. This distinction defines the precise scope of the system proposed in this study and is reflected in the Scope and Limitations stated in Chapter I.

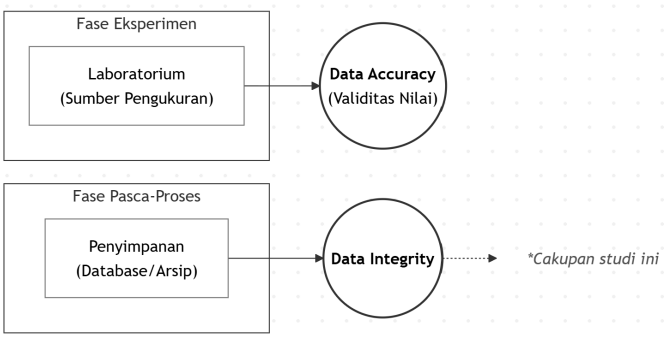

Fig. 5. Data Integrity and Data Accuracy Comparison

III. IMPLEMENTATION

A. System Architecture

The proposed tamper detection system consists of three functional modules implemented in Python 3: a canonicalization and hashing module (*hash_core.py*), an experiment runner (*run_experiment.py*), and a visualization module (*make_charts.py*). The system operates on the Indonesian food composition dataset sourced from Panganku, consisting of 1,346 records with six relevant attributes: *id, name, calories, proteins, fat*, and *carbohydrate*. The image field present in the raw dataset is excluded from the hashing process, since it contains a URL reference rather than a nutritional value and including it would not contribute to the integrity verification objective of this study.

B. Record Canonicalization

Each food record is first transformed into a single deterministic string before hashing, following the canonicalization principle described in Chapter II. This is implemented as follows:

```
def canonicalize_record(row, columns):
    parts = [f"{col}={row[col]}" for col in columns]
    return "|".join(parts)
```

Fig. 6. Canonicalize Record Function

Given a fixed column order [*id, name, calories, proteins, fat, carbohydrate*], this function produces a string such as *id=4|name=Akar tonjong segar|calories=45|proteins=1.1|fat=0.4|carbohydrate=10.8*. This string is encoded to UTF-8 bytes before being passed to the hash functions. As a result, the same logical record always produces an identical byte sequence regardless of how the underlying dataframe is loaded or ordered.

C. Hash Function Implementation

All three hash algorithms used in this study, MD5, SHA-256, and BLAKE2b, are implemented using Python's built-in hashlib module, which provides standards-compliant implementations of all three algorithms without requiring any third-party cryptographic library:

```python
HASH_FUNCS = {
    "MD5": lambda data: hashlib.md5(data).hexdigest(),
    "SHA-256": lambda data: hashlib.sha256(data).hexdigest(),
    "BLAKE2b": lambda data: hashlib.blake2b(data).hexdigest(),
}
```

Fig. 7. Hash Functions

Hashing an entire dataset is performed by iterating over every row, canonicalizing it, and computing its digest. This produces a list of hexadecimal digest strings, one per record, which serves as the input to both the tamper detection and Merkle tree mechanisms described below.

D. *Tampering Simulation*

To evaluate the system's detection capability under realistic conditions, a controlled tampering simulation is implemented. It randomly selects one record and one numeric nutritional field, then increases its value by a factor of 1.10 to 1.40 (a 10 to 40 percent inflation). This is meant to imitate a plausible data fraud or silent data-entry error scenario, such as a vendor inflating a reported nutrient value:

```python
def tamper_single_field(df, numeric_columns, seed=None):
    # simulasi tamper, naikin value 10-40% biar kerasa kayak fraud beneran
    rng = random.Random(seed)
    df_tampered = df.copy(deep=True)

    row_idx = rng.randrange(len(df_tampered))
    col = rng.choice(numeric_columns)

    original_value = df_tampered.at[row_idx, col]
    factor = rng.uniform(1.10, 1.40)
    new_value = round(float(original_value) * factor, 1) if original_value != 0 else round(rng.uniform(1, 10), 1)
    df_tampered.at[row_idx, col] = new_value

    return df_tampered, TamperResult(row_idx, col, original_value, new_value)
```

Fig. 8. Tampering Function

A fixed random seed is used for each experiment so that results can be reproduced across multiple runs.

E. *Merkle Tree Construction*

The Merkle tree is implemented as a class that takes a list of leaf hashes and combines adjacent pairs by concatenating and re-hashing them, repeating this step level by level until a single root hash remains:

```python
def _build(self):
    current_level = self.leaves
    while len(current_level) > 1:
        next_level = []
        for i in range(0, len(current_level), 2):
            left = current_level[i]
            right = current_level[i + 1] if i + 1 < len(current_level) else left
            next_level.append(self._combine(left, right))
        self.levels.append(next_level)
        current_level = next_level
```

Fig. 9. Merkle Tree Building Function

For an odd number of nodes at any level, the final node is duplicated. This is a standard way of handling unbalanced trees. The class also implements *find_changed_leaves()*, which compares the leaf hashes of two tree instances one by one and returns the indices at which they differ. This is the mechanism used to localize a tampered record without recomputing every hash from scratch.

F. *Performance Measurement*

Hashing performance is measured by timing the hash computation of the entire dataset, and scaled versions of it, using Python's *time.perf_counter()*. Each measurement is averaged over five repeated runs per algorithm to reduce the effect of temporary system load:

```python
def time_hashing(df, columns, algorithm, repeats=5):
    # rata2 dari beberapa run biar ga fluktuatif hasilnya
    durations = []
    for _ in range(repeats):
        start = time.perf_counter()
        hash_dataframe(df, columns, algorithm)
        durations.append(time.perf_counter() - start)
    return sum(durations) / len(durations)
```

Fig. 10. Time Hashing Function

Four dataset scales are tested: a 100-record subset, a 500-record subset, the full 1,346-record dataset, and a 2,692-record set produced by duplicating the full dataset. This allows performance trends to be observed as data volume increases.

IV. Testing

A. *Testing Methodology*

Three experiments were designed to evaluate the system against the research objectives stated in Chapter I: tamper detection capability, computational performance across dataset scales, and Merkle tree tamper localization. All experiments were run using the full 1,346-record TKPI-derived dataset described in Chapter III. Each experiment was repeated across different machines to check how stable the findings are.

B. *Tamper Detection Results*

In this experiment, a single record was selected and one of its numeric fields was modified by a randomly chosen factor between 10 and 40 percent. Both the original and tampered records were canonicalized and hashed using all three algorithms, and the resulting digests were compared. The results are summarized in Fig. 11

algorithm	hash_original	hash_tampered	detected
MD5	8a17799e8d13cc6e7cbe1d6c5f9a0382	c05b3b429e80b6f7f7a035abfd921a0e	True
SHA-256	03d0bd4ab2fee57365ad945e5f9b5d366a5a9cb9...	c051ab79359684f7a42fb66b9154e9826928814 2...	True
BLAKE2b	a097773444ae02549812defff28dea9ce93560165...	3e29a78d4f496fc59b4defef211112cc985f6cb80...	True

Fig. 11. Tamper Detection Results

All three algorithms produced a completely different digest after tampering. No partial similarity could be seen between the original and tampered hash values even when truncated for visual inspection. This shows that even a single-field modification, well below what would be noticeable from casually scrolling through a large spreadsheet, can be clearly detected through hash comparison, no matter which of the three algorithms is used.

To check this property at a finer level, an additional test was run in which a single nutritional value was changed by only 0.1 gram, the smallest realistic unit of change for this dataset. The resulting bit-level avalanche effect came out to approximately 58.6% for MD5, 50.0% for SHA-256, and 49.0% for BLAKE2b. This means SHA-256 and BLAKE2b sit closer to the theoretical ideal of 50% than MD5 does,

which lines up with MD5's known weaknesses compared to newer hash functions.

C. Performance Results

Hashing performance was measured across four dataset scales (100, 500, 1,346, and 2,692 records), with results shown in Fig. 12 and Fig. 13.

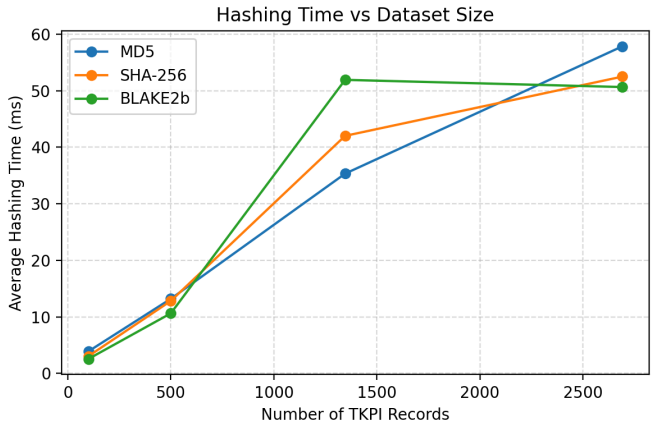

Fig. 12. Hashing Time vs Dataset Size

As Fig. 1X shows, hashing time grows roughly in a straight line as the number of records increases, for all three algorithms. This matches the expected linear (O(n)) computational cost of hashing a fixed-size dataset record by record. At the smaller dataset scales (100 and 500 records), the three algorithms perform about the same, with differences in average hashing time usually within one to two milliseconds of each other.

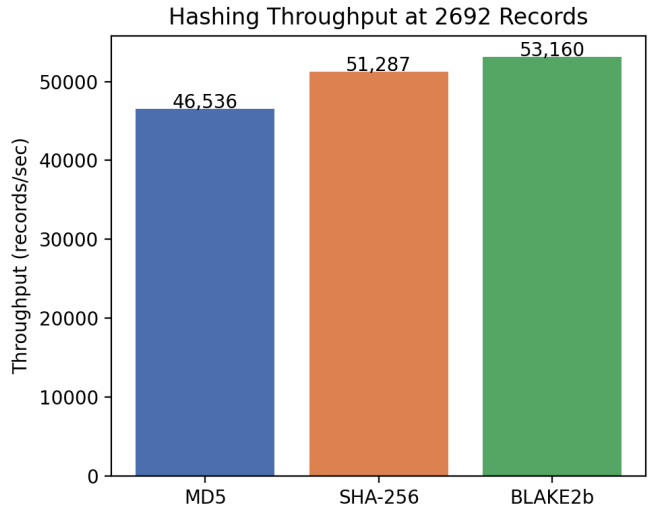

Fig. 13. Hashing Throughput at largest tested scale

At larger scales, which algorithm came out fastest varied depending on the machine the test was run on. Across repeated runs on different computers, no single algorithm was consistently ahead in throughput. Observed differences between MD5, SHA-256, and BLAKE2b at the full dataset scale were generally within about 5 to 10 percent of each other, which falls within the range that can be explained by ordinary system-level timing fluctuation (background processes, CPU scheduling, and so on) rather than a real, hardware-independent speed advantage for any one algorithm. This finding matters on its own: at the dataset scale relevant to TKPI (around 1,000 to 3,000 records), the choice between MD5, SHA-256, and BLAKE2b makes little practical difference to processing time. This means algorithm selection for this use case should be based mainly on security, not on speed.

D. Merkle Tree Localization Results

A Merkle tree was built over the SHA-256 hashes of all 1,346 records, producing a tree with 12 levels (from 1,346 leaf nodes down to a single root). After simulating a single-field tampering event, the tampered dataset's Merkle tree was rebuilt and its root hash compared against the original. The two root hashes were completely different, confirming detection. The *find_changed_leaves()* comparison correctly pointed to the exact index of the tampered record, without needing to re-hash and compare all 1,346 records individually. This shows the localization benefit described in Chapter II in practice: finding the tampered leaf required walking through a tree of depth 12 (matching $\lceil log_2(1346) \rceil = 11$, plus the leaf level itself), instead of running 1,346 separate comparisons.

E. Discussion

The experimental results support three conclusions relevant to the research objectives. First, all three hash algorithms reliably detect tampering of TKPI nutritional records, which satisfies the core requirement of the proposed system. Second, the avalanche effect measurements give a hardware-independent, repeatable basis for comparing the three algorithms on security grounds. This is unlike the throughput measurements, which turned out to be sensitive to the testing environment and should therefore be read with caution rather than treated as a final performance ranking. Third, the Merkle tree successfully extends the system from simple detection to efficient localization, which matters in particular for a dataset the size of TKPI, where checking every record by hand to find a single tampered value would not be practical.

V. Conclusion

A. Summary of Findings

This study designed, implemented, and tested a cryptographic hash-based tamper detection system applied to the Indonesian Food Composition Table (TKPI), made up of 1,346 food records. Three hash algorithms, MD5, SHA-256, and BLAKE2b, were compared, and a Merkle tree structure was implemented as an additional layer to enable efficient localization of tampered records. The experimental results show that all three algorithms detect single-field tampering of nutritional values, including changes as small as 0.1 gram, confirming that hash-based mechanisms work well as a practical foundation for nutritional data integrity verification. Avalanche effect measurements showed that SHA-256 (50.0%) and BLAKE2b (49.0%) flip close to half their output bits as expected, while MD5 (58.6%) is further from this ideal,

matching its known weaker cryptographic properties. Performance measurements showed that, at the dataset scale relevant to TKPI, throughput differences between the three algorithms are small and depend more on the testing environment than on any one algorithm being consistently faster. The Merkle tree successfully localized a tampered record within a tree of 12 levels for the full dataset, avoiding the need for a full linear comparison.

B. Recommendations

Based on these findings, SHA-256 is recommended as the preferred algorithm for applications where security and standardization matter most, given its near-ideal avalanche behavior and its status as a NIST-approved federal standard. MD5 is not recommended for new tamper-detection systems despite its competitive speed, because of its weaker avalanche characteristics and its well-documented collision vulnerabilities reported in prior cryptographic research. BLAKE2b is a reasonable alternative where implementation flexibility and a more modern design are valued, though this study did not find a consistent throughput advantage over SHA-256 at the scale tested. For practical use in monitoring national nutrition data, such as the data referenced by the Free Nutritious Meal Program (MBG) discussed in Chapter I, combining a SHA-256-based hashing scheme with a Merkle tree structure is recommended, since this combination provides both reliable tamper detection and efficient localization without needing to re-verify the entire dataset each time.

C. Limitations

This study has several limitations. The dataset used contains only four nutritional attributes (calories, protein, fat, carbohydrate), a smaller set than the roughly 21 nutrient fields present in the complete official TKPI publication. This means the findings should be read as a proof of concept on a simplified version of TKPI, not the full official table. Tampering simulations were carried out in a controlled experimental setting using synthetic changes, not data taken from an actual real-world attack. Performance measurements depend heavily on the hardware and runtime environment they are run on, so the throughput figures reported in this study should not be treated as fixed, hardware-independent benchmarks. Finally, this study only addresses data integrity (detecting unauthorized changes to data that has already been published) and does not address data accuracy (whether the originally published nutritional values are correct in the first place), nor does it address physical food safety. Both remain open problems for the wider national nutrition data ecosystem.

D. Future Work

Future research could build on this work in a few directions: applying the proposed mechanism to the complete TKPI dataset with its full set of around 21 nutritional attributes; adding digital signatures alongside hashing to verify the authenticity of the data's source, not just its integrity since publication; running performance benchmarks on standardized, dedicated hardware to get more consistent and comparable throughput numbers; and testing real-world deployment of the Merkle tree mechanism as part of a data distribution pipeline for programs such as MBG, where nutritional reference data is sent out to many independent endpoints.

Source Code

The complete implementation source code developed in this study are openly accessible in the public GitHub software repository at: https://github.com/thaffariq/NutriHash

Acknowledgment

The author expresses their deepest gratitude to Allah SWT, by whose will alone the completion of this paper was made possible. Sincere thanks are also extended to Dr. Ir. Rinaldi Munir, M.T., the instructor for the II4021 Cryptography course, for the invaluable guidance, direction, and insights provided throughout the semester, which greatly contributed to this work. Lastly, the author is deeply grateful to their parents and friends for their constant support and encouragement during the drafting process.

References

[1] Badan Gizi Nasional, "Awal 2026, Program MBG Jangkau Hampir 60 Juta Penerima Manfaat," Jan. 2026. [Online]. Available: https://www.bgn.go.id/news/siaran-pers/awal-2026-program-mbg-jangkau-hampir-60-juta-penerima-manfaat

[2] Badan Gizi Nasional, "Menu MBG Disusun Sesuai Standar AKG Kemenkes," 2026. [Online]. Available: https://www.bgn.go.id/news/berita/menu-mbg-disusun-sesuai-standar-akg-kemenkes

[3] N. Saputra et al., "Keracunan Massal pada MBG: Akibat Aturan Keamanan Pangan Hanya Formalitas?," The Conversation Indonesia, May 2026. [Online]. Available: https://theconversation.com/keracunan-massal-pada-mbg-akibat-aturan-keamanan-pangan-hanya-formalitas-277230

[4] IDN Times, "JPPI Catat 37.270 Korban Keracunan MBG Sejak Januari 2025–Mei 2026," Jun. 2026. [Online]. Available: https://www.idntimes.com/news/indonesia/jppi-catat-37-270-korban-keracunan-mbg-sejak-januari-2025-mei-2026-00-fcqw5-09484p

[5] Kementerian Kesehatan Republik Indonesia, Tabel Komposisi Pangan Indonesia. Jakarta, Indonesia: Kementerian Kesehatan RI, 2020. [Online]. Available: https://repository.kemkes.go.id/book/668

[6] Badan Pangan Nasional, Laporan Kegiatan Kajian Komposisi Gizi Pangan Segar 2025. Jakarta, Indonesia: Badan Pangan Nasional, 2025. [Online]. Available: https://badanpangan.go.id/storage/app/media/2025/BARANG%20DAN%20JASA/Bu%20Ega/LAPORAN%20KEGIATAN%20KAJIAN%20KOMPOSISI%20PANGAN%20SEGAR%202025_data%20final.pdf

[7] A. F. Hanif, "Indonesian Food and Drink Nutrition Dataset," Kaggle, 2023. [Online]. Available: https://www.kaggle.com/datasets/anasfikrihanif/indonesian-food-and-drink-nutrition-dataset

[8] R. C. Merkle, "Secrecy, Authentication, and Public Key Systems," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, 1979. [Online]. Available: https://cir.nii.ac.jp/crid/1571417125520618240

[9] R. C. Merkle, "One Way Hash Functions and DES," in Advances in Cryptology—CRYPTO '89 Proceedings, Lecture Notes in Computer Science, vol. 435. Berlin, Germany: Springer, 1990, pp. 428–446. [Online]. Available: https://doi.org/10.1007/0-387-34805-0_40

[10] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," in Fast Software Encryption (FSE 2004), Lecture Notes in Computer Science, vol. 3017. Berlin, Germany: Springer, 2004, pp. 371–388. [Online]. Available: https://doi.org/10.1007/978-3-540-25937-4_24

[11] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Internet Engineering Task Force (IETF), Apr. 1992. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1321.txt

[12] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," Federal Information Processing Standards Publication (FIPS PUB) 180-4, Aug. 2015. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.180-4
[13] M.-J. O. Saarinen and J.-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)," RFC 7693, Internet Engineering Task Force (IETF), Nov. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7693.txt
[14] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in Advances in Cryptology—EUROCRYPT 2005, Lecture Notes in Computer Science, vol. 3494. Berlin, Germany: Springer, 2005, pp. 19–35. [Online]. Available: https://doi.org/10.1007/11426639_2
[15] Investor Trust, "BGN Targetkan 32.000 SPPG dan 82,9 Juta Penerima Manfaat MBG pada April 2026," Jan. 2026. [Online]. Available: https://investortrust.id/national/91372/

STATEMENT

I hereby declare that this paper is my own original work, and that it is not an adaptation, translation, or plagiarism of anyone else's work.

Bandung, June 19th 2026

Thaffariq Azka Rahmat
NIM 18223048