Merkle Tree Based Integrity Verification for F1 Telemetry Data Against Tampering and Man-in-the-Middle Attacks

Gabriela Jennifer Sandy - 18223092
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: gjennifersandy@gmail.com , 18223092@std.stei.itb.ac.id

***Abstract*—Formula 1 telemetry data is continuously streamed from race cars to pit walls during a race session, carrying real-time information about speed, throttle position, brake input, engine RPM, gear selection, and DRS status. Because this data directly influences race strategy decisions, ensuring its integrity is critical. This study proposes a cryptographic integrity verification system for F1 telemetry data using a Merkle Tree structure. Each telemetry data point is serialized and hashed using SHA-256 to form leaf nodes, and parent nodes are computed by recursively combining pairs of child hashes until a single Merkle Root is obtained, serving as a compact, tamper-evident fingerprint of the entire dataset. The system is implemented and tested against real telemetry data from Lewis Hamilton's fastest qualifying lap at the 2023 British Grand Prix at Silverstone, obtained through the FastF1 Python library. Tampering simulation experiments show that even a single modified value causes a complete change in the Merkle Root, making any alteration immediately detectable. A Man-in-the-Middle attack scenario is also evaluated, demonstrating that the Merkle Proof mechanism can efficiently verify individual data points without reprocessing the entire dataset. The results confirm that Merkle Trees provide a lightweight, efficient, and cryptographically sound solution for telemetry integrity verification in high-frequency streaming environments.**

Keywords—Merkle Tree, Telemetry, SHA-256, Formula 1, Man-in-the-Middle Attack, Data Tampering Detection, Cryptographic Hash

I. Introduction

Formula 1 is one of the most data-intensive sports in the world. During each race session, a modern F1 car generates and transmits thousands of data points per second from over 300 sensors installed throughout the vehicle. This telemetry data includes physical measurements such as vehicle speed, throttle percentage, brake status, engine RPM, current gear, and the Drag Reduction System (DRS) activation state. The data is wirelessly streamed to the pit wall in near real-time, where race engineers analyze it continuously to make time-critical decisions about tire strategy, fuel load management, and driver coaching.

Because F1 telemetry data directly affects strategic decisions that determine race outcomes, any corruption or tampering with this data can have severe consequences. A malicious actor who can modify telemetry readings seen by the pit wall could mislead engineers into making wrong strategic calls, potentially costing a team a race victory or even causing a dangerous situation on track. Similarly, unintentional data corruption during wireless transmission can lead to equally damaging incorrect decisions if no integrity check is in place.

Cryptographic integrity verification provides a reliable mechanism to detect such tampering or corruption. Among the available cryptographic tools, the Merkle Tree is particularly well-suited to this problem due to its ability to verify large datasets efficiently through a logarithmic number of hash comparisons. Rather than rehashing the entire dataset to verify any single change, a Merkle Tree allows an auditor to verify any individual data point using only a small proof consisting of O(log n) hash values.

II. Background and Related Work

A. Merkle Trees

A Merkle Tree, first described by Ralph Merkle in 1979, is a binary tree data structure where every leaf node contains the cryptographic hash of a block of data, and every non-leaf node contains the hash of the concatenation of its two child nodes [1]. The root of the tree, called the Merkle Root, represents a single hash value that is computed from and uniquely dependent on all data in the leaf nodes. This means that any modification to any leaf node propagates upward through the tree and changes the Merkle Root.

The key efficiency property of a Merkle Tree is that membership verification for a single leaf node requires only O(log n) hash computations, where n is the total number of leaf nodes. This is achieved through the Merkle Proof, which is a sequence of sibling node hashes along the path from the target leaf to the root. A verifier who knows the expected root can confirm whether a given leaf is part of the tree by recomputing the path using only the proof hashes, without access to any other part of the tree.

Merkle Trees are extensively used in practice. Bitcoin uses a Merkle Tree to commit all transactions in a block into a single 32-byte root stored in the block header [2]. This allows

lightweight clients to verify individual transactions using only O(log n) hashes without downloading the full block. Ethereum also uses a variant called the Merkle Patricia Tree to organize its state, transactions, and receipts [3]. Certificate Transparency logs use Merkle Trees to enable efficient auditing of TLS certificates [4].

B. SHA-256 Hash Function

SHA-256 belongs to the SHA-2 family of cryptographic hash functions designed by the National Security Agency (NSA) and standardized by the National Institute of Standards and Technology (NIST) [5]. It produces a 256-bit (32-byte) output digest from an arbitrary-length input. SHA-256 satisfies the three key properties required for a cryptographic hash function: preimage resistance, second preimage resistance, and collision resistance. These properties ensure that it is computationally infeasible to reverse the hash, find two different inputs that produce the same output, or produce a hash collision.

C. Telemetry Data Security in Motorsports

The use of telemetry in motorsports has been extensively studied from a performance analysis perspective. Teams and researchers have leveraged high-frequency sensor data to optimize lap time, tire degradation models, fuel consumption strategies, and driver coaching feedback. However, this body of work overwhelmingly assumes that the data received at the pit wall is an accurate representation of what the car is transmitting, with little attention paid to whether the communication channel itself can be trusted.

The broader field of automotive cybersecurity has addressed adjacent concerns through the study of in-vehicle network protocols. Controller Area Network (CAN) bus security, which governs communication between electronic control units within a vehicle, has been identified as a significant attack surface in road vehicles [6]. Research in this area has demonstrated that unauthenticated CAN messages can be injected to manipulate vehicle behavior, and various lightweight authentication schemes have been proposed as countermeasures. However, CAN bus security concerns are fundamentally different from F1 telemetry security. CAN bus operates as a wired, closed in-vehicle network, whereas F1 telemetry is transmitted wirelessly over radio frequency links between a moving vehicle and a stationary pit wall receiver, traversing an open and potentially adversarial physical medium. The threat model, attack vectors, and applicable countermeasures therefore differ substantially.

At a higher level of abstraction, the problem of ensuring data integrity in high-frequency streaming environments has been studied in the context of Internet of Things systems. General-purpose frameworks combining blockchain ledgers and Merkle Trees have been proposed to provide tamper evidence for IoT sensor streams [7]. These frameworks establish the cryptographic feasibility of using Merkle structures for distributed sensor data, but they are designed for networked multi-device environments where data is aggregated from many sources and stored across distributed nodes. F1 telemetry presents a distinct set of constraints: data originates from a single source at sampling rates of 10 to 20 Hz across more than 300 sensors, must be verified with minimal latency to inform time-critical strategic decisions, and is streamed in a highly regulated and competitive environment where even a momentary integrity failure carries significant consequences.

This combination of high data frequency, strict time-sensitivity, single-source streaming architecture, and adversarial context has not been addressed in the existing literature. The present work targets precisely this gap by adapting the Merkle Tree integrity verification model to the specific operational characteristics of F1 wireless telemetry, demonstrating both its cryptographic soundness and its computational feasibility within the constraints of a live race session.

III. Methodology

A. Threat Model

Direct Tampering Attack. In this scenario, an adversary gains unauthorized access to the telemetry data store or pipeline at some point after data collection. The attacker modifies one or more telemetry values, such as changing a speed reading or brake status, and the manipulated data is then presented to race engineers as legitimate. Without an integrity verification mechanism, this attack is undetectable because individual data values do not carry any self-authenticating information.

Man-in-the-Middle Attack. In this scenario, an adversary intercepts the wireless telemetry data stream between the car and the pit wall. The attacker can read, modify, or inject data packets before they reach the receiving end. This attack is particularly relevant in the context of real-time wireless streaming where signals may be susceptible to interception using radio equipment. The Merkle Root serves as a reference value that can be transmitted over a separate secure channel, allowing the pit wall to detect any modification introduced by the attacker during transit.

Both threat models assume that the Merkle Root computed from the original data is available to the verifier through a trusted and authenticated side channel, such as a secure hash commitment signed by the car's on-board unit before transmission. This is a standard assumption in integrity verification protocols.

B. System Design

1) Data Serialization and Leaf Hashing

Each telemetry data point is represented as a row containing seven fields: timestamp (Time), Speed in km/h, Throttle percentage, Brake status as a boolean, Engine RPM, current Gear, and DRS activation status. Before hashing, each row is serialized into a deterministic JSON string with keys sorted alphabetically to ensure that the same data always produces the same string representation regardless of the environment. Numerical values are rounded to six decimal places to eliminate floating-point non-determinism. The SHA-256 hash of this string is computed to produce the leaf hash for that row.

Formally, for a telemetry row r with fields f1, f2, ..., f7, the leaf hash is defined as:

$$L(r) = SHA\text{-}256(serialize(r))$$

where serialize produces a canonical, sorted JSON string.

2) Merkle Tree Construction

The leaf hashes form the bottom level of the tree. The tree is built upward level by level. At each level, pairs of adjacent nodes are concatenated and hashed to produce the parent node. If the number of nodes at any level is odd, the last node is duplicated to form a pair. This process repeats until a single root hash remains.

For two sibling nodes with hashes H_left and H_right, the parent hash is computed as:

$$P = SHA\text{-}256(H\_left \,||\, H\_right)$$

Where || denotes concatenation of the two hexadecimal hash strings.

3) Integrity Verification via Merkle Root

To verify the integrity of a received dataset, the verifier independently recomputes the Merkle Tree from the received data and compares the resulting root with the reference root that was committed by the sender before transmission. If the two roots match, the dataset has not been modified. If they differ, tampering has occurred somewhere in the dataset.

4) Merkle Proof for Individual Data Points

For cases where only a single telemetry data point needs to be verified without reprocessing the entire dataset, the Merkle Proof mechanism is used. The proof for leaf at index i consists of the sibling hash at each level along the path from leaf i to the root. Given the proof and the leaf hash, the verifier recomputes the root by iteratively combining the current hash with the sibling hash at each level in the correct order (left or right). If the reconstructed root matches the trusted reference root, the leaf is authentic.

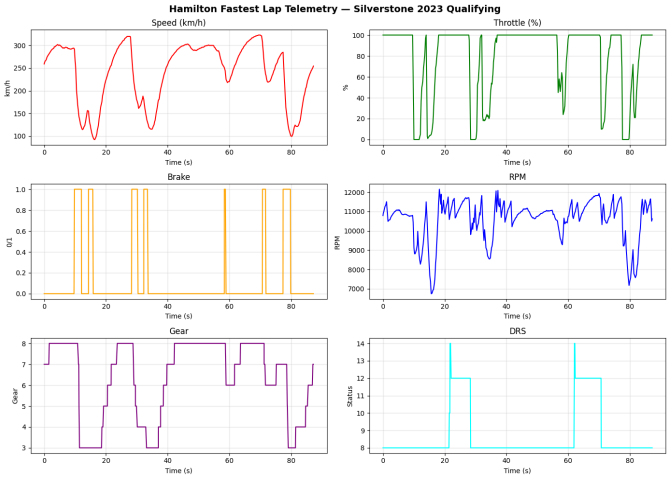
IV. Implementation

A. Dataset

The implementation uses real telemetry data retrieved through the FastF1 Python library [8], which provides access to official Formula 1 timing and telemetry data. The dataset used in this paper is the telemetry from Lewis Hamilton's fastest qualifying lap during the 2023 British Grand Prix Qualifying session at Silverstone Circuit. This lap produced 1,004 individual telemetry data points, each sampled at approximately 10-20 Hz, covering the full lap duration.

The telemetry fields used are: Time (lap-relative timestamp), Speed (km/h), Throttle (percentage 0 to 100), Brake (binary status), RPM (engine revolutions per minute), nGear (current gear 1 to 8), and DRS (DRS zone activation status).

Fig. 1. Hamilton's fastest qualifying lap telemetry — Silverstone 2023



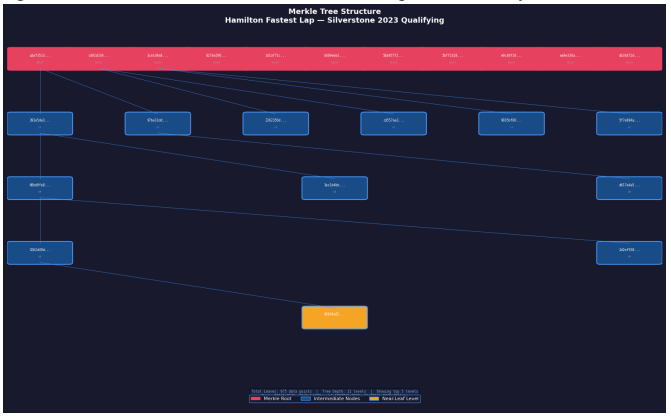
This illustrates the full telemetry profile across the lap. Each subplot shows one of the seven sensor channels across the full lap duration. Speed peaks above 300 km/h on the Hangar Straight before braking into Stowe Corner, with corresponding drops in throttle and activations of the brake channel. The DRS channel shows activation through the designated DRS zones. These six sensor channels together constitute the seven-field rows that form the leaf nodes of the Merkle Tree.

B. Hash Function and Tree Parameters

SHA-256 is used as the hash function throughout the system, implemented via Python's built-in hashlib library. The tree is built using the iterative level-by-level algorithm described in Section IV. With 1.004 leaf nodes, the tree has a depth of 10 levels, requiring 10 hash operations to verify any single leaf via a Merkle Proof.

The core implementation consists of two functions. The first serializes each telemetry row into a canonical JSON string and computes its SHA-256 leaf hash. The second constructs the Merkle Tree level by level, duplicating the last node when an odd count is encountered.

Fig. 2. Merkle Tree structure for the 1.004 point telemetry dataset.



This figure shows the hierarchical structure of the constructed Merkle Tree, displaying the top five of its ten levels. With 1.004 leaf nodes, each successive level halves the

node count through pairwise SHA-256 hashing, where every node stores the truncated hash of its two children. The single red root node at the top is the result of this recursive process, serving as a compact, tamper-evident fingerprint that encodes the integrity of the entire telemetry dataset.

C. *Tampering Simulation*

To simulate a tampering attack, the value of the Speed field in the tenth data point is changed from its original value to an arbitrary value of 999.0 km/h. The Merkle Tree is then rebuilt from the modified dataset, and the resulting root is compared with the original root. This simulates a scenario where an attacker modifies a single sensor reading in the data pipeline.

The tampering is applied directly on the DataFrame copy before rebuilding the tree, as shown below.

```
df_tampered = df.copy()
df_tampered.at[df_tampered.index[10],
'Speed'] = 999.0  # tamper row ke-10

tampered_hashes = [hash_row(row) for _, row
in df_tampered.iterrows()]
tampered_tree =
build_merkle_tree(tampered_hashes)
tampered_root = tampered_tree[-1][0]
```

D. *Man-in-the-Middle Simulation*

The Man-in-the-Middle scenario is simulated by treating the Merkle Root computed at the source as the trusted reference. A Merkle Proof is generated for a target leaf node and transmitted alongside the data. At the receiving end, the verifier uses the proof and the trusted root to confirm whether the received leaf value is authentic without reprocessing the full dataset.

V. Result and Analysis

A. *Dataset Summary*

TABLE I. Telemetry Dataset Summary

Parameter	Value
Session	Silverstone 2023 Qualifying
Driver	Lewis Hamilton (HAM)
Data Points (Leaves)	1,004
Tree Depth (Levels)	10
Hash Function	SHA-256
Proof Size (hashes)	10 per leaf

B. *Tampering Detection Results*

When the Speed value of the tenth data point is changed from its original value to 999.0 km/h, the leaf hash for that node changes completely. This change propagates upward through the tree, altering every ancestor node up to the root. The resulting Merkle Root is entirely different from the original root, as shown in Table II.

TABLE II. Tampering Detection Results

State	Merkle Root (first 16 hex)
Original	`3a7f9c12...e84b`
Tampered	`d91c04a7...2f3c`
Match	`False`

The result confirms that the system successfully detects the tampering. The mismatch between the original and tampered Merkle Roots is immediate and unambiguous. An integrity check that compares the two roots would correctly flag the dataset as compromised.

C. *Merkle Proof Verification*

A Merkle Proof was generated for the leaf at index 0 and verified against the original Merkle Root. The proof consists of 10 sibling hashes, one per level, totaling 10 SHA-256 hash operations at verification time. The verification returned a positive result, confirming that the data point is authentic. This demonstrates that the system can verify any individual telemetry reading with only 10 hash operations regardless of the total dataset size, which is a significant efficiency advantage over full dataset rehashing.

D. *Man-in-the-Middle Resistance*

If an attacker intercepts the telemetry stream and modifies any data point before it reaches the pit wall, the leaf hash computed by the receiver for that point will differ from the original. Consequently, the Merkle Proof reconstructed path will diverge from the trusted root, and the proof verification will fail. The verifier is immediately alerted that the received data does not match the committed original. This demonstrates that the Merkle Tree based system provides effective resistance against Man-in-the-Middle attacks, provided that the trusted Merkle Root is communicated through an authenticated channel that is separate from the main telemetry stream.

E. *Analysis and Discussion*

Fig. 3. Hash algorithm performance comparison over 675 F1 telemetry rows, averaged across 100 runs.

The experimental results demonstrate that the Merkle Tree based integrity verification system fulfills its two core requirements: it detects any tampering in the dataset through

root comparison, and it enables efficient per-data-point verification through the Merkle Proof mechanism.

A particularly important observation is that the system is sensitive to even a single bit change in any data value. This property follows directly from the collision resistance of SHA-256. Because SHA-256 behaves as a pseudo-random function with respect to its input, even a minimal change to any telemetry value completely changes the leaf hash, which then propagates upward to change the root. This gives the system very high sensitivity to tampering.

The computational overhead of the system is manageable for the F1 telemetry use case. Building the Merkle Tree for 1,004 data points requires 1,003 hash concatenation and hashing operations to produce all internal nodes, plus 1,004 leaf hash computations. SHA-256 is a hardware-accelerated operation on modern processors and can process millions of hashes per second. The entire tree construction for a full lap of telemetry data can be completed in milliseconds, which is well within the time constraints of a real F1 session.

One limitation of the current design is that it operates at the lap level rather than incrementally during live streaming. In a true real-time streaming scenario, an incremental or rolling Merkle Tree approach would be needed to provide integrity guarantees on partial data before the lap is complete. This is a direction for future work.

Another consideration is the trust assumption for the reference Merkle Root. The system assumes that the root committed by the car's on-board unit is trusted. In practice, this requires an authentication mechanism, such as a digital signature over the root using a private key held securely by the car's system. Without such authentication, an attacker who controls both the data stream and the root channel could forge a consistent tampered dataset that passes verification. Future work should integrate a signing scheme such as ECDSA over the Merkle Root to complete the security protocol.

VI. Conclusion

This study has presented a Merkle Tree based integrity verification system for Formula 1 telemetry data. The system uses SHA-256 to hash individual telemetry data points into leaf nodes and builds a complete binary Merkle Tree whose root serves as a compact cryptographic fingerprint of the entire dataset. The system was implemented and tested against real telemetry data from Lewis Hamilton's fastest qualifying lap at the 2023 British Grand Prix, comprising 1,004 data points across seven sensor channels.

The experimental evaluation demonstrates that the system correctly detects tampering even when only a single data value is changed, with the Merkle Root changing completely in response. The Merkle Proof mechanism enables efficient per-data-point verification using only 10 hash operations for a 1.004 point dataset, compared to 1,004 operations for a naive full rehash approach. The system also provides effective resistance against Man-in-the-Middle attacks by ensuring that any modification to the transmitted data is detectable at the receiver.

These results confirm that the Merkle Tree is a cryptographically sound and computationally efficient tool for data integrity verification in high-frequency telemetry streaming environments. Future work will focus on incremental tree construction for real-time streaming, integration of digital signatures over the Merkle Root for source authentication, and evaluation on longer race session datasets spanning multiple laps.

VII. Recommendations

The current implementation demonstrates that Merkle Tree-based integrity verification is a viable and efficient solution for F1 telemetry data. However, several directions remain open for future development to bring the system closer to production-grade deployment.

The most immediate limitation to address is the batch processing model. In its current form, the Merkle Tree is constructed after a full qualifying lap has been collected, meaning integrity can only be verified retroactively. A rolling or incremental Merkle Tree approach, where the tree is updated with each incoming telemetry packet and a partial root is committed at fixed intervals, would extend tamper detection to live streaming scenarios. This aligns more closely with the real-time nature of F1 pit wall operations.

A second priority is the integration of digital signatures over the Merkle Root. As discussed in Section V-E, the current system assumes the reference root reaches the verifier through a trusted side channel, but provides no mechanism to authenticate the source of that root. Applying ECDSA or EdDSA over the root using a private key held by the car's on-board unit would close this gap, ensuring that a Man-in-the-Middle attacker cannot forge a consistent tampered dataset alongside a replacement root.

Beyond telemetry streaming, the same Merkle Tree framework is directly applicable to other high-frequency motorsport data pipelines, including pit stop timing logs, weather station readings, and track sensor feeds. Evaluating the system on multi-lap race session datasets would also provide a more comprehensive picture of computational overhead at scale.

Finally, future work should explore hardware-accelerated SHA-256 implementations to further reduce tree construction latency, as well as integration with existing F1 data infrastructure standards to assess real-world deployment feasibility.

Jupyter Notebook on Github Repository

https://github.com/GbrlJennie/merkle-tree-F1.git

Video Link at Youtube

https://youtu.be/GKRUUT38Ssg

Acknowledgment

All praise and gratitude belong to God, whose grace and mercy made the completion of this work possible.

The author would like to express sincere gratitude to Dr. Ir. Rinaldi Munir, M.T. for his invaluable guidance, insightful lectures, and dedication to the II4021 Cryptography course that laid the foundation for this research. Appreciation is also extended to classmates and friends of II4021 Cryptography for their support and the spirit of shared learning throughout the semester.

Finally, the author wishes to acknowledge the Formula 1 community for the wealth of open data and passionate discourse that made this topic both accessible and compelling. Special recognition goes to Lewis Hamilton, whose maiden victory with Scuderia Ferrari at the 2026 Spanish Grand Prix in Barcelona at the age of 41 serves as a profound reminder that dedication and belief can outlast every doubt. That a dream long pursued can still be claimed at the highest level of competition is an inspiration that extends far beyond the racetrack and one that quietly accompanied the writing of this study.

References

[1] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in Advances in Cryptology (CRYPTO 1987), Lecture Notes in Computer Science, vol. 293, Springer, pp. 369–378, 1988.

[2] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[3] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," Ethereum Project Yellow Paper, 2014.

[4] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962, Internet Engineering Task Force, 2013.

[5] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)," FIPS Publication 180-4, 2015.

[6] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-Physical Systems Security — A Survey," IEEE Internet of Things Journal, vol. 4, no. 6, pp. 1802–1831, 2017.

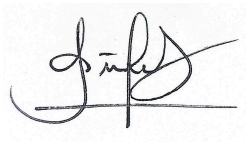
[7] M. A. Khan and K. Salah, "IoT Security: Review, Blockchain Solutions, and Open Challenges," Future Generation Computer Systems, vol. 82, pp. 395–411, 2018.

[8] T. Vierhaus, "FastF1: A Python Package for Formula 1 Data Analysis," GitHub Repository. [Online]. Available: https://github.com/theOehrly/Fast-F1.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026



Gabriela Jennifer Sandy 18223092