Enhancing Web Security and Performance with Hybrid Stateless Authentication

Benedictus Mario [1]*, Trianggoro Wiradinata [2]*, Christian [3]*
* School of Information Technology, Universitas Ciputra Surabaya
benedictusmario6@gmail com [1], twiradinata@ciputra.ac.id [2], christian03@ciputra.ac.id [3]

Article Info

Article history:

Received 2025-03-05
Revised 2025-03-18
Accepted 2025-03-26

Keyword:

Authentication,
Authorization,
Identity Access and Management,
Security,
Stateless Token Revocation

ABSTRACT

Ensuring operational integrity across industries and protecting sensitive data require strong authentication systems. This paper presents a novel hybrid stateless authentication method that integrates binary payloads, token specifications, and database solutions. By employing a distinctive expiration policy, our proposed approach overcomes limitations inherent in traditional token revocation strategies while achieving token verification speeds that are up to 86 times faster than conventional statefull session-based methods. Overall, through uniformed benchmarking experiments and a comprehensive review of the literature substantiate the performance and security advantages of our method. Ultimately, this hybrid technique offers a more scalable and secure framework for authentication management, enabling efficient and flexible deployment in high-demand distributed environments.

I. Introduction

Adopting strong security measures is essential for digital transformation to preserve operational integrity, safeguard sensitive data, and adhere to legal obligations. diverse industry-specific legislation handles the diverse issues that organizations encounter. For example, the Health Insurance Portability and Accountability Act (HIPAA) [1] regulates data protection in the healthcare industry, while the Payment Card Industry Data Security Standard (PCI DSS) is followed by the finance industry to protect sensitive financial data [2], [3]. To ensure data security, every website or app owner should pay attention to authentication and authorization which are fundamental components of Identity and Access Management (IAM), a critical framework in both cyber security and organizational governance. In earlier approaches, traditional authentication methods were significantly enhanced by modernized techniques, such as token-based authentication and cryptographic methods [4], [5]. Subsequently, the introduction of Single Sign-On (SSO) systems marked another pivotal advancement in this evolution. This approach improves user experience by reducing password fatigue and address the challenges associated with managing numerous accounts across different platforms, thereby mitigating security risks linked to the use of weak or recycled passwords [6], [7], [8]. To enable interoperability across service provider especially in distributed systems or microservices architectures, stateless tokens such as JSON Web Tokens (JWT), are commonly used in SSO frameworks because they allow for a simplified and scalable authentication process where the server does not need to maintain session state [9], [10], [11]. The term “JWT” typically refers to the JWS format, which provides a compact, URL-safe representation of claims that are digitally signed using algorithms such as HMAC or RSA, ensuring that the claims are both verifiable and trustworthy [12], [13], [14]. Moreover, studies have shown that employing JWT with HMAC significantly improves performance in terms of token generation time and transfer speed, making this approach particularly compelling for high-demand environments [15], [16].

However, using JWT in SSO systems presents several concerns that organizations must consider. One major issue is token expiration and revocation. Due to their stateless nature, once JWTs are issued, they remain valid until their expiration time. This characteristic makes it challenging to invalidate tokens immediately when a user's access needs to be revoked, such as in cases of account compromise. Moreover, JWTs can become large, especially when they contain numerous claims. This increase in size may lead to higher bandwidth usage and degraded performance, which is particularly problematic for mobile applications where network efficiency is critical [17].

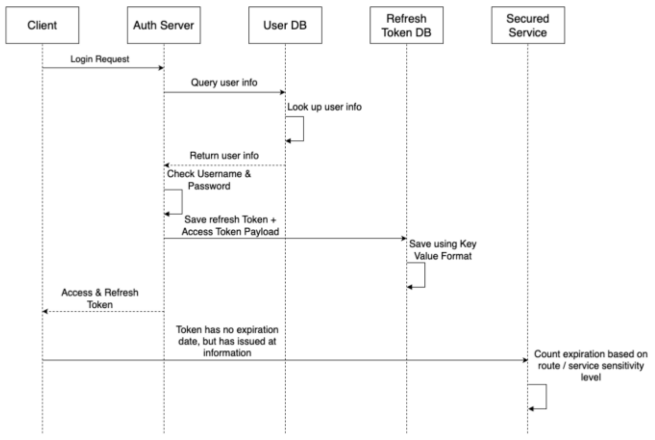
Client
Auth Server
User DB
Refresh Token DB
Secured Service
Login Request
Query user info
Look up user info
Return user info
Check Username & Password
Save refresh Token + Access Token Payload
Save using Key Value Format
Access & Refresh Token
Token has no expiration date, but has issued at information
Count expiration based on route / service sensitivity level

Figure 1. Authentication Flow from our proposed solution

Even though there are some challenges, we believe these problems are not caused by the design of JWT or other stateless token formats themselves. Instead, they arise from how these tokens are implemented and used in practice. For instance, researchers (e.g., [18], [19], [20]) have suggested various advanced methods to revoke JWT tokens when needed. However, feedback from another security experts (e.g., [21], [22], [23], [24], [25]) highlights a tricky situation with these solutions. On one hand, if you want to keep the system completely stateless (meaning no server-side storage of session data), you might have to compromise on security. On the other hand, if you want stronger security, you may need to introduce state management (like storing session data on the server), which takes away the main advantage of stateless authentication. This creates dilemma: should we prioritize security or stick to the benefits of statelessness? To address this, a new hybrid approach is needed. This approach would combine the best parts of both statefull and stateless authentications. By doing so, it could reduce extra work or overhead required for each request while also avoiding security caveats of stateless implementation. Figure 1 illustrates how this balance can be achieved.

II. TERMINOLOGIES

According to Jánoky et al. [18], [19], traditional methods for JWT revocation include the use of short-lived tokens, blacklisting, and changing the cryptographic secret employed to sign tokens. Short-lived tokens minimize the period during which a revoked token may be misused; however, this approach increases the frequency of token reissuance. Blacklisting, on the other hand, necessitates that every request check against a centralized list of invalidated tokens, thereby introducing latency and scalability challenges. For instance, each API call must query the blacklist, potentially incurring significant overhead in systems with a large user base and high request rates. Jánoky et al. propose an enhanced revocation strategy that extends the secret-change approach by partitioning the client population into groups and assigning a distinct signing secret to each group. For instance, if 100 clients are divided into 20 groups, a logout event in one group necessitates a secret change for only about five clients, thereby reducing unnecessary token revocations significantly. However, this proposed solution comes with the caveat of increased architectural complexity, as it requires managing multiple signing secrets and ensuring their synchronization across distributed services. Figure 2 visualize the revocation strategy from Jánoky et al.

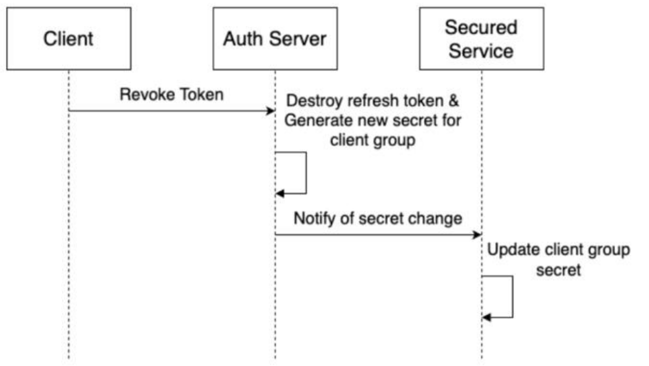
Client
Auth Server
Secured Service
Revoke Token
Destroy refresh token & Generate new secret for client group
Notify of secret change
Update client group secret

Figure 2. Revocation Strategy from Jánoky et al.

In contrast to the approach of Jánoky et al., Fugkeaw et al. [20] employ blacklisting as their method for token revocation. During token verification, the application server checks the presented token against a revocation list and immediately rejects the token if its ID is found. To enhance performance, Fugkeaw et al. implement a dynamic load balancing mechanism that manages both SSO token generation and verification requests across multiple servers. The system utilizes a multi-threaded algorithm that continuously monitors server load and health, routing requests to the server with the least current load. This strategy not only optimizes resource utilization but also ensures that the system can scale effectively in response to increasing authentication demands. Figure 3 visualize the revocation strategy from Fugkeaw et al.

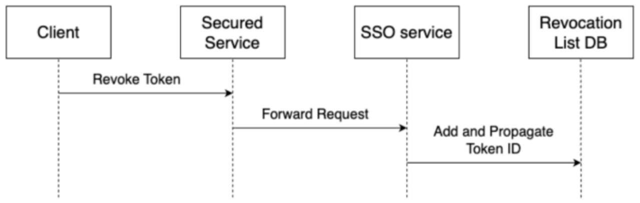

Figure 3. Revocation Strategy from Fugkeaw et al.

Different methods for revoking tokens have their own strengths and weaknesses, which makes it important to carefully evaluate them. However, we decided not to consider the approach by Jánoky et al., which involves changing cryptographic secrets for specific groups. This method can be inconvenient for users, as some may face issues when secrets are updated. Similarly, relying only on load-balancing strategies, like the one proposed by Fugkeaw et al., is not sufficient or cost-effective. If tokens are optimized for better performance, the common issue with short-lived tokens which is frequent reissuance could be reduced without adding extra costs. However, using a single, short expiration time for all tokens might be too strict and inefficient. For example, endpoints for deleting data are usually more sensitive than those for retrieving data. A standard expiration policy applies the same lifespan to all endpoints, but by using the "issued-at" timestamp, we can create more flexible expiration times. This involves combining the token's creation time with a specific expiration duration for each endpoint. As shown in Figure 4, revoking a token can be as simple as removing the refresh token from the database and deleting the access token from the client. However, in cases like account takeovers or token leaks, the token might remain valid until its expiration time. This limitation can be managed by setting appropriate expiration durations earlier.

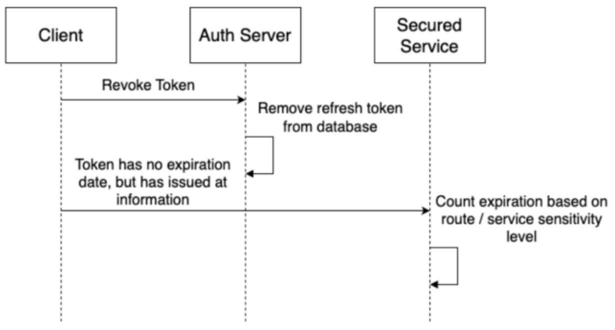
Client
Auth Server
Secured Service
Revoke Token
Remove refresh token from database
Token has no expiration date, but has issued at information
Count expiration based on route / service sensitivity level

Figure 4. Proposed Revocation Strategy

In this proposed method, Go programming language will be used as main programming language because it offers compelling advantages for backend service. It has an optimal balance of runtime performance, memory usage and ease of use compared to other languages like Java and C++ [26]. This efficiency is beneficial in cloud environments, where resource management is crucial [27]. For managing the state of refresh tokens, we choose Redis, which typically outperforms MongoDB due to its in-memory data store architecture that facilitates faster read and write operations. This makes Redis especially suitable for scenarios where rapid access to user data is crucial [28], [29]. Both MongoDB and Redis will be utilised as key value store, hence there won't be any SQL script or query to be compared. Furthermore, we have decided to incorporate Branca [30] into our proposed solution because it utilizes the XChaCha20-Poly1305 AEAD scheme. ChaCha20-Poly1305 is a widely recognized authenticated encryption with associated data (AEAD) algorithm that combines the ChaCha20 stream cipher for encryption with the Poly1305 MAC for authentication. XChaCha20-Poly1305 is essentially similar to ChaCha20-Poly1305, but it employs a larger nonce of 192 bits (24 bytes) [31], [32], [33], [34], [35]. Its design emphasizes both performance and security, particularly in environments where traditional block ciphers may be less efficient [36]. Additionally, because the payload of Branca consists of an arbitrary sequence of bytes, we can employ a binary format such as Protocol Buffers. Protocol Buffers is a language-agnostic data serialization format that enables efficient and structured data interchange. Its primary advantage lies in defining data structures through a schema, which facilitates efficient encoding and decoding while minimizing storage costs compared to schema-less formats like JSON or XML [37]. This efficiency is particularly beneficial in high-performance applications, such as those in cloud computing and microservices architectures [38].

III. METHODS

The benchmark will be conducted on a standardized device configuration and specification which is MacBook Air M2 with 16 Gb memory and running MacOS Sequoia 15.3.1 to ensure fair and consistent comparisons of execution times and per-operation memory allocation. The whole process described in more detail in Figure 5.

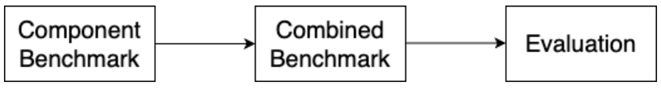

Figure 5. Evaluation process

With the assumption of a single user, each component benchmark will measure performance (ns/op) and memory allocation (B/op). It will be run ten times, and the average will be obtained. Subsequently, we will perform a combined benchmark that evaluates the speed of the hybrid stateless approach compared to approaches used in previous studies. If the combined approach proves to be faster, we will then assess its security to ensure that it remains both efficient and secure.

We will start with the payload component, which consists of marshalling and unmarshalling, to understand the bottleneck created inside the token and to find a suitable token specification that might support binary payload if the binary format turns out to be faster than plain text format. Next, we will benchmark the database, which will be used to compare stateless and stateful approaches and to save a refresh token for our hybrid stateless approach. Finally, we will benchmark the token spec's generation and parsing processes to have the smallest size, the fastest operation, and the least amount of memory usage.

Every payload and benchmark data set has the same data format and structure. Age (integer), Active (boolean), and Name (string) make up the structure. In contrast to Branca, JWT stores the token expiration information in the body itself, so for JWT payload data types, we include a "issued at" field in the body.

IV. Result and Discussions

The following tables summarize the detailed benchmark results for various components of the hybrid stateless approach, demonstrating that it is substantially faster and more secure.

TABLE I
MARSHALLING PERFORMANCE

Method	Sec/op	B/op	Allocs/op
encoding/json	114.7n ± 0%	80 ± 0%	2 ± 0%
goccy/go-json	69.71n ± 1%	80 ± 0%	2 ± 0%
protobuf	64.08n ± 4%	16 ± 0%	1 ± 0%

The marshalling benchmarks in Table I clearly indicate that Protocol Buffers substantially outpace traditional and optimized JSON-based methods in terms of speed and memory allocation.

TABLE II
UNMARSHALLING PERFORMANCE

Method	Sec/op	B/op	Allocs/op
encoding/json	472.4n ± 5%	256 ± 0%	6 ± 0%
goccy/go-json	95.80n ± 6%	80 ± 0%	2 ± 0%
protobuf	89.50n ± 0%	69 ± 0%	2 ± 0%

Like the previous benchmark, Protocol Buffers demonstrate the highest speed and efficiency for unmarshalling, as can be seen in Table II. We can conclude that Protocol Buffers prove to be a compelling choice for the token payload if the token specification supports binary data.

TABLE III
DATABASE PERFORMANCE

Method	Sec/op	B/op	Allocs/op
Mongo Insert	137.9µ ± 4%	5.560Ki ± 0%	85 ± 0%
Mongo Retrieve	93.76m ± 0%	12.75Ki ± 1%	106 ± 1%
Mongo Delete	97.55m ± 0%	11.51Ki ± 2%	332.5 ± 10%
Redis Insert	104.8µ ± 4%	679.0 ± 0%	14 ± 0%
Redis Retrieve	107.9µ ± 3%	600.0 ± 0%	12 ± 0%
Redis Delete	108.2µ ± 4%	599.0 ± 0%	12 ± 0%

The database benchmarks in Table III reveal that Redis significantly outperforms MongoDB across insert, retrieve, and delete operations. These findings further demonstrate Redis's superior design and suitability for real-time authentication management.

TABLE IV
GENERATE TOKEN PERFORMANCE

Method	Sec/op	B/op	Allocs/op
JWT HS256	1.574µ ± 1%	2.033Ki ± 0%	31 ± 0%
Branca encoding/json	2.595µ ± 3%	1.627Ki ± 0%	24 ± 0%
Branca goccy/go-json	2.481µ ± 3%	1.584Ki ± 0%	20 ± 0%
Branca Protobuf	1.531µ ± 1%	680.0 ± 0%	11 ± 0%

The token generation benchmarks in Table IV show that while the conventional JWT implementation exhibits competitive performance, when Branca is combined with Protocol Buffers, it matches or even exceeds JWT performance while simultaneously reducing memory allocations.

TABLE V
PARSE TOKEN PERFORMANCE

Method	Sec/op	B/op	Allocs/op
JWT HS256	2.558µ ± 1%	2.438Ki ± 0%	45 ± 0%
Branca TTL only	1.118µ ± 5%	488.0 ± 0%	9 ± 0%
Branca encoding/json	1.869µ ± 3%	1.094Ki ± 0%	20 ± 0%
Branca goccy/go-json	1.655µ ± 1%	1.584Ki ± 0%	19 ± 0%
Branca Protobuf	1.245µ ± 4%	1.031Ki ± 0%	11 ± 0%

The parsing performance benchmarks in Table V further validate the effectiveness of employing Branca in conjunction with Protocol Buffers. Notably, Branca permits TTL-only checks, whereas JWT requires full payload parsing because the TTL information is embedded within the payload itself.

To justify the use of stateless tokens, it is essential to ensure that the performance of token creation and validation processes surpasses that of session-only methods. In session-based verification, both database retrieval operations and payload unmarshalling are necessary. Even when excluding the unmarshalling cost, the Branca plus Protocol Buffers token is 86 times faster, primarily due to the significant overhead associated with database operations. This finding suggests that the blacklisting approach proposed by Fugkeaw et al. would be slower than both session-based and token-based authentication methods, as it requires token parsing in addition to database calls to check whether a token is blacklisted.

In contrast to the hybrid stateless approach, which requires access token creation, session-based token creation is inherently faster because it only requires the creation of random characters and saving the payload to the database (i.e., marshalling and a database insert operation). However, the difference is negligible. The extra time needed is just 1/84 of what is needed to inserting data into the database.

The cost of refreshing an access token is almost identical to that of session-based validation. The primary difference is the need to create a new token, which introduces an additional overhead of 1.531 µs. In real life, network latency will occur because the total number of requests increases to three: the user sends a standard request, receives a response indicating that the access token has expired, requests a new access token, and then sends the new token to complete the previous request. Additionally, according to our proposed solution, Jánoky et al.'s concern can be easily resolved by using the "issued-at" claim to specify different expiration times for each endpoint with varying threat levels. Although the Branca combined with Protocol Buffers implementation is only

0.043µs faster than JWT, this method enables each endpoint to have a unique expiration date, thereby reducing the need for token reissuance while maintaining a positive user experience and making implementation simple.

TABLE VI
REVOCATION STRATEGY

Method	Pro	Cons
Stateful Blacklisting (Fugkeaw et al.)	Immediate revocation	Has database call on each verification step (increase overhead)
Grouped Secret Change (Jánoky et al.)	Immediate revocation and less overhead on each verification step	User in same group forced to logged out
Dynamic expiration time (Ours)	Less overhead on each verification step	No immediate revocation and increased amount of token issuance

However, the token remains valid until its expiration time if the dynamic expiration is the only method of revoking it. This is one of the absolute cautions that must be considered and weighted against edge cases like token theft or account takeover, in which case the token must be invalidated right away. Table VI sum up the advantages and disadvantages of each revocation strategy.

TABLE VII
TOKEN SIZES

Method	Sizes (bytes)
JWT HS256	156
Branca JSON	113
Branca Protocol Buffers	76

Furthermore, as shown in Table VII, our results show that network overhead is not a significant concern when using stateless tokens because tokens generated using Branca together with Protocol Buffers are significantly smaller than JWT tokens. Additionally, research by Charyyev et al. further demonstrates that network overhead is no longer a concern, as 58% of end users can now reach a nearby edge server in less than 10 milliseconds [39].

V. CONCLUSION

Overall, our hybrid stateless approach eliminates the paradoxical situation where solutions must either compromise security by maintaining pure statelessness or forfeit the benefits of stateless authentication by introducing state management components. This can be attributed to the fact that our hybrid stateless approach's stateless verification component is 86 times faster than database calls and the dynamic expiration allows us to avoid blacklisting method proposed by Fugkeaw et al. which is significantly slower than stateless or stateful approaches because it combines both operations. Furthermore, changing the secret for smaller groups like the proposed solution from Jánoky et al. still have a detrimental effect on the user experience. In addition, we think that it is better to repeatedly consume the burden of token generation rather than consuming the burden we previously mentioned because our technique (Branca + Protocol Buffers) is just as quick as JWT HS256 (which simply signs, not encrypts), while using only 33% of JWT memory usage.

ACKNOWLEDGEMENTS

Thanks for research team from School of Information Technology Universitas Ciputra Surabaya.

REFERENCES

[1] L. C. Hamit, H. Md. Sarkan, N. F. Mohd Azmi, M. N. Mahrin, S. Chuprat, and Y. Yahya, “Adopting ISO/IEC 27005:2011-based Risk Treatment Plan to Prevent Patients Data Theft,” *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 3, pp. 914–919, Jun. 2020, doi: 10.18517/ijaseit.10.3.10172.

[2] M. Nicho, H. Fakhry, and C. Haiber, “An Integrated Security Governance Framework for Effective PCI DSS Implementation:,” *Int. J. Inf. Secur. Priv.*, vol. 5, no. 3, pp. 50–67, Jul. 2011, doi: 10.4018/jisp.2011070104.

[3] Y. Yang, “Security Evaluation of Financial and Insurance and Ruin Probability Analysis Integrating Deep Learning Models,” *Comput. Intell. Neurosci.*, vol. 2022, pp. 1–10, Jun. 2022, doi: 10.1155/2022/1857100.

[4] D. Hühnlein, T. Wich, J. Schmölz, and H.-M. Haase, “The evolution of identity management using the example of web-based applications,” *It - Inf. Technol.*, vol. 56, no. 3, pp. 134–140, Jun. 2014, doi: 10.1515/itit-2013-1036.

[5] A. Petcu, B. Pahontu, M. Frunzete, and D. A. Stoichescu, “A Secure and Decentralized Authentication Mechanism Based on Web 3.0 and Ethereum Blockchain Technology,” *Appl. Sci.*, vol. 13, no. 4, p. 2231, Feb. 2023, doi: 10.3390/app13042231.

[6] C. A. Ardagna, E. Damiani, S. De Capitani Di Vimercati, F. Frati, and P. Samarati, “CAS++: An Open Source Single Sign-On Solution for Secure e-Services,” in *Security and Privacy in Dynamic Environments*, vol. 201, S. Fischer-Hübner, K. Rannenberg, L. Yngström, and S. Lindskog, Eds., in IFIP International Federation for Information Processing, vol. 201. , Boston, MA: Springer US, 2006, pp. 208–220. doi: 10.1007/0-387-33406-8_18.

[7] T. Bazaz and A. Khalique, “A Review on Single Sign on Enabling Technologies and Protocols,” *Int. J. Comput. Appl.*, vol. 151, no. 11, pp. 18–25, Oct. 2016, doi: 10.5120/ijca2016911938.

[8] A. R. Pratama, F. M. Firmansyah, and F. Rahma, “Security awareness of single sign-on account in the academic community: the roles of demographics, privacy concerns, and Big-Five personality,” *PeerJ Comput. Sci.*, vol. 8, p. e918, Mar. 2022, doi: 10.7717/peerj-cs.918.

[9] Associate Professor of Information Technology, Faculty of Computers & Informatics, Zagazig University, Egypt, .. E., Professor of Information Technology, Faculty of Computers & Informatics, Zagazig University, Egypt, .. W., Department of Information Technology, Faculty of Computers & Informatics, Zagazig University, Egypt, and N. Salah, “Managing a Secure Refresh Token Implementation with JSON Web Token in REST API,” *Int. J. Wirel. Ad Hoc Commun.*, pp. 01–20, 2021, doi: 10.54216/IJWAC.020101.

[10] I. P. A. Pratama, L. Linawati, and N. P. Sastra, “Token-based Single Sign-on with JWT as Information System Dashboard for Government,” *TELKOMNIKA Telecommun. Comput. Electron. Control*, vol. 16, no. 4, p. 1745, Aug. 2018, doi: 10.12928/telkomnika.v16i4.8388.

[11] Faculty of Computers ; Informatics, Zagazig University, Department of Information Technology and A. Admin, “Managing a Secure JSON Web Token Implementation By Handling Cryptographic Key Management for JWT Signature in REST API: : A survey,” *J.*

Cybersecurity Inf. Manag., p. PP. 5-17, 2021, doi: 10.54216/JCIM.060101.

[12] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," RFC Editor, RFC7515, May 2015. doi: 10.17487/RFC7515.

[13] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC Editor, RFC7519, May 2015. doi: 10.17487/RFC7519.

[14] M. Jones and J. Hildebrand, "JSON Web Encryption (JWE)," RFC Editor, RFC7516, May 2015. doi: 10.17487/RFC7516.

[15] S. Dalimunthe, J. Reza, and A. Marzuki, "Model for Storing Tokens in Local Storage (Cookies) Using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in E-Learning Systems," *J. Appl. Eng. Technol. Sci. JAETS*, vol. 3, no. 2, pp. 149–155, Jun. 2022, doi: 10.37385/jaets.v3i2.662.

[16] S. Dalimunthe, E. Hasri Putra, and M. A. Fadhly Ridha, "Restful API Security Using JSON Web Token (JWT) With HMAC-Sha512 Algorithm in Session Management," *IT J. Res. Dev.*, vol. 8, no. 1, pp. 81–94, Dec. 2023, doi: 10.25299/itjrd.2023.12029.

[17] E. Al. Manish Rana, "Enhancing Data Security: A Comprehensive Study on the Efficacy of JSON Web Token (JWT) and HMAC SHA-256 Algorithm for Web Application Security," *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 11, no. 9, pp. 4409–4416, Nov. 2023, doi: 10.17762/ijritcc.v11i9.9930.

[18] L. V. Jánoky, J. Levendovszky, and P. Ekler, "An analysis on the revoking mechanisms for JSON Web Tokens," *Int. J. Distrib. Sens. Netw.*, vol. 14, no. 9, p. 155014771880153, Sep. 2018, doi: 10.1177/1550147718801535.

[19] L. V. Jánoky, P. Ekler, and J. Levendovszky, "Evaluating the Performance of Novel JWT Revocation Strategy," *Acta Cybern.*, vol. 25, no. 2, pp. 307–318, Aug. 2021, doi: 10.14232/actacyb.289455.

[20] S. Fugkeaw, S. Rattagool, P. Jiangthiranan, and P. Pholwiset, "FPRESSO: Fast and Privacy-Preserving SSO Authentication With Dynamic Load Balancing for Multi-Cloud-Based Web Applications," *IEEE Access*, vol. 12, pp. 157888–157900, 2024, doi: 10.1109/ACCESS.2024.3485996.

[21] A. Hauser, "JWT Issues Using JWTs as Session Tokens," JWT Issues Using JWTs as Session Tokens. Accessed: Nov. 11, 2024. [Online]. Available: https://www.scip.ch/en/?labs.20211014

[22] PortSwigger, "Jwt Attack," Jwt Attack. Accessed: Nov. 11, 2024. [Online]. Available: https://portswigger.net/web-security/jwt

[23] E. Pot, "JWT should not be your default for sessions," JWT should not be your default for sessions. Accessed: Nov. 11, 2024. [Online]. Available: https://evertpot.com/jwt-is-a-bad-default/

[24] S. Slootweg, "Stop using JWT for sessions," Stop using JWT for sessions. Accessed: Nov. 11, 2024. [Online]. Available: http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/

[25] S. Slootweg, "Stop using JWT for sessions, part 2: Why your solution doesn't work," Stop using JWT for sessions, part 2: Why your solution doesn't work. Accessed: Nov. 11, 2024. [Online]. Available: http://cryto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why-your-solution-doesnt-work/

[26] P. Costanza, C. Herzeel, and W. Verachtert, "Comparing Ease of Programming in C++, Go, and Java for Implementing a Next-Generation Sequencing Tool," *Evol. Bioinforma.*, vol. 15, p. 1176934319869015, Jan. 2019, doi: 10.1177/1176934319869015.

[27] F. Dahunsi, J. Idogun, and A. Olawumi, "Commercial Cloud Services for a Robust Mobile Application Backend Data Storage," *Indones. J. Comput. Eng. Des. IJoCED*, vol. 3, no. 1, pp. 31–45, Apr. 2021, doi: 10.35806/ijoced.v3i1.139.

[28] S. Alraddadi and S. Almotairi, "Performance Evaluation for MongoDB and Redis: A Comparative Study," *J. Eng. Appl. Sci.*, vol. 14, no. 19, pp. 7218–7222, Oct. 2019, doi: 10.36478/jeasci.2019.7218.7222.

[29] G. K. Spal, "Performance Evaluation of Redis and MongoDB Databases for Handling Semi-structured Data," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 6, no. 6, pp. 1255–1260, Jun. 2018, doi: 10.22214/ijraset.2018.6184.

[30] Mika Tuupola, "Branca as an Alternative to JWT?," Branca as an Alternative to JWT? Accessed: Feb. 19, 2025. [Online]. Available: https://www.appelsiini.net/2017/branca-alternative-to-jwt/

[31] Daniel Albuschat, "ChaCha," ChaCha. Accessed: Feb. 18, 2025. [Online]. Available: https://www.cryptography-primer.info/algorithms/chacha/

[32] KDDI Research, Inc., "Security Analysis of ChaCha20-Poly1305 AEAD," Feb. 2017, Accessed: Feb. 18, 2025. [Online]. Available: https://www.cryptrec.go.jp/exreport/cryptrec-ex-2601-2016.pdf

[33] A. Auernhammer, *Chacha20Poly1305*. Accessed: Feb. 13, 2025. [Online]. Available: https://github.com/aead/chacha20poly1305

[34] J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger, "New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba," in *Fast Software Encryption*, vol. 5086, K. Nyberg, Ed., in Lecture Notes in Computer Science, vol. 5086. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 470–488. doi: 10.1007/978-3-540-71039-4_30.

[35] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC Editor, RFC8439, Jun. 2018. doi: 10.17487/RFC8439.

[36] D. A. N. Gookyi and K. Ryoo, "The Hardware Implementation of NIST Lightweight Cryptographic Candidate SpoC for loT Devices," *IJASC*, vol. 3, no. 1, pp. 11–20, Mar. 2021, doi: 10.22662/IJASC.2021.3.1.011.

[37] D. Eddelbuettel, M. Stokely, and J. Ooms, "**RProtoBuf** : Efficient Cross-Language Data Serialization in *R*," *J. Stat. Softw.*, vol. 71, no. 2, 2016, doi: 10.18637/jss.v071.i02.

[38] C. Manso, R. Vilalta, R. Casellas, R. Martinez, and R. Munoz, "Cloud-native SDN Controller Based on Micro-Services for Transport Networks," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, Ghent, Belgium: IEEE, Jun. 2020, pp. 365–367. doi: 10.1109/NetSoft48620.2020.9165377.

[39] B. Charyyev, E. Arslan, and M. H. Gunes, "Latency Comparison of Cloud Datacenters and Edge Servers," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Taipei, Taiwan: IEEE, Dec. 2020, pp. 1–6. doi: 10.1109/GLOBECOM42002.2020.9322406.