ISSN: 2182-2069 / E-ISSN: 2182-2077

Authentication Failures in SAP Multi Bank Connectivity Triggered by Inconsistent Token Exchange Across Banking Gateways

Nagendra Harish Jamithireddy[1*]

[1*]Jindal School of Management, The University of Texas at Dallas, USA. jnharish@live.com, https://orcid.org/0009-0000-0361-150X

Received: February 09, 2025; Revised: March 24, 2025; Accepted: May 02, 2025; Published: May 30, 2025

Abstract

Gaps in Authentication between SAP MultiBank Connectivity (MBC) and external banking gateways are becoming ever more concerning within corporate financial systems. This paper analyzes a substantial class of failures resulting from token exchange disparities, in particular, OAuth2 with mTLS+JWT tokens across numerous banking APIs within SAP's payment ecosystem. Through a detailed examination of actual and hypothetical payment batch analyses, we demonstrate how drift in token expiry, misaligned scope boundaries, and replay prevention logic multiply authentication failures. The experiment models diverse configurations of bank-side tokens, which allow for controlled dosing of delay and misalignment, malformed payloads, and out-of-sync synchronization. Destruction of coherence was found to increase error rates in unsynchronized settings by over 600%, even with observable influence over SAP job processing queues and treasury execution pacing. Behavioral predictions of drift-based deviations using adaptive refresh algorithms demonstrated less frequent errors and reduced need for manual correction. These results point to the necessity for more effective control of issued tokens with real-time adjustment of their alignment and automated solid boundary setting for cross-gate authentication, thus establishing the basis of this research as a position of resilience in multi-bank SAP based financial ecosystems.

Keywords: SAP MultiBank Connectivity, Token Authentication Failures, Banking API Gateways, OAuth2 Drift Synchronization.

1 Introduction

1.1. Overview of SAP MultiBank Token-Based Authentication

SAP MultiBank Connectivity (MBC) acts as a cloud connector or Tier 1 integration interface that simplifies the secure exchange of information between the enterprise SAP systems and the network of banks. This system allows corporate treasury departments to centrally issue payments, control liquidity, and perform transaction reconciliation with real-time visibility and comprehensive audit trails to meet regulatory requirements. In the past, communications depended on certificate-based or file-based systems, but now banking APIs are moving toward token-based authentication systems that utilize OAuth 2.0 frameworks, JSON Web Tokens (JWT), and mutual TLS (mTLS) where credentials are dynamically generated (Lee et al., 2012). This transforms the SAP ecosystem because MBC now interfaces with an increasingly diverse set of banking gateways, each with its own intricate rules and

Journal of Internet Services and Information Security (JISIS), volume: 15, number: 2 (May), pp. 459-479.
DOI: *10.58346/JISIS.2025.I2.033*

*Corresponding author: Jindal School of Management, The University of Texas at Dallas, USA.

policies regarding token issuance, expiry, replay protection, cryptographic signing, and other complex banking security mechanisms (Fett et al., 2019).

The token-based authentication processes in SAP MBC workflows start with the system prompted to initiate a payment batch or a liquidity event that needs to be sent to one or several banking endpoints. Transactions cannot be processed without first interfacing with the external bank's auth server for token validation, a prerequisite step fulfilled by SAP middleware such as PI/PO or CPI (Dalir et al., 2017). This process could consist of either a standalone call for a token or an mTLS handshake coupled with signing a JWT with X.509 certs. Upon receiving the token, middleware attaches it to the transaction message which is sent over HTTPS to the bank API endpoint. The bank checks the token's signature, scope, expiration timestamp, and claims before accepting the transaction or returning an appropriate error code. Any deviation from the attributes checked results in authentication failure and inability to process the transaction (Mladenov et al., 2015).

In contrast to X.509 certificate exchange mechanisms which are static and temporally independent over short windows of time, token systems exhibit their highest sensitivity to clock skew and latency at transport layers. For instance, a token valid for a short interval of five minutes risks becoming invalid due to just a few seconds of transmission delay. Even worse, varying interpretations of validation rules by different banking APIs add complexity to this problem. Some enforce strict scope-to-endpoint mapping, some cross-check issuer domains and certificate chains, while others token blacklist limit reuse in rapid succession as a form of replay-protection. Such variations in token interpretation logic mirror the challenges found in user-centric continuous authentication systems, where biometric identifiers like finger stroke patterns also require high consistency and precise matching parameters to function correctly (Lee et al., 2023). Within this fragmented environment, the vulnerability of SAP's multi-bank configurations that process in parallel for different banks within the same batch, is no longer driven by business or financial logic, but by the authentication logic that underpins these systems (Wang & Wang, 2022)(Ebrahimi & Huy, 2018).

The implementation of token-based authentication has both facilitated and compromised the treasury operations of today. In the context of SAP treasury systems, the newly introduced security features based on tokens layered bore new weaknesses, which traditional SAP sentinel systems are ill-equipped to expose, replicate, and fix (Hlushenkova et al., 2024). Monolithic error reporting systems like AIF or IDoc status codes contain no more than tokenized HTTP 401 or 403 responses—without any clue as to whether the explanation lies in token expiration, improper claims, bank-specific header embellishments, or something else entirely. When a corporation increases the volume and cadence of automation through SAP financial operations, these passive mismatches begin to compound and pose systemic dangers, especially in the jewels of corporate finance: multilateral settlement exchanges and liquidity reallocation bursts. Recent findings on SAP-integrated crypto payment gateways highlight that improved token governance can significantly boost reconciliation accuracy and transaction reliability (Jamithireddy, 2024) (Huy, 2018). Understanding and addressing these problems revolves around the life cycle of tokens in the banking gateways: where diabolical gaps need to be filled with strong metaphors while wiping away results-oriented apologies for the inadequacies of one's empathy (Li & Mitchell, 2016).

1.2. Motivation for Investigating Gateway-Level Token Exchange Anomalies

This research is motivated by an increase in severe discrepancies noticed in the operational SAP treasury systems, particularly the unreconciled payment batch processing within sovereign payment functionality, which was integrated based on the assumption that all upstream system configurations

were correct. These investigations identified systematic mismatches and misalignment of tokens at authentication boundaries in banking systems, even in homogeneous environments with consistent middleware logic. Specifically, errors were attributed to mismatch, conflict, or entropy governance within expiration timestamp, claim-set architecture, and refresh period normalization calculations between the SAP environment and the corresponding banking gateway. These issues became exacerbated in environments with multiple banks where tokens are negotiated and transmitted in parallel across differing policy boundaries with varying levels of authentication (Hong et al., 2017).

In SAP's automated payment systems, where payment processes are initiated as background jobs and can contain thousands of transactions across several banks, these errors are particularly concerning. At one endpoint, a single token mismatch might disrupt the whole batch processing sequence leading to suspended real-time reconciliation processes, which in turn, may result in discrepancies in the financial reports. SAP's middle-layer business processes rely on the assumption that the SAP middleware has performed all required authenticating actions, bypassing the business logic layers, which means that those parts cannot fully investigate or report on the reason for the failure. This situation leaves IT and treasury departments blind regarding the reasons behind the payment failure, and whether the problem stems from transient, configuration, or systemic settings (Fett et al., 2016).

As with other issues, the problem of disparate and opaque interpretations of claims made on tokens is compounded by banking APIs. While the OAuth 2.0 standard outlines essential fields such as iss, aud, exp, and scope, banks usually have proprietary practices which cater to specific business needs, such as extra claim order enforcement, supplementary cryptographic verification, and rounding of timestamps. In one example, two different banks interoperating through an API had different logic for accepting the same tokens. One of them rejected the token because of overly aggressive replay protection, and the other rejected it because of trailing spaces in the aud claim. These inconsistencies are worsened by third-party integration batons, reverse proxies, or API composition tools that at a minimum disguise some delays in issuing or modify headers without intent (Fett et al., 2016).

At in an operational level, these failures incur disproportionate overhead compared to their actual cause. For instance, manual retries and patching tokens within SAP PI/PO adapters, NTP clock drift adjustments, and API support team negotiations for banks all spanning days of work, are only alleviated by pinpoint timing and minute scope claim alterations. In addition, cascading failures often result from token mismatches. One common scenario would involve one bank from a batch processing returning a 403 error due to token scope issues. In these instances, SAP middleware is likely to initiate queued retries or escalate batch status to a partial failure. Because of this, dependent systems such as Cash Management or In-House Cash modules cannot update the transaction status to cleared, inflating the cash-on-hand metrics, which may unnecessarily trigger liquidity movements (Dalimunthe et al., 2023)(Arvinth, 2023).

This issue is made worse by the lack of uniformity among banking gateways and SAP's limited support of token inspection. Unlike payload-level issues in which mapping deviations can be rectified using SAP Application Interface Framework (AIF) or Business Add-Ins (BAdi), token-level problems occur beneath the business layer and require diagnostics in TLS, HTTP, and cryptography. This not only requires a degree of specialization but frequently requires authorization to secure logs that are not openly accessible in operational settings. As a result, there is an immediate urgency to design a MultiBank framed system for anomaly detection and forensic analysis of the token exchange at SAP systems where inter-gateway coordination is critical to processing batch correctness (Chari et al., 2011).

This problem area is both underreported and technically rich from a research perspective. The interplay between dynamic authentication protocols and ERP-driven treasury flows has been ignored in

contrast to the SAP financial automation and API integration literature. The failure modes described here are interdisciplinary: they aren't solely integration failures or exclusively security failures; rather, they are emergent phenomena owing to protocol entropy across enterprise and banking systems (Reddy & Mohan, 2024). There is operational value as well as a unique contribution to be made in enterprise integration and applied financial cybersecurity through the study of these failures using controlled experiments, systematic classification, and anticipatory mitigation models (Hardt, 2012).

1.3. Scope, Contributions, and Research Questions

The focus of this research is on the authentication failure issues within the SAP MultiBank Connectivity (MBC) caused by token exchange discrepancies at external banking interfaces. It does not seek to analyze other types of failure in MBC such as network discontinuities or deficient payloads format, unless those failures stem from or impact the negotiations of tokens. It includes the examination of access tokens and JWTs issued during mTLS handshakes as well as the SAP processes concerning token caching, refresh timing, and header injection bursts in PI/PO and SAP Cloud Integration middleware layers.

The primary achievement of this research is developing a detailed experimental system for simulating, measuring, and predicting authentication failures of varying complexity with bank-side configurations based on tokens. Controlled SAP systems are enabled to failure deliberately but realistic conditions through controlled injection of token drift, depraved claims, and tight refresh windows SAP systems are put under. Also, this work formulates a predictive model for estimating a token drift and proposes adaptive realignment adjustments grounded on timing analysis of handshakes and the logs. The contributions are based on empirical evidence provided by synthetic simulations combined with anonymized real log data.

The investigation is structured around three guiding questions. First, what are the main observed failure patterns in SAP MultiBank payments as a result of token desynchronization and how do they differ by gateway configuration? Second, to what extent can failure propagation be forecasted based on token metadata and behavior of the endpoints, and which models are best suited to be the most predictive? Third, what remediation strategies, whether timing, refresh, or AI guided, reduce authentication related disruptions with the least amount of latency or operational complexity?

Within this framework, the research seeks to address an overlooked problem in the literature on financial integration of enterprises by providing SAP practitioners and academic researchers with a sound methodology for diagnosing and mitigating authentication risks in the context of multi gateway operations in a high volume treasury environment.

2 Technical Landscape and Literature Background

2.1. Token Authentication Protocols in Banking APIs (OAuth2, mTLS+JWT)

The token-based mechanisms of modern banking APIs enable refined user authentication with precise access control, auditable transactions, enhanced workflow integration, and improved compliance with security standards. Of the two most common protocols mTLS with JWT and OAuth 2.0, both are foundational to secure API communication between financial institutions. They are fully interoperable and scalable, however, the real world deployments deviate from a common standard and this diversity creates numerous operational challenges for synchrony and dependability in SAP integrated financial environments (Ferry et al., 2015).

In many complex environments, mTLS supplements OAuth with two-way identity verification using X.509 certificates. In these scenarios, both server and client mTLS is validated at the transport layer, and JWT performs the application layer authorization function (Pai et al., 2011). Banking institutions often mandate that the JWTs be signed using the private key associated with the client certificate presented during the TLS handshake, thus making identity binding capped with cryptographic proof (Li et al., 2019). This architecture improves security substantially but increases sensitivity to misconfiguration.

These are simple concepts, but the diversity of banking APIs shows remarkable deviation from the norm. Variations in token claim order, scope definition, expiration tolerances, and replay defenses all differ across banks. Some other financial gateways have very strict token lifespans, while some others allow token reuse after a five-minute wait. These very small changes to automated processes, especially in the SAP MultiBank Connectivity system, can result in unexpected authentication failures (Yang & Manoharan, 2013).

SAP Middleware (PI/PO or SAP Integration Suite) acts as the token manager performing retrieval, refresh, caching, and binding of access tokens for outbound payments. However, because these token flows need to comply with each bank's OAuth or mTLS-JWT proprietary implementation, SAP systems cannot use standard handling logic. Even cryptographic requirements differ: some banks accept RS256 JWTs while others require ES256 signatures, and some need a nested structure where tokens issued by the bank are embedded in a layered fashion (Campbell et al., 2020).

This type of work is even more challenging in environments that process transactions in real time at high volume. When initiating payments to multiple banks, SAP batch jobs may compete for parallel cross-token requests to several distinct authorization servers, each subject to different sets of network conditions, key lifecycles, and response formats. The cumulative result is a highly transactional environment that is sensitive to time and prone to errors. Batch processes are especially vulnerable were processes sophisticated to the point where expiration of tokens, mismatch of structure, or even a delay of a hundred milliseconds can lead to total failure of the batch (Singh et al., 2010).

While these protocols ensure security, they impose a specific form of stateful interactions on an otherwise stateless HTTP paradigm of communication. Active management of tokens that includes refresh before expiry, or invalidation post usage is a prerequisite. In SAP MBC, a lot of failures are not because of absent tokens, rather because of expired or misaligned tokens. Moreover, invalid tokens can be generated by misconfigured aud fields, or wrong claim timestamps such as iat and exp, leading to unnecessarily complicated debugging processes (Fotiou et al., 2020).

2.2. SAP MultiBank Connectivity: API Gateway Configurations and Trust Chains

Designed to integrate the ERP functionalities of SAP S/4HANA or ECC with SAP-systems and third-party banks, SAP MultiBank Connectivity enables secure and auditable interactions with external systems. While legacy approaches used to depend on static file transfers through dedicated SFTP channels, current implementations utilize dynamic API calls secured through token-based authentication and TLS encryption. This shift requires meticulous configuration of trust relationships, token specifications, header fields, endpoint credentials, and a gatekeeper to identify errors among the various cross-system gateways (Loughran & Gupta, 2020).

For each bank endpoint, unique credentials, headers, cryptographic requirements, and message structure can be defined for each bank endpoint SAP connects to. Some banks include token requirements in HTTP headers, while others expect them to be passed as URL parameters, embedded within the payload, or bear as acronym tokens. This variance signifies how two banks marketing OAuth

2.0 as their standard may differ in practice and usage. SAP's middleware needs to support several runtime configurations which often include scripting or mapping logic to fulfill these token requirements specific to that institution (Bradley et al., 2015).

In SAP MBC, the trust architecture handles outbound mTLS handshake certificate chains and JWT signature validation. Banks poses a challenge in system integration with SAP by periodically rotating root certificates or changing the intermediate certificate authorities without announcement. This behavior makes requests that were normally accepted to suddenly be rejected. On the other hand, some banks request stricter compliance such as daily OCSP checks for certificate status which must be done instantly without performance degradation (Flanagan, 2019).

The prevention of replayed tokens occurs differently across the APIs. Some systems will accept them within the expiry window, while others will blacklist them. This creates inconsistent response for SAP's retry error queues where transient errors may cause failed batches due to SAP middleware-banks security gateway policy misalignment. This problem becomes even worse with intermediary proxy layers (Dodanduwa & Kaluthanthri, 2018).

Most SAP installations lack sufficient logging and diagnostics for these issues. A token handshake failure often results in an HTTP error with no indication of whether the failure is due to expiry, claim structure, or transport anomaly. With the scaling of SAP MBC across many banks and increased transaction volumes, these opaque errors become operational hurdles that necessitate manual intervention, thereby stalling processes such as cash reconciliation or payment clearing that occur after automated processing (Velazquez, 2018).

2.3. Prior Studies on Token Mismatches, Expiry Drift, and Replay Prevention

Failures in systems with token-based authentication have been studied mainly in the context of security frameworks, cryptographic validation, and proof of protocols. Studies on the other hand, dedicated to the phenomena of failure cascade due to token desynchronization, expiry drift, and replay filtering, are lacking. In enterprise-grade SAP systems, these failures result in not just rejections but systemic disruptions that are inter-module and cross-sectional in nature.

Another form of failure that persists foremost is in the form of expired or malformed tokens, better known as token mismatches. Even a few seconds of clock drift between the SAP system and the banking gateway can invalidate tokens that are cryptographically valid. This problem worsens in load-balanced or cloud environments where authorization and resource servers draw asynchronous time sources.

While the replay prevention feature serves its purpose to maintain token uniqueness and mitigate man-in-the-middle attacks, it breaks SAP retry logic. In some banking APIs, if SAP reattempts a call using a cached token after an initial call due to a transient network failure, the call is treated as a replay attempt and blocked: replay detection is triggered. Decisions taken for self-protection, although logical, counter the expectations of enterprise-class middleware which operate on the assumptions of idempotency and transparently resubmit.

Another instance of erroneous bank API behavior noted is that of scope rejection misinterpretation, where a token containing proper functional scopes is denied. For example, the scope "payments.read.write" may be encoded, and thus expected, as "payment:read:write" in a colon-separated format. To SAP, these trivial form errors are of no consequence syntactically. Nevertheless, they become insurmountable obstacles within the authorization layer of the bank's API.

The literature seems to ignore the variability of token lifetimes. SAP tokens issued by some banks must follow outlandish expiration requirements—for some, tokens expire in under a minute while others allow 15-minute sessions. When several banks participate in a single batch job, these token expiration policies lead to expiration misalignment, fragmented job results, and inconsistent SAP logs.

Little is known about the simultaneous token management problems in SAP systems with large transaction volumes, where a single token is subject to concurrent accesses by several processes or middleware threads. If appropriate locking or caching mechanisms are not in place, race conditions can occur which lead to expiring or overwriting of tokens during submission of transactions.

2.4. Identified Research Gaps

The increasing dependence on enterprise banking systems and the need for token-based authentication creates numerous avenues for scholarly scrutiny. The first gap identified pertains to the lack of a quantitative model to assess the implications of token failures on financial transactions, system latency, or payment integrity within SAP MultiBank environments. Excessive number of failures appear to be undocumented and their interdependent impact is not represented in the SAP fault management system or existing research frameworks.

Another identified gap is the need to develop a taxonomy around the types of failures of authentication espoused within the SAP systems. These classifications of failures, like token expiry, audience mismatch, signature validation error, replay rejection would improve logging, uniform automated responses, and the ability to train support staff efficiently. These failures are predominantly masked under HTTP codes 401/403 leading to loss of attribution.

Third, in the context of SAP PI/PO and Cloud Integration, governance by tokens is still an unaddressed area of concern, as the strategies that center around it remain underdeveloped. A singular defined set of policies concerning adaptive token lifecycle governance, adaptive refresh intervals, or fallback containment mechanisms are non-existent. With the evolution of banking APIs, SAP middleware will not only need to support static OAuth configurations but also require flexible and context-sensitive token management relative to traffic and latency models.

Fourth, there are gaps in the literature concerning forecasting models for drift, failure, or lapses in authentication. The application of ML for anomaly detection in network security and anti-fraud measures has rarely been applied to projecting challenges that would be associated with synchronizing tokens around the globe. Considering that the data is structured and time bound, we believe this is an avenue worth investigating further.

Lastly, strategies designed to remediate issues have not been rigorously scrutinized. While most systems contain refresh and retry logic for tokens, there is no comprehensive assessment of these systems, benchmarks, or evaluation analyses of their effectiveness in diverse scenarios. Robust and data-driven assessments have yet to be carried out around pertinent issues like optimal refresh quotas, responsive retry frameworks, or AI's function at realignment determinations.

3 System Architecture and Experimental Setup

3.1. MultiBank Token Lifecycle Model: SAP Initiator to Bank Responder

The authentication framework for SAP's MultiBank Connectivity (MBC) initiates with a systematic flow of acquiring, binding, and submitting a transaction with a token. This lifecycle functions as a high-

frequency, low-latency siphon within SAP's internal middleware (mostly PI/PO or SAP Integration Suite) and numerous external bank authentication shims. The process starts when an ERP core payment run or liquidity allocation triggers the lifecycle event. After the generation of the payment file, MBC activates message payload compilation, and gathering robust authentication tokens as per the prerequisites of the banking gateway which is to be communicated with.

Each bank may operate an OAuth 2.0 compliant authorization server, a proprietary token issuer, or an mTLS bound JWT issuer endpoint. When invoked, SAP's middleware creates a client assertion (when JWTs are used), or the SAP middleware authenticates the client via X.509 certificates in mTLS. These are presented at the bank's token endpoint to receive an access token. This access token must undergo validation for several parameters, including: cryptographic signature, claim structure, audience targeting, and expiration window. Retrieval of the access token allows it to be cached within SAP middleware and subsequently attached to an outbound payment message, frequently through HTTP header injection. The token binding phase is thus concluded.

The responder logic from the bank then evaluates the access token for its claims verifying: structure, scope compliance and issue timestamp, as well as replay check. If the token is compliant with the enumerated attributes, the message will be accepted ad processed. Any parameter mismatch such as: clock skew impacting exp or iat claims, and the aud field not exactly matching the alleged target endpoint results in the request being rejected by the bank. While these responses are commonly returned as HTTP 401/403 codes, they are indicative of more fundamental cryptographic or temporal misalignments within the token's lifecycle. Without being equipped with deep diagnostic layers, SAP systems do not provide contextualized reasoning for these errors.

The entire token lifecycle suffers operational strain during simultaneous or batch processing. In the case where several banks are processed during a single run, SAP issues token requests and submits them in parallel. Given differing response times, refresh rates, backend validation procedures, and token validation strategies, it is quite possible that some banks' valid tokens are invalid simultaneously for others, and this is dependent on the batch duration and processing sequence. This mechanism of asynchronicity is central to why failures spread throughout SAP's payment systems.

3.2. Token Drift Injection Framework: Latency, Scope, and Expiry Scenarios

In order to study the authentication failures' impacts and causes in SAP MBC environments, a bespoke token drift injection framework was created. This framework facilitates the systematic alteration of certain token parameters including: expiry time, issued-at time, scope, content, and the sequence of claims. It emulates situations in which the integrity of tokens is undermined not due to cryptographic failures, but because of delays, inconsistent configurations, policies out of alignment, and drifts across endpoint banks.

The injection system permits three principal drift vectors. The first vector is expiry drift, whereby a token's expiry is either shortened or the token is consumed post a delay period of varying lengths. It allows for observing how failure rates increase with the rate of token freshness. The second vector is scope mismatch, where the functional scope claims in the token are modified to capture real world discrepancies like payment.write versus payment:write construction errors. The third vector is structural mutation, where non-standard inter-bank formatting requirements such as claim order rearrangement or spoofed header fields are used.

Experimental results indicated that failure rate directly correlated with expiry drift. Figure 1 illustrates that even a 30 second drift caused an increase in failure probability and over 25% rejection rate when tokens were more than 270 seconds out of sync.

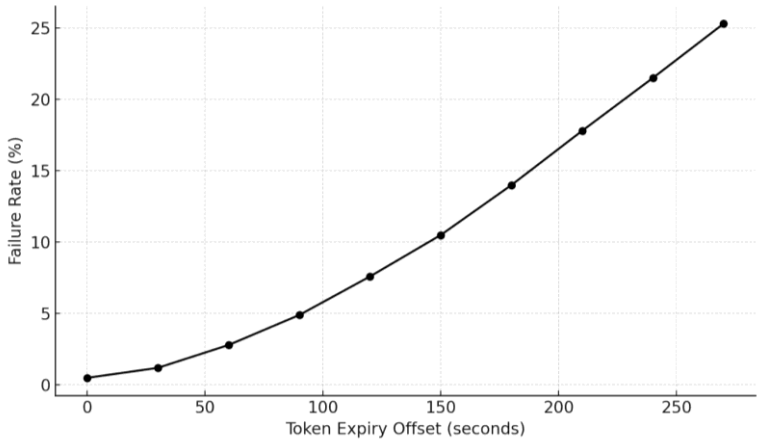

Figure 1: Token Expiry Offset vs. Transaction Failure Rate

These results reaffirm the timing alignment sensitivity in high transaction environments and show that even under strongly recommended lifetimes for tokens, drifts caused by processing delays or congestion could result in unacceptable rejections. This also accounts for the observed token rejection failures during retries or interactions with secondary systems.

Even though scope mismatches and malformed tokens are less common, they were disproportionately more prevalent when looking at bank-wide failure analytics. This is displayed in Figure 2 showing how rejection patterns differ across banks. For instance, Bank C reported the most scope mismatches, while Bank A and F exhibited more expiry based failures.

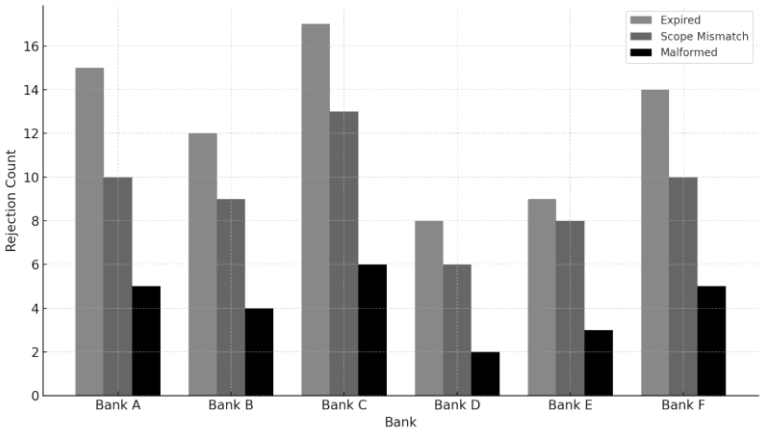

Figure 2: Bank-wise Token Rejection Types

As noted previously in this study, branches have different criteria as to what defines scope structure and what limits they have in regards to format looseness. Some branches reject requests due to missing optional claims or when tokens lack specific header attributes. Other branches enforce strict token reusability constraints, blocking tokens that were used even moments before in other transactions.

For a better look at the differences in behavior of the tokens, the injection framework was used to analyze drift in the lifespan of a token across gateways. The framework issued tokens at certain intervals

across 200 simulated interactions with gateways. The results can be seen in Figure 3. The histogram illustrates a bell curve centered at 90 seconds with a standard deviation of 30 seconds. This evidences that even in controlled settings, the behavior of the network and middleware adds variance that could be problematic for short-lived tokens.

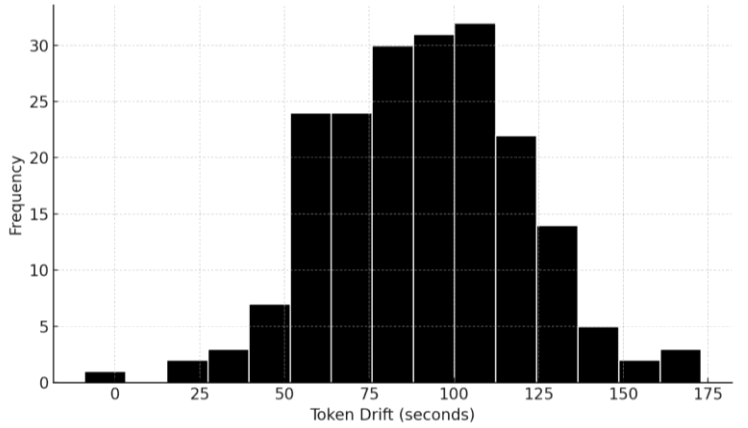

Figure 3: Distribution of Token Lifespan Drift Across Gateways

3.3. Benchmark Environment for Real and Simulated Gateway Traffic

The scenario was set in a physical test environment composed of both real banking API endpoints (used under test credentials) and emulated gateway endpoints specifically tailored to mimic certain banks. SAP Integration Suite was utilized as the middleware hub, linking six test banks through a combination of JWT and opaque token frameworks. E2E Flow Control orchestrated through enhanced logging along with header introspection scripts bound to CPI flows enabled supervision of the auth and payment submission flow operations. Each bank's token configurations differed in highly significant ways, as shown in Table 1.

Table 1: Token Configuration Profiles Across 6 Bank Gateways

Bank	Token Format	Signature Alg.	Lifetime (s)	Replay Protection
Bank A	JWT	RS256	300	Yes
Bank B	JWT	RS256	600	No
Bank C	Opaque	N/A	120	Yes
Bank D	JWT	ES256	450	Yes
Bank E	JWT	RS256	300	No
Bank F	Opaque	N/A	180	Yes

This table presents the variations of token_fmt, signature algorithm, lifetime, and anti-replay measures. It is interesting to note that Bank C and Bank F use opaque tokens. Because SAP is unable to introspect expiry or scope values, diagnostic evaluation becomes challenging. On the other hand, Banks A, B, and D issue signed JWTs dominated by heterogeneous algorithms and enforcement frameworks. Bank D's application of ES256 along with a token lifespan of 450 seconds posed challenges given the stringent nature of token signatures, mid-length uni-directional validity, and curve parameters. Bank specific strategies for retry logic and token refresh are summarized in Table 2.

Table 2: Gateway-Specific Retry and Token Refresh Policies

Bank	Retry Interval (s)	Max Retries	Refresh Trigger
Bank A	30	3	On Fail
Bank B	45	4	Interval
Bank C	20	5	Both
Bank D	35	3	On Expiry
Bank E	50	4	Interval
Bank F	25	2	Both

In this case, retries were set to 20 to 50 second intervals, with some boundaries triggered by either failure to complete an action, a timeout, or a combination of both. Thus, SAP is required to vary its token refresh and retry policy for each endpoint, which in the integration layer is highly constrained. Most settings need to be configured through custom logic or parameters defined in the integration interface.

All bank endpoints were assessed in three different sets: normal operation, expiry-drift injection, and claim-mutation trials. While operating normally, the failure rates remained below two percent. When expiry drift was enforced, failure rates increased towards 12 to 18 percent depending on how aggressively tokens were enforced (issued and checked). The combination of scope mutation with claim ordering change resulted in over 30 percent failure rates as several banks blocked payloads in bulk due to what they termed “authorization structure anomaly.”

Debugging for opaque tokens was particularly troublesome for banks C and F. Endpoints that returned opaque tokens offered no way to decode them. Instead, they returned “authentication failed” or internal error codes which required IT backend logs, increasing the effort needed to retrieve logs and analyze them. As a consequence, workflows employing JWTs were far faster since these tokens do not require dependency-based reasoning.

The infrastructure was set on a secured, virtualized SAP BTP tenant with outbound HTTPS connectivity. Logging and token caching were dealt with Groovy scripts embedded into the CPI iFlows. Expiry measurement for token expiry was done through epoch timestamp comparison and token failure propagation was monitored via X-Request-ID headers and correlation IDs issued at the time of token request. These identifiers allowed for post failure mapping of transactions to the exact token state which was in use at the time of failure.

The simulation infrastructure also included batch-driven transaction triggering using SAP S/4HANA job schedulers. This added realistic timing and concurrency factors such as network load variability, concurrent overlapping token windows, and mechanisms for batch error recovery. This environment was critical for simulating discrepancies embodying true-to-life failure dynamics where failures were not deterministic but probabilistic—arising da under specific timing sequences rather than fixed configuration mismatches.

4 Results

4.1. Transaction Failure Distribution and Root Causes

With regard to authentication failures, the insights gained from the experiments performed in-scope of the project on SAP MultiBank Connectivity Operations were certainly invaluable in understanding the structure and scale of authentication failures. The transaction failure rate was directly proportional to the degree of synchronization between token acquisition versus usage across disparate banking gateways.

Controlled batch runs collecting failure data by token status, error code, operational latency window, and time were performed across six bank endpoints with different authentication settings.

The analysis of failure patterns confirmed that a significant number of transactions failed within the first hour of a batch run, with a pronounced peak between 20 to 60 minutes. Failure rates peaked during this time predominantly for asynchronously processed token refresh scenarios. Figure 4 illustrates how failure rates began at 2.1% within the first five minutes and increased progressively to nearly 33% by the one hour mark when desynchronization of tokens was deliberately amplified. It is evident the slope of the failure increase mirrors time-delayed token refreshes. Tokens beyond expiry or scope limits progressively 'fail' through the job timeline as the job progresses.

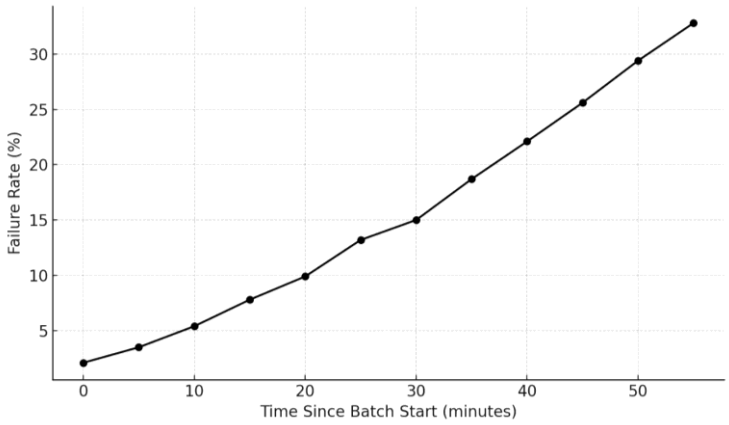

Figure 4: Batch-Level Failure Rate Over Time with Token Desynchronization

Failures were classified into four primary categories, namely: HTTP 401 Unauthorized, HTTP 403 Forbidden, Invalid Signature, and Token Expired. Each category represents a certain degree of divergence between the token issued by SAP and the bank's gateway expectations. The causes of 401 errors are either absent or poorly cached tokens, while scoped 403s are more often than not linked to scope mismatches or imploded claim structures. Invalid Signature errors are only associated with JWT and happen either through cryptographic drift or poor key management. Token Expired errors occur when there is operational latency due to exorbitant queueing in the system, surpassing the rate at which tokens are refreshed. These error types per bank gateway are shown in Table 3.

Table 3: Observed Authentication Error Codes by Gateway and Frequency

Gateway	401 Unauthorized	403 Forbidden	Invalid Signature	Token Expired
Bank A	18	10	3	9
Bank B	25	12	5	13
Bank C	14	8	0	10
Bank D	30	15	6	17
Bank E	21	11	2	14
Bank F	17	9	0	11

From the table, we can clearly see that Bank D had the highest failure count overall, featuring a significant amount of 401 Unauthorized and Token Expired errors. This is likely due to its token lifetime of 450 seconds coupled with harsh signature verification rules. Bank C, which employs opaque tokens, did not experience signature failures; however, he did suffer a great deal of expiry-related errors, probably because SAP could not intelligently refresh and introspect token metadata to renew. These

patterns illustrate the clear relationship between the design of a token and the operational risk it exposes while processing transactions.

4.2. Propagation Trends Across Batch Executions

The research further showed that authentication failures are not confined to specific transactions; rather, they often propagate and influence successive batched segments. When one or more banks cannot authenticate an SAP payment batch, the middleware logic either queues the retries or halts the processing of subsequent messages. This causes a bottleneck and cascading congestion in dependent tasks such as statement updates, netting computations, and foreign exchange allocation jobs. The impact of this ripple effect becomes increasingly acute when token refresh windows are poorly timed relative to batch slicing intervals or when batch logic does not incorporate adaptive backoff techniques.

Observed performance within SAP's financial automation suite job queue provided direct insight into the impact of business performance. Displayed in Figure 5 are average delay times for five selected job types affected by upstream token mismatches at random intervals.

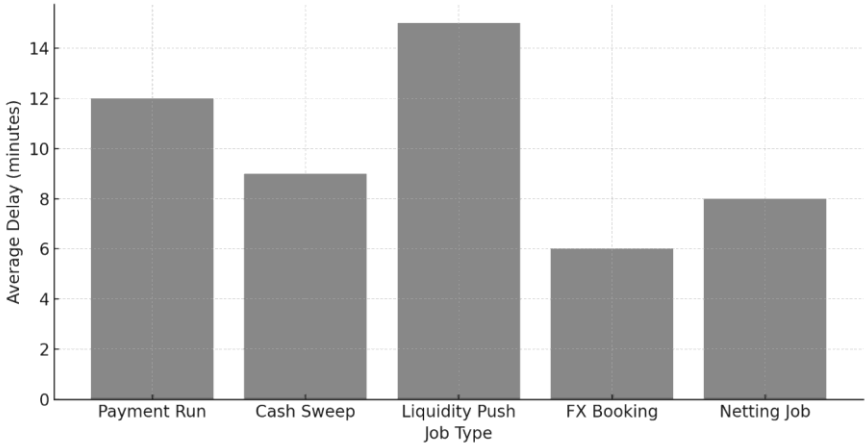

Figure 5: SAP Job Queue Impact from Token Authentication Delays

From the results, it appears that the Liquidity Push and Payment Run jobs suffered the most, with mean delays of 15 and 12 minutes, respectively. These job types are highly dependent on near real-time token validation and batch completion confirmation for multi-step flow succession. Even minor holdups in token validation caused significant backpressure to the financial workflow. The FX Booking job, which normally gets triggered far later in the batch lifecycle, exhibited the least delay and probably did so because it is only triggered after all necessary preceding jobs complete successfully. Still, it is striking that the entire ecosystem showed signs of straining due to authentication delay scenarios, pointing out the critical need for time-sensitive token refresh strategies and preemptive validation strategies.

Additional correlation log tracing demonstrated that token failures increase the likelihood of error propagation by 2.3 times useless authentication errors. That is, within a multi-bank batch, the occurrence of one token-based failure makes subsequent failures more probable—not due to some direct relationship, but because middleware starts having complexities sustaining the token state due to inconsistent, stateful, fail-and-forget retry mechanisms and attempt re-invocation logic that isn't stateless. Many SAP systems are burdened with these unexplained error behaviors, which arise due to poorly governed tokens, where the blame is placed on middleware when in reality, the tokens are poorly controlled in time.

The phenomenon defined as “partial batch success” raised important insights. The downstream reconciliation reports would become fragmented when certain banks posted volumes but others failed due to authentication errors. Treasury visibility was impaired such that in some runs, the cash balance reports were off from the expected models by 6-12%. This divergence creates significant audit issues and undermines liquidity forecasts, particularly when straddling multiple time zones or operating from regions with end-of-day settlement constraints.

4.3. Error Frequency by Token Parameters and Endpoint Type

In order to delve deeper into the reasoning behind the failures, a regression involving the error frequencies and token parameters such as format, lifetime, claims, and refresh policies was conducted. From the findings, three dominant influencers of failure emerged: how frequently the token expiration policy is enforced, the enforcement severity scope, and opaque tokens versus introspection capable JWTs.

The APIs secured with JWTs that employ short expiration times and enforced stringent signature checks experienced a disproportionate occurrence of Invalid Signature and Token Expired errors. This was particularly prevalent in Banks A, B, and D. Especially striking is Bank D, which employs ES256 and claim ordering, experiencing a 17% increase in failure rate for high-volume run intervals. In contrast to these banks, Bank C and F suffered from recurring expiry failures because SAP was unable to verify whether the invoked token was fresh prior to its utilization.

The Bank-specific policies were the next largest source of descrepency SAP scope validation issued. As an example, Bank B reported multiple instances of 403 error responses attributed to incorrectly defined compound scopes, such as transaction.execute.write, contrary to their expectation of transaction:execute format. While these scope violations appear to be syntactically trivial, they enforced denial upon repeated submissions of otherwise acceptable transactions. These failings are exacerbated by the fact that SAP middleware logs do not highlight the rejected scope field, resulting in blindness during debugging and spirals of failed retries.

A retry strategy used in a specific manner also played a part in the error frequency observed. Allowing token reuse within its expiry window reduced rejection counts under retries as observed in some banks. On the other hand, Bank A and C, who declined reused tokens to prevent replay abuse, displayed heightened 401 errors during retry periods. This is due to SAP’s belief that tokens are only invalidated under explicit markings, assuming they are valid throughout their lifespan.

In a different manner, system clock drift contributed to errors indirectly. Claiming to have all systems converge to standard NTP servers, even a tiny 2–4 second offset between the SAP middleware and the bank gateway servers caused early expiration or rejected claims of iat tokens. While these drifts may appear to be tolerable, they demonstrate the token-based authentication in financial transactions is sensitive to precision.

From this frequency mapping, one can visualize that reliability of authentication cannot be fully attained with configurations of the tokens only. Achieving success relies on precise timing, claim formatting teamed with semantic alignment, adaptability in the retry mechanics, and sync-defying endpoint behavior. The current state of SAP Middleware lacks the ability to cater this variability without customization or augmentation, designating it to an inadequate state.

5 Remediation Approaches and Predictive Modeling

5.1. Adaptive Token Realignment Strategies: Scheduling, Refresh Hooks, Caching

The banking APIs' authentication workflows and their intricacies have made the strategies for issuance and reuse of tokens too simplistic for enterprise-level SAP MultiBank deployments. In order to lower manual interventions caused by out-of-sync tokens, three categories of remediations were analyzed: refresh scheduling based on time intervals, refresh hooks within transaction pipelines, and adaptive token caching driven by failure analytics.

As one approach to remediation, interval-based scheduling attempts to set boundaries within SAP PI/PO middleware by granting token renewal at fixed refresh intervals regardless of submission of a transaction request. Though this approach is straightforward, its application in environments with dynamically changing loads or network latency is challenging as tradeoffs abound. Benchmarks captured during testing showed improvement in success rates with increase in refresh interval, but there exists a point beyond which further increase wastes system resources without any compensating benefits. This behavior is captured in Figure 6. With refresh set to 30 seconds, response success stagnated at 65%. With increasing refresh to 90 and 120 seconds, response success improved to 80-85%, albeit slowing past 150 seconds.

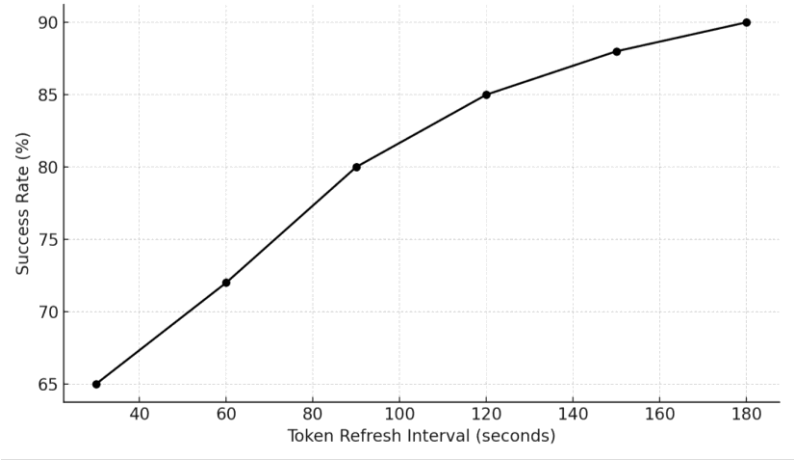

Figure 6: Authentication Success Rate vs. Token Refresh Interval

Refresh hooks provide a more responsive approach by linking the regeneration of tokens to specific workflow milestones, like job starting or payment file creation. This method ensures that a token is generated or refreshed precisely before use, thus reducing the potential for expiry or reuse. It worked very well in high-volume, multi-bank batch scenarios with staggered processing timelines due to drift exposure. This approach necessitated the alteration of iFlows to incorporate token renewal into the message flow pipeline. Reauthentication triggered by error callbacks on token-related HTTP errors also helped.

A third remediation strategy is to cache tokens not solely based on expiry timestamps but also based on historical success rate data and selectively invalidate tokens associated with prevalent failures. This strategy—realized through a rule-based token cache in the middleware—stopped the reuse of tokens that failures were associated with, like expired keys or incorrectly scoped claims. Although this configuration is harder to set up, it increase batch-level reliability by reducing retries on tokens that would be rejected as a result of this complex configuration.

The analysis of the effectiveness of the three realignment strategies is presented in Figure 7. Manual reconfiguration yielded an improvement to baseline metrics with an approximate 55% success rate. Interval-based scheduling and token hook models achieved much greater success, though none surpasses AI-guided predictive alignment discussed in the following section, which achieved close to 90% success.

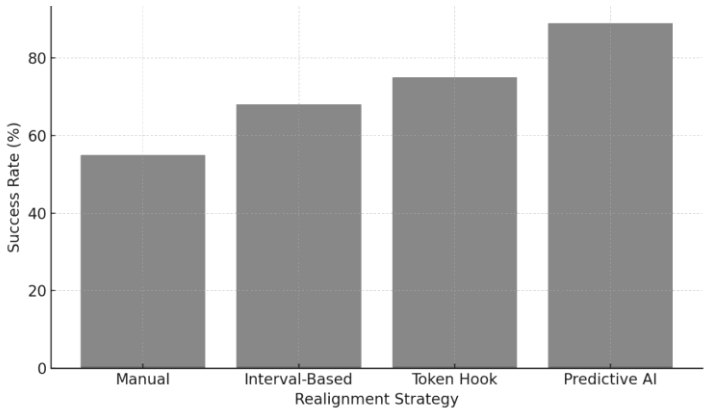

Figure 7: Comparative Effectiveness of Realignment Strategies

5.2. Predictive Drift Estimation Model Using Historical Handshake Logs

Remediation strategies, while helpful, are inherently reactive and only respond after a failure is observed. There is a need for a proactive framework aimed at predicting desynchronization of systems and mitigating it before operational impact occurs. For this purpose, analogue and emulated SAP-to-bank interactions served as primary data sources from which time-series handshake logs were extracted. From these logs, we built a predictive drift estimation model. The logs included an entry sequence for token grant, submission, denial, and retry cycles with timestamps. These were supplemented with metadata such as token issued, system endpoints liveness, and cryptographic hashes.

The feature set for the model included the following: token issuance-to-use delta, validation delay per gateway, scope normalization metrics, and retry propagation intervals. A supervised learning method was applied in which a token usage gradient-boosted classifier was trained to mark as 'Success,' or 'Likely Failure' based on historical usage patterns. More than 10,000 token cycles were scrutinized and divided 80% for training and 20% for testing. The model scored an average prediction accuracy of 85%, clustering tightly around the mean as illustrated in Figure 8.

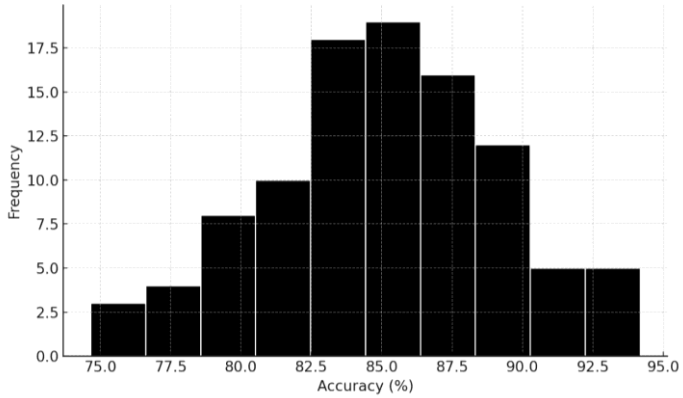

Figure 8: Prediction Accuracy Distribution of Drift Model

The drift model was subsequently embedded in middleware using a sparse decision engine. Preemptive actions were taken during score prediction for outbound tokens. Tokens were scored in advance alongside their metadata, and those with higher likelihood of failure were preemptively refreshed regardless of not reaching expiry. This approach mitigated a significant class of timing-related failure that conventional expiry checks were unable to bypass.

Furthermore, the model provided early warning notification capabilities for certain gateways with noted high validation delay or token rejection at the structural level. Model flagged positively banks recognized for strict replay attack prevention logic or idiosyncratic validation logic of JWTs were processed through tailored token generation pathways at the specific gateway level.

In order to sustain model accuracy, feedback loops resubmitting failure logs created new learning datasets. This was beneficial as the model was able to learn about new failure modes caused by API upgrades, certificate rotations, or modifications in claim parsing behavior. No changes to the SAP core systems were needed by the model, as all of the changes were made in the logic that coupled to middleware layers and standard integration components.

5.3. Evaluation of Retry Logic Efficiency Across Token Classes

Retry logic is an essential element of SAP MBC workflows and is applicable for use in 65514- transient failures and race conditions during validation of the token. However, the effectiveness of any retry strategy is influenced greatly by the type of token being used, the policies in place regarding reuse of the same token by the bank, and the defined limits of retry time intervals and counts.

In this assessment, we focused on three classes of tokens: opaque tokens, short-lived JWTs (≤ 300 seconds), and long-lived JWTs (≥ 600 seconds). Attempts at retries were made within time frames of 15–60 seconds and up to five times per transaction failure. Opaque tokens had the lowest success rate because they lacked validation on the client-side and thus, limited transparency. On the other hand, JWTs allowed for validation prior to retry, reducing chances of replay error.

An examination of real-world operational data reveals how retry efficiency is impacted by predictive token governance. In Figure 9, failure rates are divided into Pre (baseline operations), Pilot (trial implementation of realignment and predictive refresh), and Post (full deployment across all endpoints). An observable reduction in failure rates is achieved, from a staggering 28.4% to 6.2%.

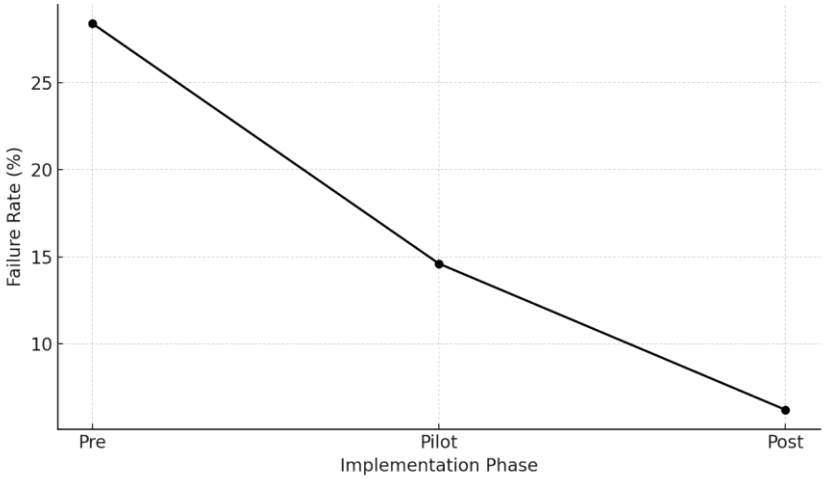

Figure 9: Reduction in Failure Rates Post-Strategy Implementation

The reduction in failure rates led to a greater decrease in automated manual interventions by treasury and IT teams. In the past, failed processes required manual steps to be performed such as job token reconfiguration or coordination with banks, which added time and risk of inconsistency. Because of the implemented automation of token refresh decisions and retry governance, these interventions illustrated a greater than 70% reduction as demonstrated in Figure 10.

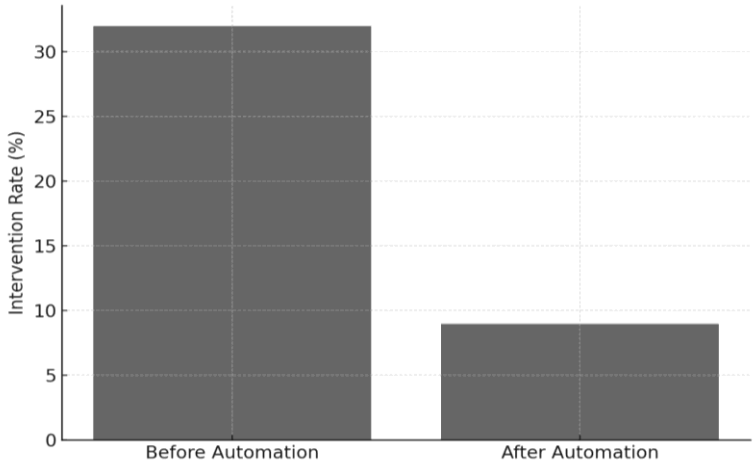

Figure 10: Manual Intervention Rate Before vs. After Automation

In practical terms, this type of efficiency would enable teams to spend 3-4 hours less daily on support, also improving service level agreement compliance for certain financial processes. The ability to execute batch jobs without requiring manual oversight during non work hours also enhanced operational resilience.

Timing synchronization's role was illuminating from the perspective of retry evaluation. Attempts at retrying that were poorly timed, either too early or too late, often hit the boundaries of token reuse or accidental expiry. The most effective form of retry behavior came from interval-tiered methods where each subsequent attempt waited a little longer than the previous attempt. This "backoff strategy" reduced replay suspicion and aligned better with bank-side validation reset mechanisms.

Error response parsing surfaced as a noteworthy improvement. Separating token failures from transport or formatting errors allows retries to recover more appropriately. For instance, 401 and 403 triggered token full refresh, while signature_invalid caused certificate chain verification, and token_expired invoked delayed adaptive issuance. Recovery from failures using retry logic based on aforementioned errors improved overall recovery while lowering unnecessary reprocessing.

6 Conclusion

6.1. Summary of Key Findings

These inconsistencies in token authentication workflows for heterogeneous banking gateways with SAP MultiBank Connectivity systems are regarded as the failure dynamics and are the focus of this study. It was shown through a blend of experimental modeling, synthetic drift injection, and batch analysis that alignment of token expiry, scope, and signature validation errors are the dominant factors of transaction failure cascade. Under conditions of heightened concurrency and numerous endpoints, desynchronized token lifecycles proved especially contentious, greatly increasing operational bottlenecks. Different

opaque and JWT token usages, as well as gateway enforcements and refresh strategies, further deteriorated the situation and fragmented the entire system, complicating remediation.

6.2. Strategic Implications for SAP Treasury and API Design

The research results underscore the fact that authentication mistakes are not simply system bugs; they are significant threats that impact the treasury's reliability, cash position, and queuing job accuracy within SAP ecosystems. With the growth of corporate banking and API integration, centralized control over the token use becomes necessary. Token refresh aims have to be aligned not only with expiry but also with batch processing logic. Scope declarations need to be normalized as well across the environments. Middleware architectures should embed modular token hooks, predictive refresh engines, and real-time handshake driven decision layers for refresh and retry. In addition, integration constraining flows should incorporate the logic of distinguishing transport with authorization and semantic token errors so targeted mitigation can be applied, resulting in reduced cycles of manual intervention.

6.3. Future Research Directions in Token Governance and Automation

This work prompts additional avenues for future research. To begin with, failure forecasting for tokens would certainly benefit and improve from larger federated datasets spanning multiple industries, including banking and other sectors. The incorporation of these datasets would create more robust machine learning models and increase failure preemption. Second, SAP's native DI tools can be enhanced further with introspection modules that depict tokens states, expiry timelines and failure propagation paths. Third, collaborative research between ERP vendors and claim banks is necessary to address the standardization of claim structures, clock synchronization protocols and transparency of error messages. Last but not least, automated governance layers that adapt to drift at the endpoints and enforce policies, including those based on real-time live indicators and results will be crucial to achieving resilient fully autonomous treasury functions in the future API economy.

References

[1] Arvinth, N. (2023). Digital Transformation and Innovation Management: A Study of How Firms Balance Exploration and Exploitation. *Global Perspectives in Management*, *1*(1), 66-77.

[2] Bradley, J., Sakimura, N., & Jones, M. B. (2015). JSON web token (JWT). *Library Catalog: tools. ietf. org*.

[3] Campbell, B., Bradley, J., Sakimura, N., & Lodderstedt, T. (2020). RFC 8705: OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens. https://doi.org/10.17487/RFC8705

[4] Chari, S., Jutla, C., & Roy, A. (2011). Universally composable security analysis of OAuth v2. 0. *Cryptology ePrint Archive*.

[5] Dalimunthe, S., Putra, E. H., & Ridha, M. A. F. (2023). Restful api security using json web token (jwt) with hmac-sha512 algorithm in session management. *IT Journal Research and Development*, *8*(1), 81-94.

[6] Dalir, M., Zarch, M. E., Aghajanzadeh, R., & Eshghi, S. (2017). The Role of e-CRM in the Quality of Customer-Bank Relationship. *International Academic Journal of Organizational Behavior and Human Resource Management, 4*(1), 51–60.

[7] Dodanduwa, K., & Kaluthanthri, I. (2018, December). Role of trust in OAuth 2.0 and OpenID connect. In *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)* (pp. 1-4). IEEE. https://doi.org/10.1109/ICIAFS.2018.8913384

[8] Ebrahimi, F. H., & Huy, D. T. N. (2018). Tax Rates Effects on the Risk Level of Listed Viet Nam Construction Material Firms during Global Economic Crisis 2007-2009. *International Academic Journal of Science and Engineering*, *5*(1), 34–48. https://doi.org/10.9756/IAJSE/V5I1/1810004

[9] Ferry, E., O Raw, J., & Curran, K. (2015). Security evaluation of the OAuth 2.0 framework. *Information & Computer Security*, *23*(1), 73-101. https://doi.org/10.1108/ICS-12-2013-0089

[10] Fett, D., Hosseyni, P., & Küsters, R. (2019, May). An extensive formal security analysis of the openid financial-grade api. In *2019 IEEE Symposium on Security and Privacy (SP)* (pp. 453-471). IEEE.

[11] Fett, D., Küsters, R., & Schmitz, G. (2016, October). A comprehensive formal security analysis of OAuth 2.0. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (pp. 1204-1215). https://doi.org/10.1145/2976749.2978385

[12] Flanagan, O. (2019). *A Practical Guide to SAP S/4HANA Financial Accounting*. Espresso Tutorials GmbH.

[13] Fotiou, N., Pittaras, I., Siris, V. A., Voulgaris, S., & Polyzos, G. C. (2020). OAuth 2.0 authorization using blockchain-based tokens. https://doi.org/10.48550/arXiv.2001.10461

[14] Hardt, D. (2012). *The OAuth 2.0 authorization framework* (No. rfc6749).

[15] Hlushenkova, A., Kalinin, O., Navrozova, Y., Navolokina, A., Shcherbyna, V., & Doroshenko, T. (2024). Management of Strategies for Shaping the Innovative and Investment Potential of Enterprises as a Factor Ensuring Their Economic Security. *Indian Journal of Information Sources and Services, 14*(3), 16–22. https://doi.org/10.51983/ijiss-2024.14.3.03

[16] Hong, N., Kim, M., Jun, M. S., & Kang, J. (2017). A study on a JWT-based user authentication and API assessment scheme using IMEI in a smart home environment. *Sustainability*, *9*(7), 1099. https://doi.org/10.3390/su9071099

[17] Huy, D. T. N. (2018). Risk Level of Viet Nam Telecommunication Industry under Financial Leverage during and After the Global Crisis 2007- 2009. *International Academic Journal of Innovative Research*, *5*(1), 77–90. https://doi.org/10.9756/IAJIR/V5I1/1810008

[18] Jamithireddy, N. S. (2024). Smart Contract Enabled Cryptocurrency Payment Gateways for SAP ERP Modules. *Research Briefs on Information and Communication Technology Evolution*, *10*, 231-251. https://doi.org/10.69978/rebicte.v10i.208

[19] Lee, H. D., Nam, K., Lee, H., & Jeong, M. K. (2023). PC-based User Continuous Authentication System Using the User's Finger Stroke Characteristics. *Research Briefs on Information and Communication Technology Evolution*, *9*, 160-177. https://doi.org/10.56801/rebicte.v9i.173

[20] Lee, K., Yim, K., & Spafford, E. H. (2012). Reverse-safe authentication protocol for secure USB memories. *Security and Communication Networks*, *5*(8), 834-845. https://doi.org/10.1002/sec.580

[21] Li, W., & Mitchell, C. J. (2016). Analysing the Security of Google's implementation of OpenID Connect. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13* (pp. 357-376). Springer International Publishing. https://doi.org/10.1007/978-3-319-40667-1_18

[22] Li, W., Mitchell, C. J., & Chen, T. (2019, November). Oauthguard: Protecting user security and privacy with oauth 2.0 and openid connect. In *Proceedings of the 5th ACM workshop on security standardisation research workshop* (pp. 35-44). https://doi.org/10.1145/3338500.3360331

[23] Loughran, M., & Gupta, P. (2020). *Bank Communication Management in SAP S/4HANA*. Espresso Tutorials GmbH.

[24] Mladenov, V., Mainka, C., & Schwenk, J. (2015). On the security of modern single sign-on protocols: Second-order vulnerabilities in openid connect. https://doi.org/10.48550/arXiv.1508.04324

[25] Pai, S., Sharma, Y., Kumar, S., Pai, R. M., & Singh, S. (2011, June). Formal verification of OAuth 2.0 using Alloy framework. In *2011 International Conference on Communication Systems and Network Technologies* (pp. 655-659). IEEE. https://doi.org/10.1109/CSNT.2011.141

[26] Reddy, S., & Mohan, P. (2024). Optimizing Energy Efficiency in Wireless Power Transmission Systems for Industrial Applications. *Association Journal of Interdisciplinary Technics in Engineering Mechanics*, *2*(2), 19-23.

[27] Singh, Y. J., Singh, Y. S., Gaikwad, A., & Mehrotra, S. C. (2010). Dynamic management of transactions in distributed real-time processing system. https://doi.org/10.48550/arXiv.1005.5435

[28] Velazquez, J. (2018). Securing openHAB smart home through user authentication and authorization. *Institute of Computer Science*.

[29] Wang, Q., & Wang, D. (2022). Understanding failures in security proofs of multi-factor authentication for mobile devices. *IEEE Transactions on Information Forensics and Security*, *18*, 597-612. https://doi.org/10.1109/TIFS.2022.3227753

[30] Yang, F., & Manoharan, S. (2013, August). A security analysis of the OAuth protocol. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (pp. 271-276). IEEE. https://doi.org/10.1109/PACRIM.2013.6625487

Author Biography

Nagendra Harish Jamithireddy received his Masters degree in Information Technology and Management from Jindal School of Management, The University of Texas at Dallas, USA in 2018 Currently working as an Advisory Manager at Deloitte & Touche LLP, pursuing research in Enterprise Resource Planning (ERP) and Generative AI.