International Journal of Advanced Engineering, Management and Science (IJAEMS)
Peer-Reviewed Journal
ISSN: 2454-1311 | Vol-12, Issue-3; May-Jun, 2026
Journal Home Page: https://ijaems.com/
DOI: https://dx.doi.org/10.22161/ijaems.123.38

Standardisation of APIs and Security Protocols in the Integration of Banking Payment Solutions with Retail Partner Infrastructure

Aditya Agarwal

Irving, USA

Received: 15 May 2026; Received in revised form: 12 Jun 2026; Accepted: 17 Jun 2026; Available online: 23 Jun 2026

Abstract— *The article examines the standardization of APIs and security protocols in the integration of banking payment solutions with retail partner infrastructure. The aim of the study is to systematize architectural, procedural, and security approaches that govern the boundary between banking payment platforms and retailer systems. The relevance of the topic stems from the growth of embedded finance, co-branded card programmes, and partner-facing banking platforms, where integration failures may affect revenue, trust, auditability, and regulatory exposure. The novelty of the article lies in the proposed core-and-periphery model, which separates the standardized banking core from the controlled customization zone of retail partners. The main findings show that mature integration architectures require OpenAPI-based contracts, TLS 1.3, mTLS, OAuth 2.x, OpenID Connect, FAPI profiles, JWS, JWE, ISO 20022, PCI DSS, ISO/IEC 27001, and NIST-aligned API security practices. The study concludes that scalable partner onboarding depends on reusable artifacts, certification gates, unified observability, SLOs, incident runbooks, and coordinated release governance. The article will be useful for researchers, banking architects, API security specialists, payment platform teams, and retail partners involved in embedded finance integrations.*

Keywords— API standardisation, banking integration, retail payments, partner onboarding, embedded finance.

I. INTRODUCTION

The digitisation of retail and the spread of co-branded card programmes have changed how a bank is perceived by a large merchant. The bank has moved beyond the role of a card issuer and now operates as a platform that admits dozens of external companies with distinct infrastructures, business cycles, and IT standards. Embedded finance accelerates this shift. A financial product can be initiated from inside a retailer application or website at the point of sale, without the customer ever opening a banking app (Ozili, 2022). The centre of gravity of digital transformation has shifted from internal banking systems to the integration layer that interfaces with external partners.

A retail partner under such an arrangement is neither a customer nor a contractor in the classic sense. The partner is a technological counterparty, and the banking payment platform must open outward while sustaining the security and regulatory robustness expected of a financial institution of systemic significance (Preziuso et al., 2023). Any defect originating in the integration layer affects partner revenue, damages the bank's reputation, and can trigger regulatory consequences.

The economic stakes of the integration layer have grown alongside its complexity. A single co-branded card programme with a national retailer can process billions of dollars in annual purchase volume, and the integration that supports it handles

authorisation, settlement, dispute resolution, statement delivery, and customer service across multiple touchpoints. Each touchpoint corresponds to one or several APIs that cross the bank-to-retailer boundary, and each of these APIs becomes a candidate for standardisation.

The same forces that pull the bank toward platform behaviour also pull the retailer into closer alignment with banking practices around security, audit, and change management. Retailers that treat the bank-side connection as a generic vendor integration accumulate operational debt and expose themselves to risks for which their internal teams hold limited training. Retailers that adopt the integration as a first-class element of their architecture gain access to a broader catalogue of co-branded financial products and shorter time-to-launch on each subsequent connection.

Two opposing forces meet at the boundary between the bank and the retail partner. The bank seeks uniformity through common API contracts, a unified security model, and consistent audit and change-management requirements. The partner seeks flexibility because each retailer maintains its own technology stack, release calendar, UX logic, and seasonal peak periods, such as Black Friday, Cyber Monday, and the December holiday season, during which any outage results in immediate revenue loss.

Without deliberate standardisation, each integration becomes a bespoke project with a distinct attack surface, defect catalogue, and cost of ownership. The aggregate complexity of such an architecture grows nonlinearly with the number of partners, and the integration layer becomes the most fragile and most expensive part of the platform (Ali & Salih, 2025).

The aim of the paper is to systematise approaches to the standardisation of APIs and security protocols in integrations between a banking payment platform and the infrastructure of retail partners. Four objectives serve this aim. The first objective maps the current landscape of standards and specifications relevant to the integration layer. The second objective identifies the architectural patterns that shape such a layer. The third objective is to develop a model that balances bank-side standardisation with partner-side customisation. The fourth objective examines partner onboarding practices and the operational resilience of the resulting integrations.

The study focuses on the integration layer between a bank's payment platform and a retail partner's IT infrastructure. The subject of analysis is the set of API standards and security protocols that govern this layer. Three areas remain outside the scope of the paper. The first excluded area concerns integrations with consumer IoT devices. The second excluded area covers general compliance and risk management questions that do not arise from the integration layer itself. The third excluded area treats mobile wallets as an independent architectural topic.

Section 2 describes the review methodology and the source base. Section 3 presents the analytical results in five parts: the standards landscape, architectural patterns, the core-and-periphery model, partner onboarding, and operational resilience. Section 4 interprets these findings, contrasts them with the open banking paradigm, addresses the principal trade-offs, and draws practical implications. Section 5 states the conclusions of the work.

II. METHOD

The work follows the conventions of a narrative review with an applied analytical layer. The choice of format reflects the nature of the object under study. The integration layer between a bank and a retail partner sits at the intersection of several fast-moving domains, including API standards, security protocols, regulatory requirements, and industry practices. Empirical characterisation of this layer is feasible within the institutions that operate it and remains beyond the reach of external observers. Reproducible quantitative measurement by an external researcher is constrained because internal API call logs, incident statistics, and performance metrics remain closed data protected by non-disclosure agreements (Sukhera, 2022). The analytical review format is therefore a defensible methodological choice for the topic.

Four classes of sources support the work. The first class comprises active international standards and specifications. The second class consists of peer-reviewed publications from recent years that address API security, enterprise integration patterns,

embedded finance, and partner-facing payment platforms. The third class includes industry reports from analytical agencies and regulators that describe trends in open banking and embedded financial services. The fourth class contains industry cases documented in the public domain.

Three criteria guided source selection. The first criterion was thematic relevance, with priority given to material focused on the bank-to-retailer integration layer or to neighbouring domains whose patterns transfer to this layer. The second criterion was recency, with preference for publications and standard revisions issued within the last five years, while preserving classic works on integration architecture. The third criterion was authority, expressed through reliance on peer-reviewed venues, recognised standard-setting organisations, and established analytical agencies.

The method of analysis combined thematic categorisation and comparison of approaches. The first step grouped material by integration stack layers that span the API contract, transport, authorisation, message format, and operational processes. The second step identified points of agreement and divergence across sources, as well as recurring industrial patterns. The third step formulated a synthesising model, which appears in Section 3 (Turnbull et al., 2023).

Three limitations follow from the chosen format. The first limitation concerns published quantitative effects of standardised integration. Industrial reports cite typical ranges of twenty to thirty per cent in the reduction of customer-impacting incidents and in time-to-market acceleration. These figures serve as illustrative examples and do not constitute reproducible experimental measurements because the underlying measurement methodology is not publicly disclosed. The second limitation is the absence of internal metrics from issuing banks or their partners, and the paper does not attempt to reconstruct such metrics from indirect signals. The third limitation arises from regional coverage. The review draws on the regulatory environments of the United States and Europe, where the practice of standardising the integration layer has reached the highest level of maturity, and the transfer of conclusions to other jurisdictions requires separate verification.

III. RESULTS

The standards landscape in the bank-to-retailer integration layer

Standards relevant to the bank-to-retailer integration layer become tractable through a stack-based view that runs from API contract and transport to payment message semantics and security management. Each layer of the stack contains one or more consolidated standards, and together they form the core of the contemporary banking integration architecture (Gounari et al., 2024).

The contract layer has settled around REST and JSON, paired with OpenAPI 3.x as the machine-readable specification. OpenAPI is used for contract testing, computer code generation, and automated cross-version compatibility testing. On the transport side, TLS 1.3 is the minimum version. Server-to-server communication between the bank and the partner is done using mutual TLS, which involves certificates on both sides.

Delegated authorization and end-user identification are based on the OAuth 2.0 and 2.1 family of specifications with OpenID Connect. Financial-grade API profiles 1.0 and 2.0 provide a security-hardened profile of these specifications for financially critical APIs. These profiles require Proof Key for Code Exchange, sender-constrained tokens and signed authorisation responses through JARM (Hosseyni et al., 2024). The message layer builds on JWS and JWE to provide signed and encrypted payloads. This is particularly useful when there are intermediate nodes on the message path.

The payment-semantic layer converges on ISO 20022 as a universal vocabulary for financial messaging. This convergence simplifies both integration work and integration audits. The compliance layer implements PCI DSS v4.0 across every flow that processes, transmits, or stores card data, and ISO/IEC 27001 serves as the management system for the security operation. An additional layer of growing influence comes from NIST recommendations on the security of microservices and APIs in SP 800-204, which establish a baseline for architectural decisions. The standards landscape in the bank-to-retail-partner integration layer is illustrated in Table 1.

Table 1. Standards landscape in the bank-to-retail-partner integration layer

Integration layer	Standard or specification	Role in the bank-to-retailer integration
API contract	OpenAPI 3.x	Machine-readable API description that supports contract testing and SDK generation
Transport	TLS 1.3, mTLS	Channel encryption and mutual authentication for server-to-server interactions
Authorisation	OAuth 2.0 / 2.1, OpenID Connect	Delegated authorisation and end-user identity propagation
Financial-critical APIs	FAPI 1.0 / 2.0	Hardened OAuth profile with PKCE, sender-constrained tokens, and signed authorisation responses
Message security	JWS, JWE	Signature and encryption applied at the payload level of API messages
Payment messaging	ISO 20022	Unified semantics of financial messages across payment ecosystem participants
Cardholder data protection	PCI DSS v4.0	Mandatory controls for the processing, transmission, and storage of card data
Security management	ISO/IEC 27001	Information security management system shared by the bank and its partners
API security baseline	NIST SP 800-204	Reference guidance for the security of microservices and API ecosystems

Architectural patterns of the integration layer

Standards establish what must hold, while architectural patterns determine how the elements are arranged. Banking platforms have validated several such patterns through repeated use. The base element is the API gateway, which serves as a single entry point and handles authentication, authorisation, rate limiting, basic validation, and telemetry collection (Venčkauskas et al., 2023). The gateway removes recurring work from product API teams and delivers a uniform security layer for all partners.

Adaptation of the common core to the specifics of a given partner relies on the Backend-for-Frontend pattern. The BFF layer translates unified internal APIs into the formats and flows of a particular channel. This layer is the natural place where customisation remains acceptable because the BFF does not erode the standard. The BFF translates the standard into a form that suits the partner while staying under bank governance.

Identity information is exchanged between services via tokens issued by the bank authorisation server, which take the form of JSON Web Tokens. Modern implementations rely on sender-constrained tokens via DPoP or mTLS binding, rendering a stolen token useless to an attacker who does not control the matching key pair. Key and secret management migrates into specialised systems such as hardware security modules and vault services.

Synchronous APIs do not cover every scenario. The partner needs asynchronous notifications about transaction state and cannot poll the bank without limit. This need introduces an event-driven layer with webhooks for lightweight notifications and message brokers for higher-volume streams. The integration layer between the bank and the retailer appears as a coordinated combination of synchronous and asynchronous communication that shares a common security envelope (Lercher et al., 2024).

Idempotency is a separate concern that shapes the design of every write-oriented API at the boundary. Network glitches, partner-side retries, and duplicate user actions all produce duplicate requests. The bank API contract must absorb these duplicates without double-charging the customer or creating

phantom transactions. The standard solution involves an idempotency key supplied by the partner together with each request, which the bank stores for a defined retention window and uses to recognise repeated submissions. The contract specifies the lifetime of such keys, the API's behaviour in the event of key collisions, and the rules for partial failure recovery.

Versioning of the public API contract is as important as idempotency. The bank serves multiple partners on different release schedules, and a breaking change to a shared API forces a coordinated migration across the entire partner base. Mature platforms commit to backwards-compatible additive evolution as the default mode of change, with major version bumps reserved for cases of true incompatibility. The contract describes the deprecation policy, the minimum overlap window during which two versions remain available in parallel, and the channels through which the bank communicates upcoming changes to partner technical teams. The bank-to-retail-partner integration layer reference architecture is shown in Figure 1.

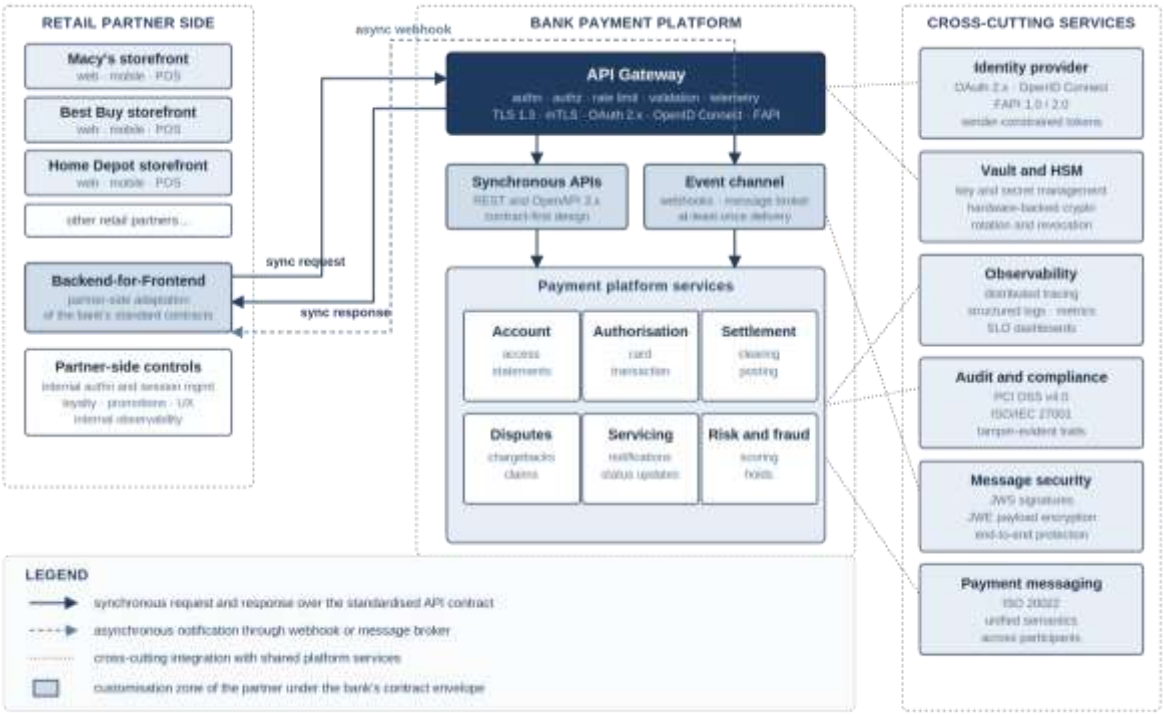

Fig. 1. Reference architecture of the bank-to-retail-partner integration layer

The standardised core and the controlled periphery

The central analytical construct that emerges from the review is an explicit separation of the integration layer into a standardised core and a controlled periphery. The principle is straightforward. Everything that touches security, audit, data contracts, and regulatory exposure belongs to the bank as the platform owner and remains identical for every partner. Everything that concerns user experience, marketing flows, and the partner's internal logic stays in the partner's zone of freedom, provided that any crossing of the integration boundary follows the agreed-upon rules (Chung, 2025).

The model becomes tangible through the example of a typical partner-facing payment platform. CitiPay represents one such case that has reached public documentation. The platform grants customers approved for co-branded cards from large retailers, including Macy's, Best Buy, and Home Depot, immediate digital access to the account, without waiting for the physical card to arrive. The partners run different technology stacks and follow different release cadences, and they connect through the same bank APIs and operate under the same security and audit requirements. The published description of the

platform states that the bank provides standardised APIs and security protocols, leaving partners with flexibility in implementing their side of the integration.

The practical value of the core-and-periphery model lies in the clear criterion it offers for design decisions. Any proposal to move a given item into the customisation zone for partner X faces a layer test. Items that touch security, contracts, or audit violate the core and must be refused, even when refusal slows down the integration. Items that concern UX, loyalty rules, or internal sequencing constitute legitimate customisation and need no review by the bank architecture committee. The standardised core and the controlled periphery across integration layers are given in Table 2.

Table 2. The standardised core and the controlled periphery across integration layers

Layer	Standardised core under the bank	Customisation zone under the partner
Security	Authentication, authorisation, encryption, key handling, and audit are fixed	Internal controls protecting the partner perimeter
Data contracts	Request and response schemas, error model, versioning policy	Mapping into internal partner models and extensions outside required fields
Business logic	Product issuance rules, scoring, regulatory checks	Loyalty programmes, promotional flows, product bundles
UX and channels	Minimal design guidance for joint branding	Visual design, copy, navigation across the partner storefront
Releases	Release windows and freezes during high-volume periods	Internal release calendar and partner-side experimentation
Observability	Standard metrics, distributed tracing, common log schema	Dashboards and alerts inside partner tooling

Partner onboarding framework

Standards and architectural patterns are put into practice through a formalised process for connecting a new partner. A mature partner onboarding framework, as documented across industry practice, includes several mandatory stages. Sandbox access through a realistic and anonymised test environment opens the process. Contract testing is based on the OpenAPI specification. The next stage is a security assessment that verifies the mTLS configuration, certificate validity, and token-handling policies. Independent third-party testing applies to high-criticality integrations. Formal certification gates determine whether the partner gains access to the production environment, and the process closes with a technical cutover plan agreed upon by the bank and the partner (Daiy et al., 2021).

The maturity of such a framework is measured by the share of reusable artefacts rather than the number of stages. The greater the proportion of partners who pass onboarding through the same templates and automated checks, the lower the marginal cost of each subsequent integration and the higher its predictability. Industrial reports that describe multi-partner payment platforms cite reductions in time-to-market for new integration capabilities in the range of twenty to thirty per cent. These values are useful as order-of-magnitude indicators. They do not constitute precise measurements.

Operational resilience of integrations

An integration that has passed onboarding enters production operation, where operational resilience determines its long-term value. Formalised SLO and SLA models for partner-facing APIs underpin this resilience. The targets cover availability, latency percentiles, and the success rate of requests. These indicators serve as contractual commitments that drive financial accountability mechanisms for both parties.

Observability of the integration layer rests on three coordinated axes. The first axis is distributed tracing across the bank and partner perimeters, with a shared format for trace identifiers. The second axis is

structured logging with a fixed schema. The third axis is business-level monitoring of key scenarios such as the completion of a payment transaction, which differs from the success of an HTTP call (Yu et al., 2026).

Incident management requires runbooks agreed in advance, a unified severity scale, and protocols of communication with the partner that specify who notifies whom, through which channel, and within which deadline in the event of degradation. Release sequencing also gains weight under the retailer calendar. Release windows fall outside high-volume sales periods, and formal code freezes are in place around Black Friday, Cyber Monday, and the December holiday peak. The joint planning of release windows between the bank and partners stands as a separate artefact of integration governance.

Disaster recovery and business continuity planning extend the resilience contour beyond day-to-day operation. The bank and the partner each maintain their own recovery procedures, and the integration layer demands that these procedures align on at least three points. The first point of alignment concerns the recovery time objective for partner-facing APIs, which the partner uses as input to its own continuity model. The second point of alignment concerns synchronising failover procedures, since an unannounced regional failover on one side can break long-lived sessions and webhooks on the other. The third point of alignment addresses the joint communication protocol during a major incident, including the responsible roles, the cadence of status updates, and the shared incident timeline.

Blast-radius containment shapes the architectural decisions that surround the integration layer. A failure in one partner integration must not cascade to other partners. Separation of resources at the API gateway tenant, identity provider configuration, message broker topic, and rate-limit pool levels prevents the noise from one partner from degrading the experience of others. The bank treats partner-level isolation as a non-negotiable property of the integration platform and invests in tooling that makes it observable and testable under realistic load.

Operational resilience of integrations depends on the joint contour that links SLOs, observability, incident management, and the release calendar. Each component of the contour relies on standardised data that originates in the core of the integration layer (Tanveer et al., 2025). The operational resilience contour is shown in Figure 2.

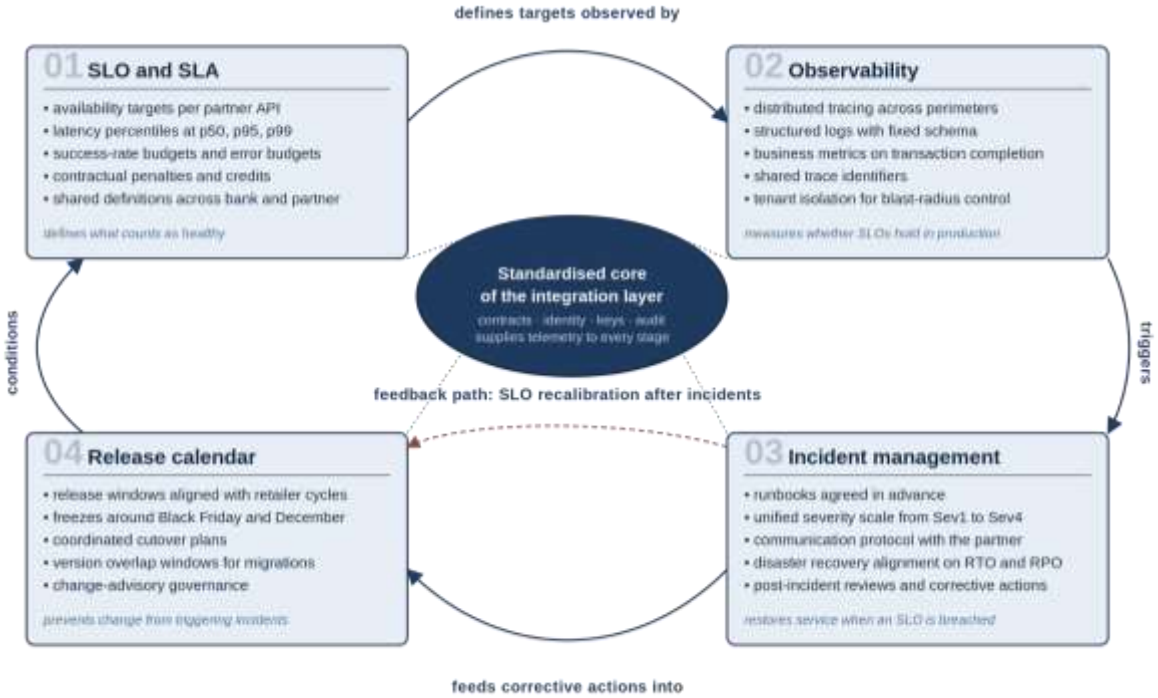

Fig. 2. The operational resilience contour: SLOs, observability, incident management, and the release calendar

IV. DISCUSSION

Interpretation of key patterns

The review suggests that the largest payoff from standardisation occurs at the lower layers of the integration stack, where partners themselves have the weakest incentive to develop local solutions. Standardising the security baseline to include TLS 1.3, mTLS, OAuth 2.x and FAPI profiles, and shared requirements for keys and tokens delivers more value than standardising UX. Such security standardisation removes a class of systemic defects. Standardised UX, in turn, clashes with partner brands and demands more enforcement effort than it returns in value.

The boundary between mandatory and recommended elements depends on whether the underlying risk can spread beyond a single integration. Risks that can propagate to the platform as a whole, including the compromise of keys, the breakdown of audit, or version drift across APIs, must remain inside the standardised core. Risks confined to the perimeter of one partner can stay under partner control without turning the integration committee into a bottleneck for the whole programme.

Comparison with the open banking paradigm

The open banking paradigm, represented by PSD2 in Europe, Open Banking in the United Kingdom, and similar initiatives in Australia and Brazil, serves as the closest methodological neighbour of the topic under review, and the differences warrant a clear statement. Open banking standardisation is imposed by the regulator and constitutes an external requirement. The bank must open a defined set of APIs to third parties in accordance with an approved specification. Standardisation of integrations with retail partners flows from the bank itself, as the platform owner, leaving this standardisation as an internal, voluntary regime shaped by the partner programme's business logic. Public policy on access to financial data does not play a governing role in this case (Hosseyni et al., 2024).

The distinction has practical consequences. Under open banking, the level of security standardisation is set at a high bar through FAPI, eIDAS certificates in the European Union, and mandatory strong customer authentication requirements, and the bank is required to implement them. Retail-partner integrations grant the bank greater freedom, along with greater responsibility for setting an adequate level of protection. Retail-partner arrangements also admit deeper customisation of business logic than the regulated open banking APIs do, and this latitude should be used where the regulator does not prescribe the form. The same latitude should not turn each connection into an exception that erodes the platform.

Trade-offs

Each architectural decision in the integration layer resolves a triple tension between speed, control, and partner experience. Excessive standardisation increases control while lengthening the time required to admit a new partner and reducing partner satisfaction. Excessive flexibility accelerates a single connection while it accumulates architectural debt and raises the chance of incidents with system-wide consequences. The middle path, combining a fixed core with an explicit zone of freedom, provides speed without sacrificing control. The middle path requires continuous work to maintain the boundary between the two zones.

A similar trade-off shapes the choice of security depth. The addition of message-level signing and encryption through JWS and JWE strengthens resistance to man-in-the-middle attacks across chains of intermediaries. Message-level cryptography also raises the entry barrier for the partner and lengthens incident diagnosis. The decision applies on a per-flow basis. Payment confirmation justifies the cost. Low-value status notifications fail to clear the same threshold.

Practical implications

The review yields a maturity checklist for banks that operate or design partner-facing payment platforms. Architectural maturity covers an explicit core of standards, a single API gateway, a unified identity model, and centralised key management. Process maturity covers a formalised partner onboarding framework with automated checks and certification gates. Operational maturity covers agreed SLOs and SLAs, unified observability, a common incident scale, and a joint release calendar. Regulatory maturity encompasses the ability to present an auditor with an end-to-end view of who acted, what they did, when the action occurred,

through which contract, and under which identifier within the integration layer.

The same review suggests that retail partners should treat the connection to a mature banking platform as the acceptance of long-term obligations. The connection no longer fits the one-off IT project model. Such obligations include adherence to security standards, willingness to participate in shared observability and incident management, discipline around API version management, and coordination of release windows. Partners that absorb these obligations into their operating model receive predictability and faster delivery of new product capabilities in return.

Limitations and directions for further research

Three limitations apply to the present review. The first limitation is the absence of empirical validation against closed corporate data, with the work resting on public sources and industry cases. Empirical validation across data from several banks and partners is a natural extension of the topic. The second limitation captures the temporal scope of the analysis, which fixes the landscape as of the mid-2020s. Several factors may reshape the picture over a five- to seven-year horizon, including migration toward post-quantum cryptography and possible revisions to international payment standards. The third limitation acknowledges that AI-driven tools within API ecosystems constitute an independent research frontier. Anomaly detection over API traffic, automated generation and maintenance of contract tests, and AI-assisted runbooks for incident management deserve a study of their own.

V. CONCLUSION

The study establishes that standardisation at the bank-to-retailer boundary is a structural condition for scalable payment integration. OpenAPI-based contracts, TLS 1.3, mTLS, OAuth 2.x, OpenID Connect, FAPI profiles, JWS, JWE, ISO 20022, PCI DSS, ISO/IEC 27001, and NIST-aligned API security practices form the normative substrate of a mature integration layer. Their combined use reduces fragmentation across partner connections and turns the integration boundary into a governed platform interface.

The proposed core-and-periphery model gives a coherent architectural answer to the tension between banking control and retail partner variation. Security, auditability, data contracts, versioning, observability, and release governance are part of the standardised banking core. User experience, loyalty logic, promotional flows, and internal partner mappings remain within a controlled periphery. This separation supports the reuse of artefacts, certification gates, sandbox testing, partner onboarding discipline, and predictable evolution of shared APIs.

Operational resilience emerges as the decisive test of integration maturity. SLOs, SLAs, distributed tracing, structured logging, incident runbooks, shared severity models, release freezes, disaster recovery alignment, and blast-radius containment convert technical standardisation into sustained platform reliability. The article substantiates that embedded finance and co-branded retail payment programmes require a reusable, auditable, security-centred integration architecture capable of supporting partner diversity without allowing each connection to become a bespoke risk surface.

REFERENCES

[1] Ali, M. A., & Salih, S. M. (2025). Impact of Application Programming Interfaces (APIs) Economy on Digital Economics in Saudi Arabia. Sustainability, 17(9), 4104. https://doi.org/10.3390/su17094104

[2] Chung, G. (2025). Qualitative Imbalance in Quantitative Growth: An Empirical Time Series Analysis of Korea's Open Banking Platform. Platforms, 3(2), 10. https://doi.org/10.3390/platforms3020010

[3] Daiy, A. K., Shen, K.-Y., Huang, J.-Y., & Lin, T. M.-Y. (2021). A Hybrid MCDM Model for Evaluating Open Banking Business Partners. Mathematics, 9(6), 587. https://doi.org/10.3390/math9060587

[4] Gounari, M., Stergiopoulos, G., Pipyros, K., & Gritzalis, D. (2024). Harmonizing Open Banking in the European Union: an Analysis of PSD2 Compliance and Interrelation with Cybersecurity Frameworks and Standards. International Cybersecurity Law Review, 5, 79–120. https://doi.org/10.1365/s43439-023-00108-8

[5] Hosseyni, P., Küsters, R., & Würtele, T. (2024). Formal Security Analysis of the OpenID FAPI 2.0 Family of Protocols: Accompanying a Standardization Process. ACM Transactions on Privacy and Security, 28, 1–36. https://doi.org/10.1145/3699716

[6] Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2024). Microservice API Evolution in Practice: A Study on

Strategies and Challenges. Journal of Systems and Software, 215, 112110. https://doi.org/10.1016/j.jss.2024.112110

[7] Ozili, P. K. (2022). Embedded finance: assessing the benefits, use case, challenges and interest over time. Journal of Internet and Digital Economics, 2(2), 108–123. https://doi.org/10.1108/jide-05-2022-0014

[8] Preziuso, M., Koefer, F., & Ehrenhard, M. L. (2023). Open banking and inclusive finance in the European Union: perspectives from the Dutch stakeholder ecosystem. Financial Innovation, 9, 111. https://doi.org/10.1186/s40854-023-00522-1

[9] Sukhera, J. (2022). Narrative Reviews: Flexible, Rigorous, and Practical. Journal of Graduate Medical Education, 14(4), 414–417. https://doi.org/10.4300/jgme-d-22-00480.1

[10] Tanveer, F., Iradat, F., Iqbal, W., & Ahmad, A. (2025). Towards Secure APIs: A Survey on RESTful API Vulnerability Detection. Computers, Materials & Continua, 84(3), 4223–4257. https://doi.org/10.32604/cmc.2025.067536

[11] Turnbull, D., Chugh, R., & Luck, J. (2023). Systematic-narrative hybrid literature review: A strategy for integrating a concise methodology into a manuscript. Social Sciences & Humanities Open, 7(1), 100381. https://doi.org/10.1016/j.ssaho.2022.100381

[12] Venčkauskas, A., Kukta, D., Grigaliūnas, Š., & Brūzgienė, R. (2023). Enhancing Microservices Security with Token-Based Access Control Method. Sensors, 23(6), 3363. https://doi.org/10.3390/s23063363

[13] Yu, M., Liu, H., Du, J., Lin, K., Dai, T., Fu, Y., & Yang, C. (2026). From distributed tracing to proactive SLO management: a mini-review of trace-driven performance prediction for cloud-native microservices. Frontiers in Computer Science, 8. https://doi.org/10.3389/fcomp.2026.1783945