Individual Submission
Internet-Draft
Intended status: Informational
Expires: 17 September 2026

J. Munoz
Greicodex Software

16 March 2026

FideX Application Statement 5 (AS5) Protocol
draft-greicodex-fidex-protocol-00

Abstract

This document specifies the FideX Protocol (AS5), a modern application-layer protocol for secure Business-to-Business (B2B) message exchange. FideX provides cryptographic non-repudiation, data integrity, and confidentiality using JOSE (JSON Object Signing and Encryption) over HTTPS, replacing legacy AS2 and AS4 standards with a REST-oriented approach accessible to modern web developers.

FideX adopts the "AS" naming lineage: AS2 ([RFC4130]) used S/MIME over HTTP; AS4 ([OASIS-ebMS]) used SOAP/WS-Security; AS5 (FideX) uses REST/JSON/JOSE over HTTPS. This document defines the message format, cryptographic operations, partner discovery, state management, acknowledgment receipts (J-MDN), and error handling.

Status of This Document

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document is a work in progress and represents the current thinking of the FideX Protocol Working Group at Greicodex Software.

Status of This Memo

Table of Contents

1. Introduction

1.1. Purpose

FideX (Fast Integration for Digital Enterprises eXchange) defines a secure, reliable message exchange protocol for B2B electronic data interchange. The protocol ensures:

* *Non-repudiation:* Cryptographic proof of message origin
* *Integrity:* Detection of message tampering
* *Confidentiality:* End-to-end encryption
* *Reliability:* Asynchronous acknowledgments with retry semantics

1.2. Scope

This specification defines:

* Message format and structure
* Cryptographic operations (sign-then-encrypt using JOSE)
* Partner discovery and registration
* State management and acknowledgments (J-MDN)
* Error codes and handling

This specification does NOT define:

* Business document formats (the protocol is payload-agnostic)
* Application-specific processing logic
* Implementation details or code examples
* Deployment or operational procedures

1.3. AS Naming Lineage

FideX adopts the "AS" (Application Statement) naming lineage from established B2B interchange standards:

* *AS2* ([RFC4130]) -- MIME/S/MIME over HTTP (2005)
* *AS4* ([OASIS-ebMS]) -- SOAP/WS-Security over HTTP (2013)
* *AS5* (FideX) -- REST/JSON/JOSE over HTTPS (2026)

The designation signals evolutionary continuity while marking a generational leap to modern web-native architecture. AS5 is implementable by any developer with knowledge of REST APIs and standard JOSE libraries, without requiring specialized B2B middleware.

1.4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Additional terms used in this document:

Node A FideX-compliant server capable of sending and receiving messages.

Partner A trading partner with an established trust relationship.

Message A business document wrapped in a FideX envelope.

J-MDN JSON Message Disposition Notification -- a signed receipt acknowledging receipt and processing of a FideX message.

JWKS JSON Web Key Set -- a JSON structure representing a set of public keys.

AS5 Configuration A JSON document published by a node describing its endpoints, supported algorithms, and identity.

2. Transport Layer

2.1. Required Transport

Nodes MUST support HTTP/1.1 over TLS 1.3 as defined in [RFC8446]. TLS 1.2 ([RFC5246]) MAY be supported with Perfect Forward Secrecy (ECDHE key exchange) as fallback for backward compatibility.

Transport requirements:

* Port 443 (HTTPS)

* Valid certificate from a trusted Certificate Authority (CA)

* Server Name Indication (SNI) per [RFC6066]

* Full certificate chain validation

2.2. Optional Transports

Nodes MAY additionally support:

* *HTTP/2* ([RFC9113]): For multiplexed connections and improved throughput.

* *HTTP/3* ([RFC9114]): For connection resilience using QUIC.

2.3. Request Method

All FideX message transmissions MUST use the HTTP POST method to the receiving endpoint specified in the partner's AS5 configuration (see Section 6).

2.4. Content Type

FideX message envelopes MUST use Content-Type: application/json. Registration requests MUST use Content-Type: application/jose (see Section 6.3).

2.5. Message Size Limits

Implementations MUST accept messages up to 10 megabytes (complete HTTP request body). Implementations MAY accept larger messages by prior bilateral agreement. Senders MUST NOT assume support above 10 MB without prior explicit agreement. Implementations SHOULD reject oversized messages with HTTP 413 and the error code PAYLOAD_TOO_LARGE (see Section 8.3).

3. Message Structure

3.1. Message Envelope

A FideX message envelope consists of exactly two top-level JSON fields:

```
{
  "routing_header": { ... },
  "encrypted_payload": "eyJhbGc..."
}
```

The routing_header is cleartext JSON containing message metadata readable by intermediate infrastructure without decryption. The encrypted_payload is a JWE compact serialization that wraps the signed business document. These two fields are the ONLY top-level fields in a FideX message envelope.

3.2. Routing Header

The routing header is a cleartext JSON object conveying message routing metadata. It MUST contain the following fields:

fidex_version (string, REQUIRED) Protocol version in semantic "major.minor" format, e.g., "1.0".

message_id (string, REQUIRED) Globally unique message identifier. UUID v4 ([RFC9562]) with an "fdx-" prefix is RECOMMENDED (e.g., "fdx-550e8400-e29b-41d4-a716-446655440000").

sender_id (string, REQUIRED) URN of the sending organization. See Section 3.2.1 for valid formats.

receiver_id (string, REQUIRED) URN of the receiving organization. See Section 3.2.1 for valid formats.

document_type (string, REQUIRED) Business document type identifier. MUST match the pattern ^[A-Z0-9_]+$. See Section 3.3.

timestamp (string, REQUIRED) ISO 8601 UTC timestamp with millisecond precision. Format: YYYY-MM-DDTHH:mm:ss.SSSZ. The UTC designator "Z" MUST be used; no local timezone offsets are permitted.

receipt_webhook (string, OPTIONAL) HTTPS URL where the J-MDN receipt SHOULD be delivered. HTTP (non-TLS) is NOT allowed. When omitted, the receiver MUST deliver the J-MDN to the sender's receive_receipt endpoint from the sender's AS5 configuration (see Section 7.3.5).

payload_digest (string, OPTIONAL) SHA-256 digest of the encrypted_payload field value. Computed as "sha256:" followed by the lowercase hexadecimal encoding of SHA-256 applied to the UTF-8 byte representation of the encrypted_payload string as it appears in the JSON envelope. Enables routing-layer integrity checks without decryption.

Extension fields MAY be included and MUST use the prefix "x-" to avoid collision with future standard fields. Implementations MUST ignore unknown extension fields without error.

3.2.1. Identifier Format

The sender_id and receiver_id fields MUST use URN format. The following URN namespaces are recognized:

* urn:gln:{gln} -- GS1 Global Location Number

* urn:duns:{duns} -- D-U-N-S Number

* urn:lei:{lei} -- Legal Entity Identifier (ISO 17442)

* urn:tin:{tin} -- Tax Identification Number

* urn:custom:{identifier} -- Organization-defined identifier

3.3. Document Type Registry

The document_type field uses a two-tier naming system to accommodate both standardized inter-organization document types and organization-specific custom types.

3.3.1. Standard Document Types

The following standard document types are defined by the FideX Working Group. Standard types use uppercase alphanumeric characters and underscores, matching the pattern ^[A-Z0-9_]+$. GS1 document types are defined in [GS1-JSON].

Type Identifier	Standard	Description
GS1_ORDER_JSON	GS1	Purchase order (JSON binding)
GS1_INVOICE_JSON	GS1	Commercial invoice (JSON binding)
GS1_DESADV_JSON	GS1	Despatch advice (JSON binding)
GS1_RECADV_JSON	GS1	Receiving advice (JSON binding)
GS1_CATALOG_JSON	GS1	Product catalog (JSON binding)
X12_850	ANSI X12	Purchase order
X12_810	ANSI X12	Invoice
X12_856	ANSI X12	Advance ship notice
EDIFACT_ORDERS	UN/EDIFACT	Purchase order message
EDIFACT_INVOIC	UN/EDIFACT	Invoice message
EDIFACT_DESADV	UN/EDIFACT	Despatch advice message
UBL_ORDER_21	OASIS UBL 2.1	Order document
UBL_INVOICE_21	OASIS UBL 2.1	Invoice document

Table 1: Standard FideX Document Types

3.3.2. Custom Document Types

Organizations MAY define custom document types using reverse domain notation to avoid collision with standard types and with other organizations' custom types.

Custom types MUST follow the pattern: {TLD}_{ORG}_{DOCTYPE}_{VERSION}, for example: COM_ACME_WAREHOUSE_RECEIPT_V2.

Custom types MUST NOT use the reserved prefixes: GS1_, X12_, EDIFACT_, or UBL_.

3.3.3. Receiver Behavior for Document Types

Receiver nodes:

* MUST accept messages with any syntactically valid document_type.
* SHOULD return a J-MDN with error code UNKNOWN_DOCUMENT_TYPE for types they cannot process.
* MUST NOT reject messages at the HTTP level solely because of an unknown document_type. The J-MDN mechanism MUST be used instead.

3.4. Encrypted Payload

The encrypted_payload field contains a JWE (JSON Web Encryption) token in compact serialization as defined in [RFC7516]. The JWE encrypts a JWS (JSON Web Signature) token as defined in [RFC7515], creating a nested "sign-then-encrypt" structure:

JWE(JWS(business_document))

This pattern ensures both authenticity (the signature is inside the encryption envelope, protecting it from substitution) and confidentiality (the business content is encrypted end-to-end).

4. Cryptographic Requirements

4.1. Signature (JWS)

Messages MUST be signed using the sender’s private key before encryption.

The REQUIRED signing algorithm is *RS256* (RSASSA-PKCS1-v1_5 with SHA-256) as defined in [RFC7518].

The following signing algorithms are OPTIONAL:

* RS384, RS512 -- stronger RSA variants
* PS256, PS384, PS512 -- RSA-PSS variants
* ES256 (ECDSA with P-256 and SHA-256) -- RECOMMENDED for new deployments
* ES384 (ECDSA with P-384 and SHA-384)

Minimum key sizes: 2048 bits for RSA (4096 bits RECOMMENDED); P-256 minimum for ECDSA (P-384 RECOMMENDED).

The JWS Protected Header MUST include:

```
{
  "alg": "RS256",
  "kid": "{sender-key-id}"
}
```

4.2. Encryption (JWE)

Signed messages MUST be encrypted using the receiver’s public key.

The REQUIRED algorithms are:

* *Key Encryption:* RSA-OAEP as defined in [RFC7518]

* *Content Encryption:* A256GCM (AES-256-GCM)

The JWE Protected Header MUST include:

```
{
  "alg": "RSA-OAEP",
  "enc": "A256GCM",
  "cty": "JWT",
  "kid": "{receiver-key-id}"
}
```

The cty (content type) header parameter MUST be set to "JWT" to indicate the JWE payload contains a nested JWS/JWT token, per [RFC7516] Section 4.1.12. This is required for correct processing of nested tokens.

4.3. Payload Encoding

The JWS payload MUST be the UTF-8 encoding of the JSON-serialized business document. Binary payloads (images, PDFs, etc.) MUST be base64-encoded before JWS signing.

4.4. Sender Identity Verification

Receivers MUST verify that the JWS signing key identified by kid in the JWS header belongs to the partner identified by sender_id in the routing header. Specifically:

* The kid MUST resolve to a public key in the sender’s JWKS.

* The sender’s JWKS MUST be fetched from the public_domain associated with the sender_id in the local partner registry.

This prevents cross-partner key substitution attacks where an attacker uses a valid key from one partner to impersonate another.

4.5. Prohibited Algorithms

Implementations MUST NOT use:

* The none algorithm (unsigned tokens)
* Symmetric signature algorithms (HS256, HS384, HS512)
* RSA keys smaller than 2048 bits
* Deprecated algorithms: MD5, SHA-1-based signatures

5. Key Distribution

5.1. JSON Web Key Set (JWKS) Endpoint

Nodes MUST publish their public keys via a JWKS endpoint at the well-known URI:

https://{public_domain}/.well-known/jwks.json

This endpoint:

* MUST be publicly accessible without authentication.
* MUST return Content-Type: application/json.
* SHOULD include Cache-Control headers (1 hour RECOMMENDED).

5.2. JWKS Format

Per [RFC7517], keys MUST include at minimum the following fields. An RSA signing key example:

```
{
  "keys": [
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "acme-sign-rsa-2026-01",
      "alg": "RS256",
      "n": "{base64url-encoded-modulus}",
      "e": "AQAB"
    }
  ]
}
```

An EC signing key example:

```
{
  "keys": [
    {
      "kty": "EC",
      "use": "sig",
      "kid": "acme-sign-ec-2026-01",
      "alg": "ES256",
      "crv": "P-256",
      "x": "{base64url-x-coordinate}",
      "y": "{base64url-y-coordinate}"
    }
  ]
}
```

Field semantics:

kty Key type: "RSA" or "EC".

use "sig" for signing keys; "enc" for encryption keys.

kid Unique key identifier used in JWS/JWE headers.

alg Algorithm for this key (RS256, ES256, RSA-OAEP, etc.).

5.3. Key Rotation

Implementations SHOULD rotate keys annually. During rotation, the following procedure MUST be followed to ensure uninterrupted partner communication:

1. Publish the new key alongside the old key in the JWKS.

2. Maintain both keys for a transition period (30-60 days RECOMMENDED).

3. Begin signing new outbound messages with the new key after partners have had time to cache it.

4. Remove the old key from the JWKS after the transition period.

The kid value SHOULD encode the purpose and date to aid key management, e.g., "org-sign-rsa-2026-01".

6. Partner Discovery

6.1. Discovery Overview

Partner discovery enables automated trading partner onboarding without manual certificate exchange. The process consists of four phases:

1. Configuration discovery

2. Key retrieval

3. Signed registration

4. Mutual confirmation

The AS5 configuration URL MAY be distributed via QR code, email, or a trading partner portal. The protocol is designed so that scanning a QR code initiates fully automated mutual trust establishment.

6.2. AS5 Configuration Endpoint

Nodes MUST expose an AS5 configuration document at an HTTPS URL. This URL:

* MAY be at any path (no required well-known location).

* MAY include a single-use security token as a query parameter.

* SHOULD be shareable via QR code.

The AS5 configuration document structure:

```
{
  "fidex_version": "1.0",
  "supported_versions": ["1.0"],
  "conformance_profile": "core",
  "node_id": "urn:gln:1234567890123",
  "organization_name": "Example Corp",
  "public_domain": "fidex.example.com",
  "supported_document_types": ["GS1_ORDER_JSON", "GS1_INVOICE_JSON"],
  "endpoints": {
    "receive_message": "https://fidex.example.com/api/v1/receive",
    "receive_receipt": "https://fidex.example.com/api/v1/receipt",
    "register": "https://fidex.example.com/api/v1/register",
    "jwks": "https://fidex.example.com/.well-known/jwks.json"
  },
  "security": {
    "signature_algorithm": "RS256",
    "encryption_algorithm": "RSA-OAEP",
    "content_encryption": "A256GCM",
    "minimum_key_size": 2048
  }
}
```

Required fields in the AS5 configuration:

fidex_version Current active protocol version (string, REQUIRED).

supported_versions Array of all protocol versions this node supports (array, REQUIRED).

node_id URN identifier for this node (string, REQUIRED).

organization_name Human-readable organization name (string, REQUIRED).

public_domain Public-facing domain name hosting JWKS and endpoints (string, REQUIRED).

endpoints Object containing service endpoint URLs (object, REQUIRED). MUST include: receive_message, receive_receipt, register, jwks.

security Cryptographic algorithm declarations (object, REQUIRED).

conformance_profile "core", "enhanced", or "edge" (string, OPTIONAL).

supported_document_types Array of document_type identifiers this node processes (array, OPTIONAL).

6.2.1. Version Negotiation

When two nodes with different supported versions communicate, the sender MUST:

1. Retrieve the receiver’s supported_versions from their AS5 configuration.
2. Select the highest version present in BOTH supported_versions arrays.
3. Set fidex_version in the routing header to the negotiated version.

The receiver MUST reject messages with a fidex_version not present in its supported_versions.

If no common version exists, the sender MUST NOT transmit and SHOULD report an incompatibility error to the local system.

6.3. Discovery Handshake

The four-phase discovery handshake proceeds as follows:

Phase 1 -- Initiator Discovers Responder:

1. Initiator obtains the responder’s AS5 configuration URL.
2. Initiator fetches the responder’s AS5 configuration document.
3. Initiator fetches the responder’s JWKS from the well-known endpoint.

Phase 2 -- Initiator Registers:

The initiator constructs and submits a signed registration payload:

```
{
  "fidex_version": "1.0",
  "initiator_node_id": "urn:gln:...",
  "initiator_as5_config_url": "https://...",
  "security_token": "...",
  "timestamp": "2026-03-09T19:00:00.000Z"
}
```

1. Initiator constructs the registration payload as shown above.

2. Initiator signs the payload with its private key (JWS compact serialization).

3. Initiator POSTs the JWS to the responder’s register endpoint with Content-Type: application/jose.

Phase 3 -- Responder Validates:

1. Responder validates the security token (if configured).

2. Responder fetches the initiator’s AS5 configuration from the URL in the request.

3. Responder fetches the initiator’s JWKS and verifies the JWS signature.

4. Responder stores the initiator’s details and returns HTTP 200.

Phase 4 -- Completion:

1. Initiator receives HTTP 200 confirmation.

2. Initiator stores the responder’s details.

3. Both parties MAY immediately exchange FideX messages.

6.4. Registration Security Requirements

Registration requests MUST:

* Be signed with the initiator’s private key (RS256 minimum).

* Be transmitted with Content-Type: application/jose.

* Include a timestamp within +/-15 minutes of current time.

* Include the security token if the responder has configured one.

Security tokens MUST contain at least 128 bits of entropy, generated using a cryptographically secure pseudo-random number generator (CSPRNG). Examples: UUID v4, 32 hexadecimal characters, or 22 base64url characters.

Responders MUST:

* Validate the JWS signature before trusting any payload content.

* Reject registrations with timestamps outside +/-15 minutes.

* Reject invalid or previously used security tokens.

6.5. Partner De-Registration

De-registration is a local, unilateral operation. There is no protocol-level de-registration handshake.

Partner states:

State	Description
ACTIVE	Normal operation -- messages accepted
SUSPENDED	Temporarily paused -- messages rejected with HTTP 503
INACTIVE	De-registered -- messages rejected with HTTP 401

Table 2: Partner States

De-registration procedure:

1. The initiating party sets partner status to INACTIVE locally.

2. The initiating party SHOULD notify the partner via out-of-band channel.

3. The initiating party MUST continue accepting J-MDNs for in-flight messages.

4. After a 24-hour grace period, new inbound messages MUST be rejected with HTTP 401.

5. Partner records SHOULD be retained for audit purposes (7 years RECOMMENDED).

7. Message States and Receipts

7.1. Message State Machine

Messages transition through the following states:

State	Description
QUEUED	Created locally, awaiting transmission
SENT	Transmitted to receiver, awaiting J-MDN receipt
DELIVERED	J-MDN received and cryptographically verified
FAILED	Permanent failure or maximum retries exceeded

Table 3: Message States

7.2. Synchronous HTTP Response

Upon receiving a message, nodes MUST perform structural validation and return:

* *HTTP 202 Accepted:* Message is structurally valid and queued for processing.

* *HTTP 4xx/5xx:* Immediate rejection (see Section 8).

HTTP 202 indicates structural acceptance only. It does NOT indicate successful decryption, business document processing, or J-MDN generation.

Idempotency: Receivers MUST handle duplicate message_id values idempotently. If a receiver encounters a message_id it has already accepted, it MUST return HTTP 202 again and MUST NOT process the message a second time. This ensures safe sender retries when HTTP responses are lost.

7.3. Asynchronous Receipt: J-MDN

The J-MDN (JSON Message Disposition Notification) is the legally most significant artifact in the FideX protocol. It provides cryptographic proof that a specific message was received, decrypted, and accepted or rejected by the trading partner.

After processing a message, the receiver MUST send a J-MDN to the sender. The delivery target is determined as follows:

1. If receipt_webhook is present in the routing header, deliver to that URL.

2. Otherwise, deliver to the sender’s receive_receipt endpoint from the sender’s AS5 configuration.

3. If neither is available, the J-MDN MUST be stored and the failure logged; J-MDNs MUST NOT be discarded.

7.3.1. J-MDN Payload Schema

The J-MDN JSON object contains the following fields:

original_message_id (string, REQUIRED) The message_id from the original message’s routing_header.

status (string, REQUIRED) Either "DELIVERED" or "FAILED".

receiver_id (string, REQUIRED) URN of the receiver generating this J-MDN. MUST match the receiver_id in the original routing_header.

hash_verification (string, REQUIRED) SHA-256 hash of the raw business payload bytes before JWS signing. Format: "sha256:{64 hex characters}". See Section 7.3.2.

timestamp (string, REQUIRED) ISO 8601 UTC timestamp when the J-MDN was created. Format: YYYY-MM-DDTHH:mm:ss.SSSZ.

error_log (object|null, REQUIRED) MUST be null when status is "DELIVERED". MUST be an error object when status is "FAILED".

signature (string, REQUIRED) JWS compact serialization covering all other J-MDN fields. See Section 7.3.3.

Positive J-MDN example (DELIVERED):

```
{
  "original_message_id": "fdx-a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "status": "DELIVERED",
  "receiver_id": "urn:gln:9876543210987",
  "hash_verification": "sha256:9f86d08...b0f00a08",
  "timestamp": "2026-03-09T19:30:02.000Z",
  "error_log": null,
  "signature": "eyJhbGciOiJSUzI1NiIs..."
}
```

Negative J-MDN example (FAILED):

```
{
  "original_message_id": "fdx-a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "status": "FAILED",
  "receiver_id": "urn:gln:9876543210987",
  "hash_verification": "sha256:0000...0000",
  "timestamp": "2026-03-09T19:30:02.000Z",
  "error_log": {
    "error_code": "DECRYPTION_FAILED",
    "error_message": "Unable to decrypt payload",
    "details": "Key ID mismatch"
  },
  "signature": "eyJhbGciOiJSUzI1NiIs..."
}
```

7.3.2. Hash Verification

The hash_verification field proves that the receiver decrypted exactly the payload the sender signed.

Definition:

```
hash_verification =
    "sha256:" || hex( SHA-256( raw_business_payload_bytes ) )
```

Where raw_business_payload_bytes is the UTF-8 encoded bytes of the business JSON payload BEFORE JWS signing (i.e., the exact cleartext that was the input to the JWS sign operation).

When status is "FAILED" and the receiver could NOT decrypt the payload, hash_verification MUST be set to a string of 64 zero hex digits: "sha256:" followed by 64 ASCII "0" characters.

7.3.3. J-MDN Signature Requirements

The signature field MUST contain a JWS compact serialization covering all other J-MDN fields (i.e., all fields except signature itself).

JWS Protected Header for J-MDN signing:

```
{
  "alg": "RS256",
  "kid": "{receiver-signing-key-id}"
}
```

Signing procedure:

1. Construct a JSON object with all J-MDN fields EXCEPT signature.

2. Serialize to canonical JSON: no extra whitespace; keys in alphabetical order (error_log, hash_verification, original_message_id, receiver_id, status, timestamp).

3. Sign using RS256 with the receiver’s private signing key.

4. Set signature to the resulting JWS compact serialization.

Verification procedure (sender side):

1. Extract signature from the received J-MDN.

2. Parse JWS; extract kid from the JWS header.

3. Look up the receiver’s public key from their JWKS using kid.

4. Verify the JWS signature.

5. Compare the JWS payload with the remaining J-MDN fields for consistency.

7.3.4. J-MDN Error Codes

When status is "FAILED", the error_log object MUST contain:

error_code (string, REQUIRED) Machine-readable code from the table below.

error_message (string, REQUIRED) Human-readable description.

details (string, OPTIONAL) Additional diagnostic info. MUST NOT contain sensitive data.

Code	Description
DECRYPTION_FAILED	Cannot decrypt JWE (wrong key or corrupted ciphertext)
SIGNATURE_INVALID	JWS signature verification failed
UNKNOWN_DOCUMENT_TYPE	The document_type is not supported by the receiver
PAYLOAD_TOO_LARGE	Message exceeds receiver’s processing limits
INTERNAL_ERROR	Receiver encountered an internal processing error

Table 4: Standard J-MDN Error Codes

7.3.5. J-MDN Delivery Protocol

The J-MDN is delivered via HTTP POST:

```
POST {receipt_webhook} HTTP/1.1
Host: {sender_host}
Content-Type: application/json
X-FideX-Original-Message-ID: {original_message_id}

{J-MDN JSON body}
```

Expected response:

```
{
  "receipt_acknowledged": true
}
```

Timing requirements:

* Receivers SHOULD send J-MDN within 5 minutes of receiving the original message.
* If processing takes longer, the J-MDN MUST still be sent when processing completes.
* There is no strict upper time limit; delivery is asynchronous by design.

7.3.6. J-MDN Delivery Retry Schedule

If the J-MDN delivery endpoint is unreachable, the receiver SHOULD retry according to the following schedule:

* Attempt 1: Immediate
* Attempt 2: +1 minute
* Attempt 3: +5 minutes
* Attempt 4: +15 minutes
* Attempt 5: +1 hour

After 5 failed attempts, the receiver SHOULD log the failure and store the J-MDN for manual retrieval. The J-MDN MUST NOT be discarded.

7.4. Sender Retry Semantics

Senders SHOULD retry failed transmissions with exponential backoff:

* Attempt 1: Immediate
* Attempt 2: +1 minute
* Attempt 3: +5 minutes
* Attempt 4: +15 minutes
* Attempt 5: +30 minutes
* Attempt 6: +1 hour

After 5-6 failed attempts, the message SHOULD transition to FAILED state requiring manual operator intervention.

8. Error Handling

8.1. HTTP Status Codes

Code	Meaning	Sender Action
202	Accepted for processing	Wait for J-MDN
400	Invalid message structure	Do not retry (permanent

		error)
401	Authentication failed or partner INACTIVE	Do not retry (permanent error)
413	Payload too large	Do not retry (permanent error)
429	Rate limit exceeded	Retry with backoff; honor Retry-After header
500	Server internal error	Retry with backoff
503	Service unavailable or partner SUSPENDED	Retry with backoff

Table 5: HTTP Status Code Semantics

Implementations SHOULD include a Retry-After header on HTTP 429 responses, per [RFC9110] Section 10.2.3.

8.2. Error Response Format

All error responses MUST use the following JSON structure:

```
{
  "error": {
    "code": "INVALID_ROUTING_HEADER",
    "message": "Missing required field: message_id",
    "timestamp": "2026-03-09T19:00:00.000Z"
  }
}
```

8.3. Standard Error Codes

Message transmission errors:

* INVALID_ROUTING_HEADER -- Missing or malformed routing header

* UNKNOWN_RECEIVER -- receiver_id not recognized in partner registry

* UNKNOWN_DOCUMENT_TYPE -- document_type not supported

* PAYLOAD_TOO_LARGE -- Message exceeds the 10 MB size limit

Cryptographic errors:

* DECRYPTION_FAILED -- Cannot decrypt JWE

* SIGNATURE_INVALID -- JWS signature verification failed

* UNKNOWN_KEY_ID -- Key ID not found in JWKS

Discovery errors:

* INVALID_TOKEN -- Security token invalid or expired

* DUPLICATE_REGISTRATION -- Partner already registered

* CONFIG_UNREACHABLE -- Cannot fetch AS5 configuration

9. Security Considerations

9.1. Threat Model

FideX addresses the following threats:

Man-in-the-Middle (MitM) Mitigated by TLS 1.3 transport encryption combined with JWE application-layer encryption. An attacker who intercepts the TLS session cannot read the encrypted payload.

Message Tampering Mitigated by JWS digital signatures. Any modification to the signed payload invalidates the signature, detected during verification.

Replay Attacks Mitigated by unique message IDs combined with timestamp validation. Receivers MUST maintain a replay cache and reject duplicate message_id values.

Repudiation Mitigated by cryptographic signatures on both messages (JWS) and receipts (J-MDN signature), creating a legally auditable chain of custody.

Key Compromise Mitigated by key rotation procedures and JWKS-based distribution. Compromised keys can be revoked by removing them from the JWKS.

Cross-Partner Key Substitution Mitigated by sender identity verification (Section 4.4): the receiver verifies the signing key belongs to the claimed sender_id.

9.2. Implementation Security Requirements

Implementations MUST:

* Validate TLS certificates against a trusted CA store on all outbound connections.
* Use constant-time comparison for signature verification to prevent timing attacks.
* Maintain a cache of recently seen message IDs to detect replay attacks.
* Reject messages with timestamps outside a +/-15-minute window of current time.
* Maintain the message ID replay cache for at least 24 hours.
* Synchronize system clocks using NTP ([RFC5905]) or equivalent.
* Generate cryptographically secure random message IDs using a CSPRNG.
* Never expose private keys in logs, error messages, or API responses.

Implementations SHOULD:

* Implement per-partner rate limiting on all endpoints.
* Use separate key pairs for signing and encryption (different use values).
* Rotate keys annually following the procedure in Section 5.3.
* Monitor for anomalous patterns (repeated authentication failures, invalid signatures).

9.3. Key Management

Private keys:

* MUST be generated using a cryptographically secure random number generator.
* MUST be stored encrypted at rest.
* MUST NOT be transmitted over any network.
* SHOULD be stored in a Hardware Security Module (HSM) for high-security deployments.

Public keys:

* MUST be distributed exclusively via the JWKS endpoint.

* SHOULD include a kid value encoding purpose and date (e.g., "org-sign-rsa-2026-01").

* MAY be cached by consuming parties for up to 24 hours.

9.4. Compliance Considerations

FideX is designed to support compliance with:

* Non-repudiation requirements for legally binding electronic transactions.

* FDA 21 CFR Part 11 (electronic signatures in regulated industries).

* GDPR and similar data protection frameworks (via encrypted payloads).

* GS1 and UN/CEFACT EDI standards (via payload-agnostic design).

Recommended audit trail retention periods:

* Message metadata: 7 years

* Cryptographic signatures (JWS): 7 years

* J-MDN receipts: 7 years

9.5. Deployment Note

FideX is designed to operate on standard web infrastructure. A conforming node can be implemented as a standard HTTPS web application behind any reverse proxy (Nginx, Apache, Caddy, cloud load balancers). No specialized B2B middleware, message broker, or gateway software is required. This is a key differentiator from AS2 (which required dedicated gateway software) and AS4 (which required SOAP stacks and ebMS processing engines).

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

11.2. Informative References

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC4130] Moberg, D. and R. Drummond, "MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)", RFC 4130, DOI 10.17487/RFC4130, July 2005, <https://www.rfc-editor.org/info/rfc4130>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[OASIS-ebMS] OASIS, "OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features", October 2007, <https://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/os/>.

[GS1-JSON] GS1, "GS1 Web Vocabulary", 2023, <https://www.gs1.org/voc/>.

Appendix A: Complete Message Example

The following illustrates a complete FideX message exchange.

HTTP request (sender to receiver):

```
POST /api/v1/receive HTTP/1.1
Host: partner.example.com
Content-Type: application/json

{
  "routing_header": {
    "fidex_version": "1.0",
    "message_id": "fdx-a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "sender_id": "urn:gln:1234567890123",
    "receiver_id": "urn:gln:9876543210987",
    "document_type": "GS1_ORDER_JSON",
    "timestamp": "2026-03-09T19:30:00.000Z",
    "receipt_webhook": "https://sender.example.com/api/v1/receipt"
  },
  "encrypted_payload": "eyJhbGciOiJSU0EtT0FFUCIs..."
}
```

HTTP 202 response (receiver accepts):

```
{
  "status": "accepted",
  "message_id": "fdx-a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "timestamp": "2026-03-09T19:30:00.500Z"
}
```

J-MDN receipt (receiver delivers to sender’s webhook):

```
POST /api/v1/receipt HTTP/1.1
Host: sender.example.com
Content-Type: application/json
X-FideX-Original-Message-ID: fdx-a1b2c3d4-e5f6-7890-abcd-ef1234567890

{
  "original_message_id": "fdx-a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "status": "DELIVERED",
  "receiver_id": "urn:gln:9876543210987",
  "hash_verification": "sha256:9f86d08...b0f00a08",
  "timestamp": "2026-03-09T19:30:02.000Z",
  "error_log": null,
  "signature": "eyJhbGciOiJSUzI1NiIs..."
}
```

Appendix B: Glossary

AS5 Application Statement 5 -- the formal designation of the FideX protocol.

B2B Business-to-Business electronic commerce.

CSPRNG Cryptographically Secure Pseudo-Random Number Generator.

EDI Electronic Data Interchange.

GLN Global Location Number -- GS1 standard organization identifier.

HSM Hardware Security Module -- tamper-resistant hardware for key storage.

J-MDN JSON Message Disposition Notification -- the FideX signed delivery receipt.

JOSE JSON Object Signing and Encryption -- the IETF framework comprising JWS, JWE, JWK, and JWA.

JWE JSON Web Encryption (RFC 7516).

JWK JSON Web Key (RFC 7517).

JWKS JSON Web Key Set -- a JSON document containing a set of public keys.

JWS JSON Web Signature (RFC 7515).

Node A FideX-compliant server capable of sending and receiving messages.

URN Uniform Resource Name -- a persistent, location-independent identifier.

Appendix C: Conformance Profiles

FideX defines three conformance profiles to enable progressive adoption. Implementations MUST declare which profile(s) they conform to via the conformance_profile field in their AS5 configuration.

C.1 FideX Core

An implementation claiming *FideX Core* conformance MUST support all requirements listed in Sections 2 through 8 of this specification, including:

* HTTP/1.1 over TLS 1.3

* Sign-then-encrypt: JWE(JWS(payload))

* RS256 signatures; RSA-OAEP + A256GCM encryption

* RSA key size >= 2048 bits
* JWKS endpoint at /.well-known/jwks.json
* AS5 configuration endpoint
* 4-phase discovery handshake
* Message state machine (QUEUED, SENT, DELIVERED, FAILED)
* J-MDN generation, signing, and delivery
* J-MDN fallback delivery via AS5 config receive_receipt endpoint
* Standard error codes and HTTP status codes
* Replay detection via message_id cache (>= 24 hours)
* Timestamp validation (+/-15-minute window)

C.2 FideX Enhanced

An implementation claiming *FideX Enhanced* conformance MUST satisfy FideX Core AND additionally support:

* HTTP/2 for multiplexed connections
* Separate signing and encryption key pairs (different kid and use values)
* RSA key size >= 4096 bits
* Key rotation support (overlapping key publication)
* J-MDN delivery retry per the schedule in Section 7.3.6
* Per-partner rate limiting on all endpoints
* Structured JSON logging with security event correlation IDs
* Health endpoints (/health and /ready)

C.3 FideX Edge

An implementation claiming *FideX Edge* conformance MUST satisfy FideX Enhanced AND additionally support:

* HTTP/3 over QUIC for connection resilience

* Mutual TLS (mTLS) -- client certificate authentication

* HSM-based private key storage

Appendix D: Interoperability Test Vectors

This appendix provides known-answer test vectors to allow implementers to verify their JOSE cryptographic operations produce correct output.

WARNING: The key material in this appendix is public and MUST NOT be used for any production traffic.

Test payload (UTF-8 bytes):

```
{"order_id":"PO-TEST-001","amount":100.00,"currency":"USD"}
```

SHA-256 hash of test payload:

```
sha256:
  bf21a9e8fbc5a3846fb05b4fa0859e0917b2202f9a69e4c98b7b0f09cb281e71
```

Expected JWS header for signing:

```
{"alg":"RS256","kid":"test-sign-2026-01"}
```

Expected JWE header for encryption:

```
{
  "alg": "RSA-OAEP",
  "enc": "A256GCM",
  "cty": "JWT",
  "kid": "test-enc-2026-01"
}
```

Test routing header:

```
{
  "fidex_version": "1.0",
  "message_id": "fdx-00000000-0000-0000-0000-000000000001",
  "sender_id": "urn:gln:0000000000001",
  "receiver_id": "urn:gln:0000000000002",
  "document_type": "GS1_ORDER_JSON",
  "timestamp": "2026-01-01T00:00:00.000Z",
  "receipt_webhook": "https://test.sender.example.com/receipt"
}
```

Verification procedure:

1. *Sign Test:* Sign the test payload using RS256. Verify using the test public key. Verified payload MUST match original.

2. *Encrypt Test:* Encrypt the JWS using RSA-OAEP/A256GCM. Decrypt using the test private key. Decrypted content MUST match the JWS.

3. *Hash Test:* Compute SHA-256 of the raw payload bytes. Result MUST equal bf21a9e8fbc5a3846fb05b4fa0859e0917b2202f9a69e4c98b7b0f09cb281e71.

4. *Round-Trip Test:* Construct a complete envelope, parse it, decrypt, verify signature, and extract payload. Result MUST match original test payload.

5. *J-MDN Test:* Construct a J-MDN for the test message. Sign with receiver’s key. Verify the J-MDN signature.

Appendix E: JSON Schema Definitions

The following JSON Schema (Draft-07) definitions provide machine-readable validation rules for FideX structures.

E.1 Routing Header Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://fidex-protocol.org/schemas/v1/routing-header.json",
  "title": "FideX Routing Header",
  "type": "object",
  "required": [
    "fidex_version", "message_id", "sender_id",
    "receiver_id", "document_type", "timestamp"
  ],
  "additionalProperties": true,
  "properties": {
    "fidex_version": {
      "type": "string", "pattern": "^\\d+\\.\\d+$"
    },
    "message_id": {
      "type": "string", "minLength": 1, "maxLength": 256
    },
    "sender_id": {
      "type": "string",
      "pattern": "^urn:(gln|duns|lei|tin|custom):.+$"
    },
    "receiver_id": {
      "type": "string",
      "pattern": "^urn:(gln|duns|lei|tin|custom):.+$"
    },
    "document_type": {
      "type": "string", "pattern": "^[A-Z0-9_]+$",
      "minLength": 1, "maxLength": 128
    },
    "timestamp": { "type": "string", "format": "date-time" },
    "receipt_webhook": {
      "type": "string", "format": "uri",
      "pattern": "^https://"
    },
    "payload_digest": {
      "type": "string",
      "pattern": "^sha256:[a-f0-9]{64}$"
    }
  },
  "patternProperties": { "^x-": {} }
}
```

E.2 J-MDN Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://fidex-protocol.org/schemas/v1/jmdn.json",
  "title": "FideX J-MDN",
  "type": "object",
  "required": [
    "original_message_id", "status", "receiver_id",
    "hash_verification", "timestamp", "error_log", "signature"
  ],
  "additionalProperties": false,
  "properties": {
    "original_message_id": { "type": "string", "minLength": 1 },
    "status": {
      "type": "string", "enum": ["DELIVERED", "FAILED"]
    },
    "receiver_id": {
      "type": "string",
      "pattern": "^urn:(gln|duns|lei|tin|custom):.+$"
    },
    "hash_verification": {
      "type": "string",
      "pattern": "^sha256:[a-f0-9]{64}$"
    },
    "timestamp": { "type": "string", "format": "date-time" },
    "error_log": {
      "oneOf": [
        { "type": "null" },
        {
          "type": "object",
          "required": ["error_code", "error_message"],
          "properties": {
            "error_code": {
              "type": "string",
              "enum": [
                "DECRYPTION_FAILED",
                "SIGNATURE_INVALID",
                "UNKNOWN_DOCUMENT_TYPE",
                "PAYLOAD_TOO_LARGE",
                "INTERNAL_ERROR"
              ]
            },
            "error_message": { "type": "string" },
            "details": { "type": "string" }
          }
        }
      ]
    },
    "signature": { "type": "string", "minLength": 1 }
```

```
    }
  }
```

E.3 AS5 Configuration Schema

```
  {
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "https://fidex-protocol.org/schemas/v1/as5-config.json",
    "title": "FideX AS5 Configuration",
    "type": "object",
    "required": [
      "fidex_version", "supported_versions", "node_id",
      "organization_name", "public_domain",
      "endpoints", "security"
    ],
    "properties": {
      "fidex_version": {
        "type": "string", "pattern": "^\\d+\\.\\d+$"
      },
      "supported_versions": {
        "type": "array",
        "items": { "type": "string" },
        "minItems": 1
      },
      "conformance_profile": {
        "type": "string",
        "enum": ["core", "enhanced", "edge"]
      },
      "node_id": { "type": "string", "pattern": "^urn:" },
      "organization_name": { "type": "string", "minLength": 1 },
      "public_domain": { "type": "string" },
      "supported_document_types": {
        "type": "array",
        "items": {
          "type": "string", "pattern": "^[A-Z0-9_]+$"
        }
      },
      "endpoints": {
        "type": "object",
        "required": [
          "receive_message", "receive_receipt",
          "register", "jwks"
        ],
        "properties": {
          "receive_message": { "type": "string", "format": "uri" },
          "receive_receipt": { "type": "string", "format": "uri" },
          "register": { "type": "string", "format": "uri" },
          "jwks": { "type": "string", "format": "uri" }
```

```
        }
      },
      "security": {
        "type": "object",
        "required": [
          "signature_algorithm", "encryption_algorithm",
          "content_encryption", "minimum_key_size"
        ],
        "properties": {
          "signature_algorithm": { "type": "string" },
          "encryption_algorithm": { "type": "string" },
          "content_encryption": { "type": "string" },
          "minimum_key_size": {
            "type": "integer", "minimum": 2048
          }
        }
      }
    }
  }
```

Author’s Address

Javier Munoz
Greicodex Software
Email: javier@greicodex.com
URI: https://greicodex.com