Integrating with the INTA E-Invoicing System

A Step-by-Step Developer's Guide to Connecting, Authenticating, and Submitting Electronic Invoices in Compliance with Iranian Tax Law.

The Developer’s Journey to Compliance

This guide breaks down the INTA integration into four distinct, sequential parts, taking you from initial setup to a successfully verified submission.

1. The Foundation: Setup & Core Concepts
2. The Gateway: Authentication
3. The Core Task: Creating & Submitting Invoices
4. The Confirmation: Verification & Management

Understanding the Core Concepts & Terminology

Digital Signature (امضای دیجیتال)

A process that uses a private key to sign a message, allowing the recipient to verify the sender's identity and message integrity using the corresponding public key.

Digital Signature Certificate (گواهی امضا)

An electronic certificate issued by a trusted authority, containing the public key, expiration date, and identity information of the owner (in .crt or .cer format).

Tax Memory ID (شناسه یکتای حافظه مالیاتی)

A unique identifier assigned by the INTA through the taxpayer portal (Karpoosheh). This ID is required for issuing all electronic invoices.

JSON Web Token (JWT)

A standard for creating access tokens. In this system, JWTs are used to create a signed token (JWS) for authentication.

JSON Web Signature (JWS)

The standard used to sign data (like authentication requests and invoices) to ensure data integrity.

JSON Web Encryption (JWE)

The standard used to encrypt the signed invoice data to ensure confidentiality.

Prerequisites & Initial Setup Checklist

1 **Obtain a Digital Signature Certificate:** Acquire a valid electronic signature certificate for the legal entity from a trusted Iranian Certificate Authority.

2 **Register on the Taxpayer Portal (Karpoosheh):** The taxpayer must register and become a member of the official INTA portal.

3 **Create a Tax Memory Profile:** Within Karpoosheh, create a profile for the 'Tax Memory' from which invoices will be issued.

4 **Upload Your Public Key:** Upload your 2048-bit public key or signature certificate (`.crt`) to the Tax Memory profile in Karpoosheh.

5 **Receive Your Unique Tax Memory ID** (`شناسه یکتای حافظه مالیاتی`)**:** Once the profile is set up, the INTA assigns the unique Tax Memory ID. This ID is your primary **`cliientld`** for all API interactions.

**Note*: Per document RC_TICS.IS_v1.6, the taxpayer must select their information submission method (e.g., 'By taxpayer') and upload their public key certificate to Karpoosheh to receive their Tax Memory ID.

The Authentication Flow: Generating Your Access Token

Every API request (except the first) must be authenticated with a single-use JWS token. This token is generated through a five-step challenge-response process.

1 **GET Nonce**
Request a unique, temporary challenge string from the server.

↓

2 **Build Payload**
Create a JSON object containing the `nonce` and your `clientId` (Tax Memory ID).

↓

3 **Build Header**
Create a JSON header containing the signing algorithm (**RS256**), your public key certificate (**x5c**), and the signature timestamp (**sigT**).

↓

4 **Sign Packet**
Use your private key to sign the Base64Url-encoded Header and Payload, creating the final JWS token.

↓

5 **Use Token**
Place the JWS token in the `Authorization: Bearer [token]` header of your next API request.

Step 1: The Handshake – Getting a Nonce

A `Nonce` is a random, single-use challenge string with a limited time-to-live. It prevents replay attacks and ensures each request is unique.

API Request Example

- Method: `GET`
- Endpoint: `https://tp.tax.gov.ir/requestsmanager/api/v2/nonce`
- Parameter: `timeToLive` (Optional, integer between 10-200 seconds, default 30)

```
curl -X 'GET'
'https://tp.tax.gov.ir/requestsmanager/
api/v2/nonce?timeToLive=20'
-H 'accept: */*'
```

API Response Example

- Content-Type: `application/json`

```
{
  "nonce": "ab202a55-e106-445c-b2a3-
5a7364991a66",
  "expDate": "2023-08-
22T16:07:18.277824208Z"
}
```

Steps 2 & 3: Constructing the JWS Header and Payload

JWS Protected Header

```
{
  "alg": "RS256",
  "x5c": ["MIIDe..."],
  "sigT": "2023-05-13T10:44:47Z",
  "crit": ["sigT"]
}
```

Algorithm. Must be `RS256`.

Certificate. An array containing the Base64-encoded X.509 certificate.

Signature Timestamp. The UTC time of signing in `yyyy-MM-dd'T'HH:mm:ss'Z'` format.

Critical. Indicates that `sigT` is a critical header parameter that must be understood by the server.

JWS Payload

```
{
  "nonce": "ab202a55-...",
  "clientId": "A11226"
}
```

Nonce. The exact string received from the `/nonce` endpoint.

Client ID. Your unique Tax Memory ID.

Step 4: Signing and Generating the Final JWS Token

The **Header** and **Payload** are each Base64Url-encoded, joined by a period, and then signed with your private key using the **RSASSA-PKCS1-v1_5 using SHA-256** algorithm to create the final JWS token.

```
BASE64URL(Header) + . + BASE64URL(Payload) + . + BASE64URL(Signature)
```

Java Code Snippet

```java
// Loading Private Key and Certificate
PrivateKey privateKey = ...;
X509Certificate certificate = ...;

// Generate Signature Time
String signatureTime = LocalDateTime.now(ZoneOffset.UTC)
    .format(DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss'Z'"));

// Set Payload
String payload = "{\"nonce\":\"...\",\"clientId\":\"A11226\"}";

// Generate JWS
JsonWebSignature jws = new JsonWebSignature();
jws.setPayload(payload);
jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);
jws.setKey(privateKey);
jws.setCertificateChainHeaderValue(certificate);
jws.setHeader("sigT", signatureTime);
jws.setHeader("crit", new String[]{"sigT"});

// Sign and serialize
String jwt = jws.getCompactSerialization();
```

.NET Code Snippet

```csharp
// Loading Private Key and Certificate
var privateKey = ...; // from PemReader
var certificate = ...; // from PemReader
var publicKey = ...; // from certificate

var payload = "{\"nonce\":\"...\",\"clientId\":\\"A11226\"}";

// Generate JWS
var jws = JwtBuilder.Create()
    .WithAlgorithm(new RS256Algorithm(publicKey, privateKey))
    .AddHeader(HeaderName.X5c, new[]
        {Convert.ToBase64String(certificate.GetRawCertData()}})
    .AddHeader("sigT", DateTime.UtcNow.ToString("yyyy-MM-
dd'T'HH:mm:ss'Z'"))
    .AddHeader("crit", new[] {"sigT"})
    .Encode(JsonSerializer.Deserialize<JsonNode>(payload));
```

Key Dependencies: `jose4j` (Java), `jose-jwt`, `JWT`, `Portable.BouncyCastle` (.NET)

The Core Task: Invoice Submission Workflow

Once authenticated, submitting an invoice is a four-step process of structuring the data, signing it for integrity, encrypting it for confidentiality, and sending it to the INTA.

1. Structure Invoice Data

Create the complete invoice as a JSON object according to the INTA specification.

2. Sign the Invoice (JWS)

The entire invoice JSON becomes the payload of a JWS packet, signed with your private key.

3. Encrypt the Packet (JWE)

The signed JWS packet is encrypted using a symmetric key, which is itself encrypted with the INTA's public key.

4. POST to API

Send the final, encrypted JWE packet to the `/invoice` endpoint.

INTA

Step 1: Structuring the Invoice Data

The invoice is a detailed JSON object. While the full specification contains over 80 fields, they can be understood through three main logical sections. Always use the official Unit of Measurement codes.

```
{
  "header": {
    "taxid": "A11216...",         // Unique Tax ID for the invoice
    "indatim": 1683997837988,     // Invoice creation timestamp (Unix ms)
    "tins": "14003778990",        // Seller's National ID / Economic Code
    // ... other header fields
  },
  "body": [
    {
      "sstid": "2710000138624", // Goods/Service ID
      "sstt": "فوالد صنعت قطعات سرسیلندر", // Goods/Service Description
      "mu": "164",              // Unit of Measurement Code (e.g., 164 = Kilogram)
      "am": 2,                  // Quantity
      "fee": 10000,             // Unit Price
      // ... other line item fields
    }
  ],
  "payments": [
    // ... payment details if applicable
  ]
}
```

Reference document `RC_UMGS.ST_V1.18` for the complete list of official Unit of Measurement (mu) codes.

MU Code	Description
164	Kilogram (کیلوگرم)
166	Meter (متر)
179	Piece (عدد)
180	Liter (لیتر)
...	...

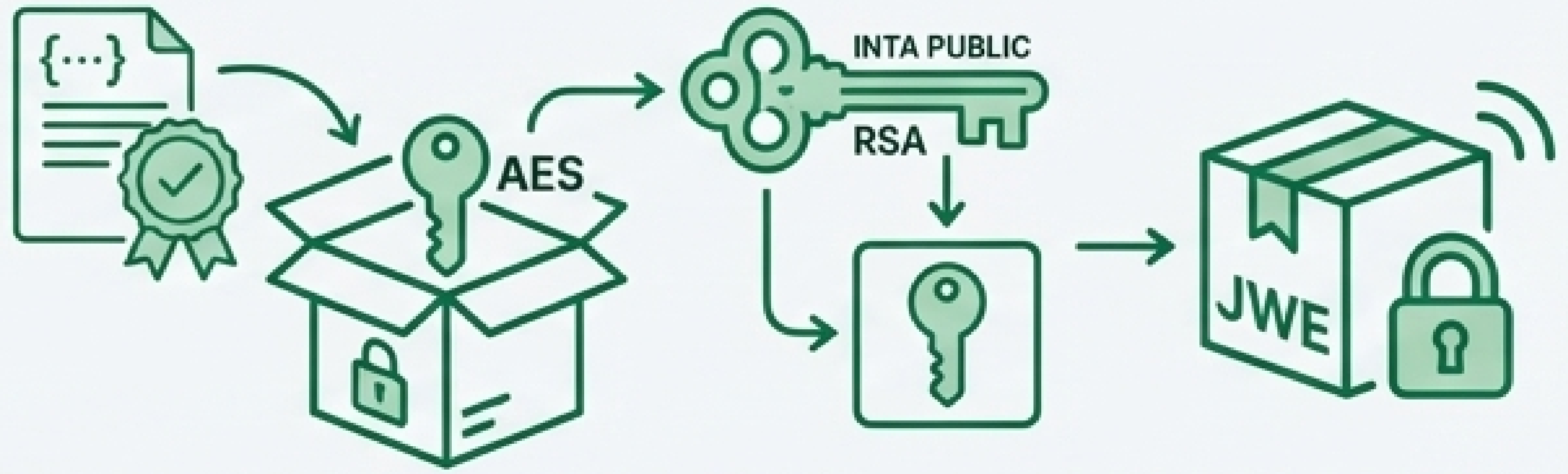
Steps 2 & 3: Signing for Integrity (JWS) and Encrypting for Confidentiality (JWE)

Part 1: Signing for Integrity (JWS)

The entire invoice JSON from the previous step is used as the payload for a JWS packet. This process is identical identcal to the authentication token signing, using your private key.

⚠ **Purpose:** Guarantees to the INTA that the invoice data has not been altered since it was created by the authenticated sender.

Part 2: Encrypting for Confidentiality (JWE)



1. Fetch the INTA's public encryption key and its ID (`` `kid` ``) from the `` `GET /server-information` `` endpoint.
2. Generate a random, local symmetric key (AES-256-GCM).
3. Encrypt the *entire JWS packet* using this symmetric key.
4. Encrypt the *symmetric key* itself using the INTA's public key (RSA-OAEP-256).
5. Assemble the final JWE packet containing the encrypted key, initialization vector (IV), encrypted data (ciphertext), and the INTA's key ID (`` `kid` ``).

⚠ **Purpose:** Ensures that the invoice content is confidential and can only be decrypted by the INTA server.

INTA

Step 4: Sending the Invoice and Capturing the Response

The final encrypted JWE string is sent as the `payload` in a `POST` request. The response will contain unique identifiers for tracking.

API Request Example

Method: `POST`
Endpoint: `https://tp.tax.gov.ir/requestsmanager/api/v2/invoice`
Headers: `Authorization: Bearer [JWS_Auth_Token]`,
`Content-Type: application/json`

```
[
  {
    "payload": "eyJhbGciOiJSU0EtT0FFUC0yNTYi...[JWE]...",
    "header": {
      "requestTraceId": "cf019c26-f235-11ed-a05b-0242ac120003",
      "fiscalId": "A11216"
    }
  }
]
```

API Response Example (on success)

```
{
  "timestamp": 1684054900556,
  "result": [
    {
      "uid": "cf019c26-f235-11ed-a05b-0242ac120003",
      "packetType": null,
      "referenceNumber": "3645b684-2c1e-400c-8584-f739c09d99fb",
      "data": null
    }
  ]
}
```

****Important**:** Immediately store the `uid` and `referenceNumber`. You will need them to query the invoice status.

INTA

The Confirmation: How to Verify Invoice Status

After submission, an invoice enters a processing queue. You must query the API to confirm its final status (Success or **Failure**). The system provides three methods for inquiry.

Endpoint	Key Parameter(s)	Typical Use Case
`GET /inquiry-by-reference-id`	`referenceIds`	The most common method. Check the status of one or more specific invoices immediately after submission using the returned `referenceNumber`.
`GET /inquiry-by-uid`	`uidList, fiscalId`	Useful for checking the status using your own internal request ID (`requestTraceId` becomes `uid`) that you generated before sending.
`GET /inquiry`	`start, end, pageNumber, pageSize`	Best for batch reconciliation, retrieving all submissions within a specific date range to check for any missed or failed invoices.

INTA

Decoding the Status Response: Success vs. Failure

FAILED Response Example

```
{
  "referenceNumber": "93367b02...",
  "uid": "2b982bfd-...",
  "status": "FAILED",
  "data": {
    "error": [
      {
        "code": "012802",
        "message": "The value entered in the 'Settlement
Method' field is not among the allowed values.",
        "errorType": "ERROR"
      }
    ], ...
  },
  "fiscalId": "A111OK",
  "sign": ""
}
```

The `data` object contains an `error` array with specific codes and human-readable messages detailing what went wrong.

SUCCESS Response Example

```
{
  "referenceNumber": "f9173085...",
  "uid": "c5352f85-...",
  "status": "SUCCESS",
  "data": {
    "error": [],
    "warning": [],
    "success": true
  },
  "fiscalId": "A111OK",
  "sign": "eyJhbGciOiJSU...[JWS]"
}
```

The `sign` field contains a JWS packet signed by the INTA. You can verify this signature with the INTA's public key to confirm the authenticity of the success status.

NotebookLM

INTA

Further Management & Official Resources

Utility Endpoints

Beyond invoice submission and status checks, the API provides endpoints for managing taxpayer and fiscal device information.

- `GET /taxpayer?economicCode={code}`
  Retrieves public information about a taxpayer profile, including their `taxpayerStatus` (e.g., `ACTIVE`).
- `GET /fiscal-information?memoryId={id}`
  Retrieves details about a specific Tax Memory device, including its `fiscalStatus`.
- `POST /invoice/payment`
  Allows for sending payment data related to invoices with settlement methods of credit or installments.

Official Resources

For a complete and exhaustive list of all fields, validation rules, and error codes, always refer to the latest official INTA documentation:

Technical Connection Guide
`RC_TICS.IS_v1.6`

Electronic Invoice Issuance Guide
`RC_IITP.IS_V7.6`

Goods/Service Unit of Measurement Codes
`RC_UMGS.ST_V1.18`