Token Authentication

Header Format

```
Authorization: Token <api_token>
```

Description All API requests to the RcertVault Digital Signature API require Token-based authentication. The token must be passed in the `Authorization` header with the prefix `"Token"` (including the trailing space).

Request Headers

Authorization (required)

```
Authorization: Token <api_token>
```

⚠ **Important:** The header value must start with the literal string `Token ` followed by your API key. Do not use Bearer, Basic, or other authentication schemes.

Example Request

```
curl -X GET \
  http://202.51.70.18/api/v1/signing-challenge/ \
  -H "Authorization: Token 1234567890abcdef"
```

Error Responses

403 Forbidden Returned when the Authorization header is missing, malformed, or contains an invalid token.

```
{
  "detail": "Authentication credentials were not provided."
}
```

Common Causes:

- Missing `Authorization` header entirely
- Header value does not start with `Token ` prefix
- Expired or revoked API token
- Token does not have sufficient permissions for the requested resource

Notes

- Tokens are tied to specific user accounts and inherit their permissions
- All endpoints except the API documentation (`/redoc/`) require this header
- Tokens should be kept secure and never committed to version control
- If you receive 403 errors, verify the exact format: `Token` + space + `<your-api-key>`

Signing Process:

Signing Challenge API

Endpoint

```
POST /api/v1/signing-challenge/
```

Description: Initiates a digital signing session by either returning an existing active session or generating a challenge that must be signed externally by the user.

This API is the entry point for all signing operations (PDF or text).

Request Format

Content-Type

```
application/json
```

Request Headers

Authorization (required)

```
Authorization: Token <api_token>
```

⚠ If this header is missing or invalid, the API returns 403 Forbidden.

Request Body

force_create (optional)

- **Type:** boolean
- **Description:** Controls whether to reuse an existing active signing session or force creation of a new one.
- **Behavior:**
  - `true` → deletes existing session (if any) and creates a new one
  - `false` → returns existing active session (if available)
- **Default:** false

Example Request (curl)

```
curl -X POST \
  http://202.51.70.18/api/v1/signing-challenge/ \
  -H "Accept: application/json" \
  -H "Authorization: Token <api_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "force_create": true
  }'
```

Response Scenarios

Case 1: Existing Active Signing Session

Returned when auto-signing is enabled and an active session already exists.

```
{
  "signing_session_id": "8b88a1ad-5778-40fc-b216-14296ea2b907",
  "max_pdfs": 10,
  "expires_in_sec": 50884,
  "message": "Existing active signing session returned"
}
```

Meaning:

- The session can be used immediately for signing
- No challenge verification is required

Case 2: Challenge Generated (Manual Signing Required)

Returned when auto-signing is not enabled.

```
{
  "challenge_id": "57e48e56-efce-42d6-adc8-ead7860c2530",
  "challenge_text":
"eJwNyzkOgCAQAMAvLSCuFJSG2JAoWhuEtcQD6Hi89jNuMnayZhnnbXRrq5lezVu6UidoRQAw
AT3p/JEILFHVAf4jtSAR3vpqZQLxf2O5x6umopm0ErW7Jf/VCg+/Q4e9Q==",
  "expires_in_sec": 300
}
```

Meaning:

- `challenge_text` must be signed using DsignerClient or EMSigner
- The signed output must be submitted to the Verify Challenge API
- Challenge expires after 300 seconds

Error Responses

403 Forbidden

Returned when authentication credentials are missing or invalid.

```
{
  "detail": "Authentication credentials were not provided."
}
```

Common Causes:

- Missing Authorization header
- Invalid or expired API token

Processing Logic

1. Authenticate API token
2. Check for existing active signing session
3. Apply force_create logic
4. Either:
   a. Return active signing session, or
   b. Generate a signing challenge
5. Return session or challenge response

Notes

- Signing sessions are valid for 24 hours
- Only one active session may exist per user (unless forced)
- This API must be called before signing PDFs or text
- Challenge expires in 300 seconds (5 minutes)

Verify Challenge API

Endpoint

```
POST /api/v1/verify-challenge
```

Description: Verifies a cryptographic signature generated by DsignerClient or EMSigner against a previously issued challenge. Upon successful verification, establishes a signing session that can be used for bulk signing operations.

This endpoint is required only when auto-signing is disabled, and a challenge was returned by the Signing Challenge API.

Request Format

Content-Type

```
application/json
```

Request Headers

Authorization (required)

```
Authorization: Token <api_token>
```

Request Body

challenge_id (required)

- **Type:** string (UUID)
- **Description:** The unique identifier of the challenge issued by the Signing Challenge API
- **Example:** `"57e48e56-efce-42d6-adc8-ead7860c2530"`

signature (required)

- **Type:** string
- **Description:** The cryptographic signature generated by signing the `challenge_text` using DsignerClient or EMSigner
- **Format:** Base64-encoded string or raw signature string (depending on client configuration)

force_create (optional)

- **Type:** boolean
- **Description:** If `true`, creates a new signing session and deletes any existing session
- **Default:** false
- **Note:** This ensures a fresh session is created even if one already exists

Example Request (curl)

```
curl -X POST \
  http://202.51.70.18/api/v1/verify-challenge \
  -H "Accept: application/json" \
  -H "Authorization: Token <api_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "challenge_id": "57e48e56-efce-42d6-adc8-ead7860c2530",
    "signature": "<signature-from-dsigner-client>",
    "force_create": true
  }'
```

Response Scenarios

Case 1: Successful Verification

Returned when the signature is valid and matches the challenge.

```
{
  "signing_session_id": "8b88a1ad-5778-40fc-b216-14296ea2b907",
  "max_pdfs": 10,
  "expires_in_sec": 86400,
  "message": "Signing session established successfully"
}
```

Meaning:

- Verification successful
- `signing_session_id` can now be used with the Bulk Sign API
- Session is valid for 24 hours (86400 seconds)

Case 2: Invalid Signature

Returned when the signature does not match the challenge or is malformed.

```
{
  "status": "error",
  "message": "Invalid signature or challenge expired",
  "code": "VERIFICATION_FAILED"
}
```

Meaning:

- The signature provided does not verify against the challenge
- Challenge may have expired (300-second limit)
- Must restart the process by calling Signing Challenge API again

Error Responses

400 Bad Request

Returned when required fields are missing or malformed.

```
{
  "challenge_id": ["This field is required."],
  "signature": ["This field is required."]
}
```

403 Forbidden

Authentication failure.

```
{
  "detail": "Authentication credentials were not provided."
}
```

404 Not Found

Challenge ID does not exist or has expired.

```
{
  "detail": "Challenge not found or expired."
}
```

Processing Logic

1. Authenticate API token
2. Validate request payload (challenge_id and signature presence)
3. Retrieve challenge by challenge_id
4. Verify challenge has not expired (300-second window)
5. Verify signature against challenge_text using cryptographic validation
6. If valid:
    a. Create signing session (or reuse existing if force_create is false)
    b. Delete challenge to prevent replay attacks
    c. Return signing_session_id
7. If invalid, return error response

Notes

- Challenge can only be verified once; subsequent attempts will fail with 404
- The signing session returned is identical to those returned when auto-signing is enabled
- If `force_create: true`, any existing signing session for the user is immediately invalidated
- The signature must be generated by signing the exact `challenge_text` bytes returned by the Signing Challenge API

Bulk Sign API

Endpoint

```
POST /api/v1/bulk-sign
```

Description: Signs multiple PDF documents using an active signing session. This endpoint performs the actual digital signature operation on the provided PDF files, applying cryptographic signatures and optional visual signature appearances.

Requires a valid `signing_session_id` obtained either from the Signing Challenge API (auto-signing enabled) or Verify Challenge API (manual signing).

Request Format

Content-Type

```
application/json
```

Request Headers

Authorization (required)

```
Authorization: Token <api_token>
```

Request Body

signing_session_id (required)

- **Type:** string (UUID)
- **Description:** Active signing session identifier
- **Source:** Returned by Signing Challenge API or Verify Challenge API

pdfs (required)

- **Type:** array of objects
- **Description:** List of PDF documents to sign
- **Structure:**
  - `pdf_id` (string): Unique identifier for the document (for tracking)
  - `content` (string): Base64-encoded PDF content
  - `signature_coordinates` (object): Positioning for visual signature
    - page (integer): Page number (1-indexed)
    - x1 (float): Bottom-left X coordinate
    - y1 (float): Bottom-left Y coordinate
    - x2 (float): Top-right X coordinate
    - y2 (float): Top-right Y coordinate

Example Request (curl)

```
curl -X POST \
  http://202.51.70.18/api/v1/bulk-sign \
  -H "Accept: application/json" \
  -H "Authorization: Token <api_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "signing_session_id": "8b88a1ad-5778-40fc-b216-14296ea2b907",
    "pdfs": [
      {
        "pdf_id": "doc-001",
        "content": "<base64-encoded-pdf-content>",
        "signature_coordinates": {
          "page": 1,
          "x1": 400,
          "y1": 100,
          "x2": 600,
          "y2": 200
        }
      }
    ]
  }'
```

Response Scenarios

Case 1: Successful Signing

All PDFs signed successfully.

```
{
  "status": "success",
  "signed_pdfs": [
    {
      "pdf_id": "doc-001",
      "signed_content": "<base64-encoded-signed-pdf>",
      "signature_id": "sig-uuid-123",
      "signed_at": "2026-02-02T10:30:00Z"
    }
  ],
  "session_remaining_seconds": 86000
}
```

Meaning:

- PDFs have been digitally signed
- `signed_content` contains the Base64-encoded signed PDF
- Session remains valid for the indicated duration

Case 2: Partial Failure

Some PDFs failed to sign.

```
{
  "status": "partial_success",
  "signed_pdfs": [...],
  "failed_pdfs": [
    {
      "pdf_id": "doc-002",
      "error": "Invalid coordinates",
      "error_code": "COORDINATES_OUT_OF_BOUNDS"
    }
  ]
}
```

Case 3: Session Expired

Signing session has expired.

```
{
  "status": "error",
  "message": "Signing session expired or invalid",
  "code": "SESSION_EXPIRED"
}
```

Error Responses

400 Bad Request

Invalid request format or coordinates.

```
{
  "pdfs": ["This field is required."],
  "signing_session_id": ["Invalid UUID format."]
}
```

403 Forbidden

Authentication failure or session does not belong to the user.

```
{
  "detail": "Authentication credentials were not provided."
}
```

404 Not Found

Signing sessions are not found.

```
{
  "detail": "Signing session not found."
}
```

429 Too Many Requests

Exceeded maximum PDFs per session.

```
{
  "detail": "Maximum number of PDFs (10) exceeded for this session."
}
```

Processing Logic

1. Authenticate API token
2. Validate signing_session_id exists and belongs to user
3. Check session has not expired (24-hour limit)
4. Validate PDF array structure and coordinates
5. For each PDF:
    a. Decode Base64 content
    b. Validate PDF format
    c. Apply digital signature using session credentials
    d. Add visual signature appearance at specified coordinates (if provided)
    e. Encode signed PDF as Base64
6. Update session expiration or usage counters
7. Return results array

Notes

- Maximum 10 PDFs per session (as indicated by `max_pdfs` in session response)
- Coordinates use PDF points (1/72 inch) with origin at bottom-left
- Visual signature appearance is optional but recommended for legal validity
- Session is valid for 24 hours from creation but may expire earlier if `max_pdfs` limit is reached
- Once a PDF is signed, the signature is permanent and cryptographically bound to the document

Verify Form/Text API

Endpoint

```
POST /api/v1/verify/verify_text/
```

Description: Verifies digital signatures on form data or text content. Validates the cryptographic integrity of signatures, checks document hashes, and verifies certificate validity against trusted Certificate Authorities.

This endpoint is used for form signing verification and text signature validation (non-PDF content).

Request Format

Content-Type

```
application/json
```

Request Headers

Authorization (required)

```
Authorization: Token <api_token>
```

Request Body

Parameters depend on the type of content being verified (form fields or text). Refer to the API documentation at `/redoc/` for specific parameter schemas based on your use case.

General Parameters:

content (required)

- **Type:** string/object
- **Description:** The signed content or form data to verify
- **Format:** Depends on signature type (JSON for forms, string for text)

signature (required)

- **Type:** string
- **Description:** Base64-encoded digital signature

certificate (optional)

- **Type:** string
- **Description:** PEM-encoded certificate or certificate identifier
- **Note:** If not provided, system will attempt to extract from signature

Example Request (curl)

```
curl -X POST \
  http://202.51.70.18/api/v1/verify/verify_text/ \
  -H "Accept: application/json" \
  -H "Authorization: Token <api_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "content": "<text-or-form-data>",
    "signature": "<base64-signature>",
    "certificate": "<optional-cert>"
  }'
```

Response Scenarios

Case 1: Successful Verification

All checks passed.

```
{
  "status": "success",
  "message": [
    [
      {
        "cert_name": "Test1",
        "signature_ok": true,
        "hash_ok": true,
        "cert_ok": true
      }
    ],
    true
  ]
}
```

Meaning:

- `signature_ok: true` → Signature was created with the certificate's private key and content hasn't been altered
- `hash_ok: true` → Document hash matches the encrypted hash at time of signing (integrity verified)
- `cert_ok: true` → Certificate is valid, issued by trusted CA, not expired or revoked
- Final `true` → Overall verification status passed (all checks successful)

Case 2: Verification Failed

One or more checks failed.

```
{
  "status": "success",
  "message": [
    [
      {
        "cert_name": "Test1",
        "signature_ok": false,
        "hash_ok": true,
        "cert_ok": true
      }
    ],
    false
  ]
}
```

Meaning:

- Final boolean `false` indicates overall verification failure
- In this example, signature verification failed (content may have been altered after signing) but certificate and hash are valid
- Check individual boolean fields to identify specific failure

Case 3: Certificate Issues

Certificate validation failed.

```
{
  "status": "success",
  "message": [
    [
      {
        "cert_name": "Test1",
        "signature_ok": true,
        "hash_ok": true,
        "cert_ok": false
      }
    ],
    false
  ]
}
```

Meaning:

- Signature and hash are valid
- Certificate is invalid (expired, revoked, or untrusted CA)
- Overall status is false due to certificate failure

Error Responses

400 Bad Request

Missing required fields.

```
{
  "content": ["This field is required."],
  "signature": ["This field is required."]
}
```

403 Forbidden

Authentication failure.

```
{"detail": "Authentication credentials were not provided."}
```

422 Unprocessable Entity

Invalid signature format or corrupt data.

```
{
  "detail": "Unable to parse signature."
}
```

Processing Logic

1. Authenticate API token
2. Parse content and signature from request
3. Extract certificate from signature or use provided certificate
4. **Signature Verification:** Decrypt signature hash using public key, compare with computed content hash
5. **Hash Verification:** Compute current content hash, compare with signed hash
6. **Certificate Verification:**
    a. Validate certificate chain against trusted CAs
    b. Check expiration dates
    c. Check revocation lists (CRL/OCSP)
7. Aggregate results into response array
8. Return final boolean indicating overall validity (true only if all checks pass)

Notes

- Response always returns HTTP 200 for successfully processed requests, even if verification fails (check the final boolean)
- The nested array structure allows for multiple signature verification in future versions
- `cert_name` indicates the Common Name (CN) of the signing certificate
- A `false` result in any individual check (`signature_ok`, `hash_ok`, `cert_ok`) results in the final boolean being `false`

- Hash verification ensures content integrity (document hasn't changed since signing)
- Certificate verification ensures the signer's identity is trusted and valid

Verify PDF API

Endpoint

```
POST /api/v1/verify/verify_text/
```

Description: Verifies digital signatures embedded within PDF documents. Extracts signature fields from the PDF, validates cryptographic integrity, and checks certificate validity.

Note: This endpoint shares the same URL as the Form Verification API but accepts PDF content instead of text/form data. The response format is identical.

Request Format

Content-Type

```
multipart/form-data
```

or

```
application/json
```

Request Headers

Authorization (required)

```
Authorization: Token <api_token>
```

Request Body

pdf (required for multipart)

- **Type:** file
- **Description:** PDF file containing digital signatures to verify

OR

pdf_content (required for JSON)

- **Type:** string
- **Description:** Base64-encoded PDF content

Example Request (curl - Multipart)

```
curl -X POST \
  http://202.51.70.18/api/v1/verify/verify_text/ \
  -H "Authorization: Token <api_token>" \
  -F "pdf=@/path/to/signed-document.pdf"
```

Example Request (curl - JSON)

```
curl -X POST \
  http://202.51.70.18/api/v1/verify/verify_text/ \
  -H "Accept: application/json" \
  -H "Authorization: Token <api_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "pdf_content": "<base64-encoded-pdf>"
  }'
```

Response Scenarios

Case 1: Valid PDF Signature

All signatures in the PDF are valid.

```
{
  "status": "success",
  "message": [
    [
      {
        "cert_name": "Signer Name",
        "signature_ok": true,
        "hash_ok": true,
        "cert_ok": true
      }
    ],
    true
  ]
}
```

Case 2: Tampered PDF

PDF has been modified after signing.

```
{
  "status": "success",
  "message": [
    [
      {
        "cert_name": "Signer Name",
        "signature_ok": false,
        "hash_ok": false,
        "cert_ok": true
      }
    ],
    false
  ]
}
```

Meaning:

- Document has been altered (hash mismatch)
- Certificate is valid but document integrity is compromised

Case 3: No Signatures Found

PDF does not contain digital signatures.

```
{
  "status": "error",
  "message": "No digital signatures found in PDF",
  "code": "NO_SIGNATURES"
}
```

Error Responses

400 Bad Request

Invalid PDF format.

```
{
  "pdf": ["Invalid PDF file format."]
}
```

403 Forbidden

Authentication failure.

```
{
  "detail": "Authentication credentials were not provided."
}
```

Processing Logic

1. Authenticate API token
2. Parse PDF file from request (multipart) or decode Base64 (JSON)
3. Extract all digital signature fields from PDF
4. For each signature:
    a. Extract signature dictionary and certificate
    b. Verify signature using public key from certificate
    c. Compute document hash (excluding signature bytes) and compare with signed hash
    d. Validate certificate chain, expiration, and revocation status
5. Compile results array with individual signature checks
6. Determine overall status (true only if all signatures valid)
7. Return response

Notes

- Supports multiple signatures per PDF (returns array of results)
- Response format is the same as Form Verification API
- Hash calculation excludes the signature bytes themselves to allow verification
- Validates PDF structure integrity as well as cryptographic signatures
- If PDF is encrypted, it must be decrypted before verification
- Certificate trust chain is validated against system trust store

Generate Public Certificate (WebSocket)

Endpoint

`ws://202.51.70.18/ws/dsigner/`

Description:
Establishes a WebSocket connection to generate `.p7b` format public certificate files for upload to the RcertVault Signing Portal. This certificate file contains the public key components necessary for signature verification and is used to configure signing identities in the portal.

Connection Headers

Authorization (required)

```
Authorization: Token <api_token>
```

Protocol Standard WebSocket protocol with JSON message framing.

Message Format

Request Message

```
{
  "action": "generatePublicKey",
  "output": "C:/Users/username/Downloads/dev.p7b"
}
```

Parameters:

action (required)

- **Type:** string
- **Value:** `"generatePublicKey"`
- **Description:** Specifies the operation to perform

output (required)

- **Type:** string
- **Description:** Absolute file system path where the `.p7b` file should be saved
- **Format:** OS-specific path (Windows or Unix-style)
- **Permissions:** The application must have write permissions to the specified directory

Example Client Implementation (JavaScript)

```
const ws = new WebSocket('ws://202.51.70.18/ws/dsigner/');

ws.onopen = function() {
  console.log('Connected to certificate generator');
  ws.send(JSON.stringify({
    action: "generatePublicKey",
    output: "C:/Users/username/Downloads/dev.p7b"
  }));
};

ws.onmessage = function(event) {
  const response = JSON.parse(event.data);
  if (response.status === "success") {
    console.log("Certificate generated:", response.fileType);
    console.log("Content:", response.message); // Base64 encoded
  } else {
    console.error("Error:", response.message);
  }
};

ws.onerror = function(error) {
  console.error('WebSocket Error:', error);
};
```

Response Scenarios

Case 1: Success

Certificate generated successfully.

```
{
  "status": "success",
  "message": "MIILxAYJKoZIhvcNAQcCoIILtTCC...",
  "fileType": ".p7b"
}
```

Meaning:

- Certificate generated in PKCS#7 (.p7b) format
- `message` contains Base64-encoded certificate data
- File has been saved to the specified output path on the server/local system
- Can be decoded and saved client-side or accessed at the specified path

Case 2: Path Error

Invalid output path specified.

```
{
  "status": "error",
  "message": "Invalid output path or permission denied",
  "code": "INVALID_PATH"
}
```

Case 3: Authentication Error

Token invalid or expired during handshake.

Connection closed with code 1008 (Policy Violation) or 1002 (Protocol Error).

Alternative: REST API Approach

If WebSocket is unavailable, decode the Base64 response:

```
import base64

# Response from WebSocket
response = {
    "status": "success",
    "message": "MIILxAYJKoZIhvcNAQc...",
    "fileType": ".p7b"
}

# Save locally
with open("certificate.p7b", "wb") as f:
    f.write(base64.b64decode(response["message"]))
```

Error Responses

Connection Refused

WebSocket endpoint unavailable or authentication failed during handshake.

Timeout

No response within expected timeframe (30 seconds recommended timeout).

```
{
  "status": "error",
  "message": "Operation timed out",
  "code": "TIMEOUT"
}
```

Processing Logic

1. Establish WebSocket connection
2. Authenticate token during handshake
3. Wait for client message with action and output path
4. Validate output path format and permissions
5. Generate public certificate from signing credentials:
    a. Extract public key components
    b. Format as PKCS#7 (.p7b) structure
    c. Encode as Base64
6. Save file to specified output location
7. Return success response with file content and type

8. Close connection or keep open for additional operations

Notes

- The `.p7b` file contains the public certificate (PKCS#7 format) needed for RcertVault upload
- The `output` path is used by the local Dsigner/EMSigner client to save the file
- Alternative: Decode the `message` field from the response and persist as `.p7b` file manually
- This certificate is used to configure your signing identity in the RcertVault Signing Portal
- WebSocket connection is stateful but short-lived; close after receiving response
- File paths must be absolute and accessible to the client application
- The generated certificate corresponds to the private key stored in your hardware token or secure storage

Coordinate System Reference

PDF Coordinate System

Standard PDF coordinate system used for signature placement in Bulk Sign API.

Origin (0,0)

- **Location:** Bottom-left corner of the page
- **Axis Orientation:** X increases right, Y increases up
- **Unit:** Points (1 point = 1/72 inch)

Signature Box Definition

```
Top of PDF
    ↑ Y
    |
    |    (x1,y2) ------- (x2,y2)
    |       |               |
    |       |   SIGNATURE   |
    |       |      BOX      |
    |       |               |
    |    (x1,y1) ------- (x2,y1)
    |
    +--------------------------→ X
Bottom of PDF
```

Parameters

x1 (required)

- **Type:** float
- **Description:** X-coordinate of bottom-left corner
- **Example:** `400`

y1 (required)

- **Type:** float
- **Description:** Y-coordinate of bottom-left corner
- **Example:** `100`

x2 (required)

- **Type:** float
- **Description:** X-coordinate of top-right corner
- **Must be:** Greater than x1
- **Example:** `600`

y2 (required)

- **Type:** float
- **Description:** Y-coordinate of top-right corner
- **Must be:** Greater than y1
- **Example:** `200`

Calculation

```
Width = x2 - x1 (points)
Height = y2 - y1 (points)
Position from left = x1 (points)
Position from bottom = y1 (points)
```

Example

```
Box: (400, 100, 600, 200)
- Width: 200 points (~2.78 inches)
- Height: 100 points (~1.39 inches)
- Position: 400 points from left, 100 points from bottom
```

Common Positions (A4 Page: 595×842 points)

Bottom Right

```
{
  "page": 1,
  "x1": 400,
  "y1": 50,
  "x2": 550,
  "y2": 150
}
```

Top Right

```
{
  "page": 1,
  "x1": 400,
  "y1": 750,
  "x2": 550,
  "y2": 850
}
```

Center

```
{
  "page": 1,
  "x1": 200,
  "y1": 400,
  "x2": 400,
  "y2": 500
}
```

Notes

- Coordinates must be within page boundaries (check PDF page size first)
- Visual signature appearance will be scaled to fit the defined box while maintaining aspect ratio
- If coordinates are invalid or out of bounds, the API returns 400 Bad Request
- For multi-page PDFs, specify the correct page number (1-indexed)
- Origin is always bottom-left regardless of PDF rotation
- 1 point = 1/72 inch = 0.3528 mm

Verification Response Fields

Field Definitions

When using the Verify Form/Text API or Verify PDF API, the response contains detailed verification results.

cert_name

- **Type:** string
- **Description:** Common Name (CN) of the signing certificate
- **Example:** `"Test1"`
- **Source:** Extracted from the certificate's subject field

signature_ok

- **Type:** boolean
- **Description:** Indicates whether the digital signature was created with the certificate's private key and whether the document has been altered since signing
- **True:** Signature is valid and document integrity is maintained
- **False:** Signature verification failed (document tampered or signature corrupted)
- **Process:**
  - Extract signature from document
  - Decrypt signature hash using certificate's public key
  - Compare with computed document hash
  - Match = true, Mismatch = false

hash_ok

- **Type:** boolean
- **Description:** Indicates whether the document hash matches the encrypted hash at the time of signing
- **True:** Content hash matches signed hash (content unchanged)
- **False:** Content has been modified since signature was applied
- **Process:**
  - Compute hash of current document content (excluding signature bytes)
  - Compare with hash embedded in signature
  - Match = true, Mismatch = false
- **Relation to signature_ok:** Usually matches signature_ok status, but provides granular integrity info

cert_ok

- **Type:** boolean
- **Description:** Indicates whether the signing certificate is valid and trusted
- **True:** Certificate is valid, issued by trusted CA, not expired, not revoked
- **False:** Certificate invalid, expired, revoked, or issued by untrusted authority
- **Checks performed:**
  - Certificate chain validation against trusted root CAs
  - Expiration date verification (notBefore, notAfter)
  - Revocation status (CRL or OCSP checking)

    - Certificate purpose and key usage validation

Final Boolean (Overall Status)

- **Type:** boolean
- **Position:** Last element in the message array
- **Description:** Aggregated verification status
- **True:** Only if ALL checks pass (signature_ok, hash_ok, cert_ok all true)
- **False:** If ANY check fails (one or more of the above are false)
- **Usage:** Quick indicator for pass/fail status without parsing individual fields

Example Interpretations

All Valid

```
{
  "cert_name": "John Doe",
  "signature_ok": true,
  "hash_ok": true,
  "cert_ok": true
}
// Final: true
// Meaning: Document signed by John Doe, not modified, certificate
trusted
```

Tampered Document

```
{
  "cert_name": "John Doe",
  "signature_ok": false,
  "hash_ok": false,
  "cert_ok": true
}
// Final: false
// Meaning: Document was altered after signing, but certificate is valid
```

Expired Certificate

```
{
  "cert_name": "John Doe",
  "signature_ok": true,
  "hash_ok": true,
  "cert_ok": false
}
```

```
// Final: false
// Meaning: Document is intact and signature valid, but signer's
certificate expired
```

Processing Logic

1. Extract signature and certificate from input (PDF or text)
2. **Signature Check:** Cryptographic verification using public key
3. **Hash Check:** Content integrity verification
4. **Certificate Check:** PKI validation (trust chain, expiry, revocation)
5. Aggregate individual booleans into final status
6. Return structured response

Notes

- A response with `final: false` means the signature cannot be trusted for legal or compliance purposes
- `signature_ok: false` almost always implies `hash_ok: false` (tampering affects both)
- `cert_ok: false` with others true indicates the document is authentic but the signer's identity cannot be trusted
- Always check the final boolean first for quick validation; inspect individual fields for detailed diagnostics
- Certificate revocation checking may introduce slight latency in verification response