Get started

Learn about DECTA API specifics before starting. Every new API integration is fully supported by a designated Project Manager. Please review this section carefully, and once you're ready, contact your Project Manager to get started.

Authentication and Authorization

Each request to the DECTA API must be signed with a certificate that allows DECTA to identify the API Customer. This guarantees the authenticity of the request received from the client.

Customer Certificate Signing

DECTA API supports SHA256 x509 certificates.

During the signing process, DECTA will compare the values in your request with the values stored in our system. If they do not match, the request will be declined.

> **Important Note:** The authentication process is executed only once for each client. Upon successful authentication, DECTA assigns a certificate to the client, which must be used in every future API request.

Here's a step-by-step guide to simplify the process:

Step 1: Define Data for the Certificate

There are 3 mandatory pieces of information required for signing. We strongly advise confirming these details with your DECTA Customer Manager before proceeding:

- **Legal Name (CN):** Usually your company name. It **must** be placed in the `CN` field of the certificate request.
- **IP Address:** All static IP addresses you will use to make API requests. They **must** be placed in the IP section of the `Subject Alternative Name` (SAN) field.
- **Secret Phrase:** A one-time secret token provided by DECTA. This **must** be passed in the `SECRET` header of your signing request.

Step 2: Create Certificate Request Configuration (`csr.cfg`)

To ensure all metadata is generated correctly, create a local configuration file named `csr.cfg` .

Below is an example. Make sure to replace the values in the `[ dn ]` and `[ alt_names ]` sections with your actual data:

Example

```
 [req]
default_bits = 2048
prompt = no
default_md = sha256
req_extensions = req_ext
distinguished_name = dn

[ dn ]
C=LV
ST=London
L=62 Bayswater Road
O=Decta
OU=IT
emailAddress=support@decta.com
CN = Decta Ltd.                      # Replace with your official Company Name

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
IP.1 = 127.154.125.1                # Replace with your primary static IP
IP.2 = 154.192.15.4                 # Replace with your secondary static IP (optional)
```

Step 3: Generate Key and CSR

Choose the most convenient method for your workflow to generate the Private Key and Certificate Signing Request:

Option A: Using OpenSSL

Run the following command in your terminal using the csr.cfg file created above:

```
 openssl req -new -sha256 -nodes -out request.csr -newkey rsa:2048 -keyout theKey.key -
config csr.cfg
```

> This will output theKey.key (**your private key - keep it safe!**) and request.csr (the signing request).

Option B: Using Python Script

If you prefer a programmatic approach, use this script to generate the metadata and key simultaneously:

Python Example

```
 import ipaddress
import os
import requests
from cryptography import x509
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.x509.oid import NameOID

# Configuration via environment variables
CERTIFICATE_SIGN_PATH = os.environ.get('CERTIFICATE_SIGN_PATH') # URL for DAPI certificate
sign request
SECRET_CODE = os.environ.get('SECRET_CODE')                   # Secret code provided by
DECTA

# Define the CSR Details (Subject) using standard DECTA metadata
subject = x509.Name([
    x509.NameAttribute(NameOID.COUNTRY_NAME, "GB"),                          # United
Kingdom
    x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, "Greater London"),    # County/State
    x509.NameAttribute(NameOID.LOCALITY_NAME, "London"),                     # City
    x509.NameAttribute(NameOID.ORGANIZATION_NAME, "Example company"),        # Legal
Company Name
    x509.NameAttribute(NameOID.ORGANIZATIONAL_UNIT_NAME, "Example Operations"),
    x509.NameAttribute(NameOID.EMAIL_ADDRESS, "admin@company.co.uk"),
    x509.NameAttribute(NameOID.COMMON_NAME, "company-name"),
])

# Define the CSR Details (AlternativeName) using your service static IP
san_extension = x509.SubjectAlternativeName([
    x509.IPAddress(ipaddress.IPv4Address("192.0.2.1"))
])

# Generate the Private Key
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)

# Build the CSR
csr = (
    x509.CertificateSigningRequestBuilder()
    .subject_name(subject)
    .add_extension(
        san_extension,
        critical=False,
    )
    .sign(private_key, hashes.SHA256())
)

# Serialize the Private Key (PEM format)
private_key_pem = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncryption()
)
```

```
# Serialize the CSR (PEM format)
csr_pem = csr.public_bytes(serialization.Encoding.PEM)

# Call certificate sign request
try:
    response = requests.post(
        CERTIFICATE_SIGN_PATH,
        json={"encodedCSR": csr_pem.decode('utf-8')},
        headers={
            "SECRET": SECRET_CODE,
            "Content-Type": "application/json"
        }
    )
    response.raise_for_status()

    print(f"HTTP Status: {response.status_code}")
    print(f"Response: {response.text}") # Save response data in a safe place!

    # Securely save the private key to a local file
    with open('private_key.pem', 'wb') as f:
        f.write(private_key_pem)
    print("Your private key has been safely saved to 'private_key.pem'")

except Exception as e:
    print(f"Error occurred: {e}")
```

Step 4: Submit and Sign Certificate

The final step is to send the content of your request.csr to the DECTA API endpoint (POST / v1/certificate/sign).

> **Crucial JSON Rule**: The value of the encodedCSR parameter must be formatted as a single line, with all newlines escaped as \n according to the JSON specification.

Example payload

```
 {
  "encodedCSR": "-----BEGIN CERTIFICATE REQUEST-----
\nMIIBtTCCARICAQAwGjEYMBYGA1UEAwwPU29tZSBDb21wYW55IEx0ZDCCASIwDQYJ\n...[truncated for
brevity]...\n-----END CERTIFICATE REQUEST-----\n"
}
```

Troubleshooting & Error Codes

If your request fails, check the HTTP status code returned by the API:

- **401 Unauthorized**: The value of the SECRET header is incorrect or expired.

- **412 Precondition Failed**: The certificate request contains wrong data. The CN or IP lists inside the CSR do not match the records in the DECTA database.

Once the certificate is successfully signed, you will receive its thumbprint.

> ⓘ Save both the signed certificate and the thumbprint for further usage.

Authorization & Request Signing

Each request must be signed using JOSE specifications (specifically RFC 7515). It is vital to follow the standard and use Base64url Encoding without Padding for token encoding.

> ⓘ **Important Note for JWS Payloads:** When signing a GET request that includes query parameters (such as data filters), the parameters **must** be included in the payload string you are signing. Append a "?" after the URI in the JWS Payload, followed by your parameters joined by "&".

Python Example: Request Signing and Execution

This script demonstrates how to form the correct JWS payload for a GET request with filters, sign it using your private key (theKey.key), and perform the authorized API call:

Python Example

```
 import json
import os
import requests
from jwcrypto import jwk, jws
from jwcrypto.common import json_encode
from urllib.parse import urlencode, quote

# Configuration via environment variables
DAPI_URL = os.environ.get('DAPI_URL')               # DAPI base URL
PRIVATE_KEY_ENV = os.environ.get('PRIVATE_KEY')     # PEM string (without end of line
symbols)
THUMBPRINT = os.environ.get('THUMBPRINT')           # Signed certificate thumbprint

# Define request data
uri = "/v1/api/cards"
params = {"product": "eq:100", "cardName": "eq:Test Name"}
data = "" # Empty body for GET request

# Format query parameters for the JWS Payload
query_string = urlencode(params, safe=' :', quote_via=quote)

# Construct JWS Payload: ensure URI is followed by '?' if query parameters exist
jws_payload = uri + ("?" + query_string if query_string else "") + data

# JWCrypto Setup
jwk_key = jwk.JWK.from_pem(PRIVATE_KEY_ENV.encode('utf-8'))
jwstoken = jws.JWS(jws_payload.encode('utf-8'))

# Add signature using RS256 and protected header
jwstoken.add_signature(
    jwk_key,
    'RS256',
    protected=json_encode({"x5t#S256": THUMBPRINT, "alg": "RS256"})
)
signature = json.loads(jwstoken.serialize())

# Send authorized GET request
try:
    response = requests.get(
        DAPI_URL + uri,
        data=data,
        params=params,
        headers={
            "token-header": signature['protected'],
            "token-signature": signature['signature'],
            "Content-Type": "application/json"
        }
    )
    print(f"HTTP Status: {response.status_code}")
    print(f"Response: {response.text}")
except Exception as e:
    print(f"Error occurred: {e}")
```

Requests with Filters

Several endpoints support data filters to narrow down the results.

Friendly Reminder: If you are appending these filters to a GET request, remember that the entire query string (starting with `?`) must be included in your signed JWS payload. See the Authorization & Request Signing section for details.

Filter Syntax

```
<requestFilter> ::= <filterName>=[eq|ne|gt|ge|lt|le|co|sw]:<filterValue>
```

Supported Operators

Operator	Name
`eq`	Equals
`ne`	Not equals
`gt`	Greater than
`ge`	Greater or equals than
`lt`	Less than
`le`	Less or equals than
`co`	Contains
`sw`	Starts with

Example:

```
cardState="eq:BLOCKED_BY_HOLDER"
```

```
/v1/api/cards?cardState=eq:BLOCKED_BY_HOLDER
```

DAPI Idempotency

Idempotency ensures that repeating the same API request multiple times produces the same result as a single request. This mechanism is used to safely handle retries caused by network issues, timeouts, or client-side errors.

To use the DAPI idempotency functionality, each API request must include a `Request-Id` header with a unique value in UUID format in each request. Accordingly, the DECTA API response includes the same `Request-Id` value.

Example:

DAPI User request	DAPI response

Request with `Request-Id`
Example:
`header: Request-Id={UUID}`
`payload:{"someData":"someValue"}`
	Response with the same `Request-Id` from client's request
	Example:
	`header: Request-Id={UUID}`
	`payload: {"someResponseData":"someResponseValue"}`

Important:

- `Request-Id` for any API request must be unique in UUID format.
- re-applying the same code `Request-Id` value in any of the API requests will:
  - show no error
  - produce the same result in the response as the first time