SecureDNA

Synthclient API

SecureDNA

Contents

This page documents the v1 production API https endpoints, which talk to our backend infrastructure.

Terminology

Uses of the term *MAY*, *SHOULD*, *MUST*, *MAY NOT*, *SHOULD NOT*, *MUST NOT*, and so forth, are interpreted according to RFC2119.

Security

This API is not a substitute for an architectural specification. However, for implementors' basic knowledge:

- Once the inputs below are handed to our library and/or REST API endpoint, they are immediately encrypted using a *blinding function* inside this endpoint. No unencrypted sequences are ever transmitted or stored anywhere.
- Thus, as long as an implementer runs a local synthclient instance, unencrypted sequence data will never leave their network.
- It is not possible for SecureDNA or any network eavesdropper to know which sequences are being screened; all computation is done on encrypted data.
- If a hazard hit is detected, then and only then is it even possible for SecureDNA to know anything about the sequence which hit, and the only disclosure is those particular short (30-60 bp) sequences which actually match a known hazard or a mutant/variant of one. SecureDNA uses that information to explain in its response which likely hazardous organism(s) matched the hit(s).
- Only *outbound* network connections are made from this endpoint to the SecureDNA screening infrastructure. No inbound network connections are ever required.

API examples

Here are some quick examples of how to use the API in Bash and Python:

Bash example

```bash
#!/usr/bin/env bash

# change this to your running instance of synthclient,
# e.g. 'http://localhost'
BASE_URL='http://your_synthclient_instance'

function format_json() {
  # use jq if it's available
  if command -v jq &>/dev/null; then
    jq '.'
  else
    cat
  fi
}

# If your version of curl doesn't have --no-progress-meter, you may
# remove it or use --silent instead. However, --silent will also
# suppress errors.

curl "$BASE_URL/v1/screen" \
  --header "Content-Type: application/json" \
  --no-progress-meter \
  --data-raw '
{
  "fasta": ">NC_007373.1\nGAATCGCAATTAACAATAACTAAAGAGAAAAAAGAAGAACTC",
  "region": "all",
  "provider_reference": "documentation"
}
' | format_json
```

Python example

```
#!/usr/bin/env python3

import json
import urllib.request
import urllib.error
import sys

# change this to your running instance of synthclient,
# e.g. 'http://localhost'
base_url = "http://your_synthclient_instance"


def print_json(s: str, file=sys.stdout) -> None:
    """Pretty-print a JSON string."""
    print(json.dumps(json.loads(s), indent=2), file=file)


fasta = """
>NC_007373.1
GAATCGCAATTAACAATAACTAAAGAGAAAAAAGAAGAACTC
"""
data = {
    "fasta": fasta,
    "region": "all",
    "provider_reference": "documentation",
}

request = urllib.request.Request(
    base_url + "/v1/screen",
    method="POST",
    data=json.dumps(data).encode("utf-8"),
)
```

```
request.add_header("Content-Type", "application/json")

try:
    response = urllib.request.urlopen(request)
    print_json(response.read().decode("utf-8"))
except urllib.error.HTTPError as err:
    print(f"Error: {err.code} {err.reason}", file=sys.stderr)
    print_json(err.read().decode("utf-8"), file=sys.stderr)
    sys.exit(1)
```

Note that the example uses `http`, but if you are running a network-accessible instance of synthclient, you may want to use an SSL-terminating proxy like nginx or Caddy and require `https` instead.

Note also that the `bash` script isn't designed for extremely long strings (it will complain) but it's easy to pass files to `curl` instead. If you have a very old version of `curl`, it may complain that either `--no-progress-meter` or `--data-raw` don't exist; if you can't upgrade, there are workarounds, but it's likely you may be using some other tool anyway as part of a real API and not `curl`.

For detailed explanations of the inputs and outputs to the API, please see the sections below.

Input fields

A synthclient API request has the following type, in Typescript syntax: (Note that a question mark after the field name means the field may be omitted.)

```
/** A request to the /v1/screen endpoint. */
interface ApiRequest {
  /**
   * The input FASTA. This field MUST be included.
   */
  fasta: string,

  /**
   * The screening region. This field MUST be included.
   * See below for more details.
   */
  region: "us" | "prc" | "eu" | "all";

  /**
   * An optional arbitrary string that will be returned in the
   * response, for your tracking purposes. This field MAY be included.
   *
   * Note that this string may be logged in our backend, so be careful
   * about including sensitive information (such as customer names).
   */
  provider_reference?: string | null,
}
```

`fasta` field

`fasta` is the actual FASTA information to check. This is a string containing any number of newline-separated records, each one of which looks like

```
>Nipah virus
ACCAAACAAGGGAGAATATGGATACGTTAAAATATATAACGTATTTTTAAAACTTAGGAA
CCAAGACAAACACTTTTGGTCTTGGTATTGGATCCTCAAGAAATATATCATCATGAGTGA
TATCTTTGAAGAGGCGGCTAGTTTTAGGAGTTATCAATCTAAGTTAGGGAGAGATGGGAG
```

Note that by our parsing rules, a bare DNA sequence (e.g., `"fasta": "GCAACATAGGAAACACACCTATGGGTCATG"`) is considered a valid FASTA with an empty header. See Synthclient_API#Notes for more details.

`region` field

`region` is the jurisdiction region you wish the server to use when evaluating whether a request should be granted. The options are:

- `"region": "us"` for the United States (Select Agent and Australia Group lists)
- `"region": "eu"` for the European Union (European Union and Australia Group lists)
- `"region": "prc"` for the People's Republic of China (PRC export control lists)
- `"region": "all"` for all regions (the request will be denied if it would be denied in any region)

The determination is made via organism tags; see the tags section below for more information.

`provider_reference` field

`provider_reference` is an arbitrary provider-supplied UTF-8 Unicode string. This field will be returned unchanged in the results, and will be signed over in those modes in which results are signed. This allows providers who store screening results in a database to correlate a particular order result with their own order reference (likely a PO number or some sort of UUID), and, if signed, to prove that this particular result was associated with the given provider identifier. It may also be logged by SecureDNA servers to enable debugging (wherein a provider informs SecureDNA of a particular string so it can be found in logs). Hence, providers SHOULD NOT include customer-proprietary data in this field. If this field is not supplied, returned results MAY either omit the field or emit it with a value of the null string; provider implementations SHOULD NOT depend upon one behavior or the other.

Notes

- The FASTA format is poorly standardized; see the URLs on this page for pointers to several partial and conflicting definitions. We adopt a consensus view and try to be liberal in what we accept.
- One *record* consists of one or more consecutive header lines, followed by one or (typically, many more) DNA nucleotides.
- There can be any number of these records concatenated into a single string; we will screen them all.
- Header lines *MUST* have > or the semi-obsolete `;` (0x3b) in the first column to be recognized as such.
- Header lines *MAY* use UTF-8, although > or `;` *MUST* be ASCII characters (0x3e or 0x3b).
- Header lines *MAY* be nonunique. In other words, the same header line *MAY* appear in more than one place in the input.
- Header lines are ignored for screening purposes but are used when identifying hazard hits for customer convenience.
- Any text present in the input before the first header line is treated as if it is sequence data; it is *not* ignored. In other words, in this case, the very first record *MAY* have *zero* header lines associated with it. (Some FASTA

files appear to treat this as a comment, but we cannot, because we have no guarantee that all providers will do so. Any provider which treated a headerless sequence as a synthesis request could therefore allow a trivial screening bypass if we ignored this text; customers with such files *SHOULD* be encouraged to fix them via *the provider* checking their input and complaining before attempting to screen.)

- Multiple header lines *MAY* appear with no intervening sequence; if so, they are treated as if all of them describe the following sequence.
- The sequence information itself *MAY* be on separate lines of any length, or on one long single line, with no line-length limit.
- Sequence information *MUST* use only ASCII characters. Characters outside of allowable DNA nucleotides, plus ASCII whitespace, will cause the request to be rejected.
  - We explicitly allow ambiguous DNA, aka wobble codes, as well as specific DNA bases. Thus, the following nucleotides are allowed: `ABCDGHKMNRSTVWY`.
  - Windows which contain wobbles are internally expanded to a large but variable number of possibilities and each possibility is screened. Windows which would exceed expansion limits are simply dropped and will not be screened. Thus, for example `NNNNNNNNNNNNNNNNNNNNAAAAAAAAAA` will not be screened, but `NNAAAAAAAAAAAAAAAAAAAAAAAAAAAA` definitely will be.
  - If the synthesis itself cannot support wobble codes, `synthclient` expects that its *caller* (automation at the provider or in the benchtop, upstream of `synthclient`) will take pains to inform the customer that the order cannot be synthesized as specified.
  - We do not allow amino acids, because the resulting order would be ambiguous due to degeneracy. However, the submitted DNA sequence is translated to both directions and all three reading frames of amino acids and those translations are also used for screening.
- Line termination *MAY* be ASCII newline ('\n', 0x0a), ASCII carriage return ('\r', 0x0d), or both ('\r\n', x0a0d).
- The input string is *NOT REQUIRED* to end with a line termination character, although it is likely that it will.
- Alphabetic case in sequences is ignored.
- A single screening request *MUST NOT* include more than one customer's order.
- A single screening request *MUST* include all of that customer's order.

(The latter two requirements are forward-looking to when screening includes a permissions system.)

Thus, this is a valid input:

```
> header 1
ACCAAACAAGGGAGAATATGGATACGTTAAAATATATAACGTATTTTTAAAACTTAGGAA
CCAAGACAAACACTTTTGGTCTTGGTATTGGATCCTCAAGAAATATATCATCATGAGTGA
> header 2
> header 3
GCTAGTTTTAGGAGTTATCAATCTAAGTTAGGGAGAGAT
```

Output fields

A synthclient API response has the following type, in Typescript syntax: (Note that a question mark after the field name means the field may be omitted.)

```
/** The top-level response. */
export interface ApiResponse {
    /** Whether synthesis should be allowed to proceed. */
    synthesis_permission: "granted" | "denied";
    /**
```

```
     * If provided in the input, `provider_reference` will be
     * returned here. `null` otherwise.
     */
    provider_reference?: string | null;
    /**
     * If `synthesis_permission` is `"denied"` due to one or more
     * screening hits, this list will contain those hits, grouped
     * by which record they occurred in.
     */
    hits_by_record?: FastaRecordHits[];
    /** Any non-fatal warnings will be in this list. */
    warnings?: ErrorOrWarning[];
    /**
     * Will contain fatal errors if `synthesis_permission
     * is `"denied"` due to an error.
     */
    errors?: ErrorOrWarning[];
}

/** Screening hits, grouped by which record they occurred in. */
export interface FastaRecordHits {
    /** The record header, possibly empty. */
    fasta_header: string;
    /** Line range in FASTA input this record covers. */
    line_number_range: [number, number];
    /** The length of the record sequence. */
    sequence_length: number;
    /**
     * The hits that occurred in this record, grouped by similarity.
     */
    hits_by_hazard: HazardHits[];
}

/** A list of hits grouped by similarity. */
export interface HazardHits {
    /** Whether this hit group matched nucleotides or amino acids. */
    type: "nuc" | "aa";
    /**
     * Whether this hit group matched a hazard wild type
     * (observed genome) or predicted functional variant
     * (mutation SecureDNA believes would still be hazardous).
     * This field is always `null` for `type: "nuc"` hit groups.
     */
    is_wild_type: boolean | null;
    /**
     * A list of regions in the sequence that matched this
     * hazard group.
     */
    hit_regions: HitRegion[];
    /** The most likely organism match for this hazard group. */
    most_likely_organism: Organism;
    /**
     * All possible hazard matches for this hazard group,
     * including `most_likely_organism`.
     */
    organisms: Organism[];
}

/** A region of a record sequence that matched one or more hazards. */
export interface HitRegion {
    /** The matching subsequence. */
    seq: string;
    /** The start of `seq` in the record sequence, in bp. */
    seq_range_start: number;
    /** The (exclusive) end of `seq` in the record sequence, in bp. */
    seq_range_end: number;
}

/** Organism metadata. */
export interface Organism {
    /** The SecureDNA name for this organism. */
    name: string;
    /** The high-level classification of this organism. */
    organism_type: "Virus" | "Toxin" | "Bacterium" | "Fungus";
    /** A list of NCBI accession numbers for this organism. */
    ans: string[];
    /**
     * A list of SecureDNA tags for this organism.
     * A table of current tags is included below,
     * but more may be added in the future.
     */
    tags: string[];
}

/** An error or warning. */
export type ErrorOrWarning = {
    /**
     * The diagnostic code.
     * A list of current diagnostic codes is provided
     * below, but more may be added in the future.
     */
    diagnostic: string;
    /** Additional information about the cause of this error. */
    additional_info: string;
    /**
     * If applicable, a line number range in the
     * input FASTA that caused this error or warning.
     */
    line_number_range?: [number, number] | null;
}
```

Following are examples of typical responses.

Example Response: No hazards detected

No hazards were detected in the customer's input, therefore synthesis *SHOULD* proceed.

```
{
  "synthesis_permission": "granted"
}
```

Example Response: Hazards detected

Hazards were detected in the customer's input, therefore synthesis *MUST NOT* proceed without some additional authorization. (We intend to soon make available a certificate system allowing customers to be preapproved for particular sequences by their regulatory agencies, allowing SecureDNA to check such preapprovals, without requiring human input, and authorize synthesis if the customer's preapproval matches all hazards in the order submitted.)

```
{
  "synthesis_permission": "denied",
  "hits_by_record": [
    {
      "fasta_header": "MERS_segment_2",
      "line_number_range": [24, 79],
      "sequence_length": 1234,
      "hits_by_hazard": [
        {
          "type": "nuc",
          "is_wild_type": null,
          "hit_regions": [
            {
              "seq": "CTTCATCCGCACGTGCCAGACCCTTATTCTAAGGTGGCACTT",
              "seq_range_start": 0,
              "seq_range_end": 42
            }
          ],
          "most_likely_organism": {
            "name": "coronavirus MERS-CoV",
            "organism_type": "Virus",
            "ans": ["GCA_000901155.1", "NC_019843.3"],
            "tags": [
              "AustraliaGroupHumanAnimalPathogen",
              "EuropeanUnion",
              "PotentialPandemicPathogen",
              "HumanToHuman"
            ]
          },
          "organisms": [
            {
              "name": "coronavirus MERS-CoV",
              "organism_type": "Virus",
              "ans": ["GCA_000901155.1", "NC_019843.3"],
              "tags": [
                "AustraliaGroupHumanAnimalPathogen",
                "EuropeanUnion",
                "PotentialPandemicPathogen",
                "HumanToHuman"
              ]
            },
            {
              "name": "Human coronavirus 229E",
              "organism_type": "Virus",
              "ans": ["GCA_000853505.1"],
              "tags": ["HumanToHuman"]
            }
          ]
        }
      ]
    }
  ]
}
```

Organism type tags

Tags allow providers to determine *why* an organism was flagged. Not all tags indicate that a given organism is regulated via lists such as the Australia Group; some exist to allow providers to give particular orders extra scrutiny even if they are not regulated.

Tags may represent:

- a transmission pathway
- a risk categorization
- inclusion in a list published by an organization of sequences of concern

Note:

- Tags may be added in the future.
- Code should treat tags in a *case-independent* manner.
- Many organisms have multiple simultaneous tags.

This table contains the current tags and whether they will trigger a denial in a specific region. However, note that because organisms can have multiple tags, these denials are *cumulative*: if an organism has the **AustraliaGroupHumanAnimalPathogen** tag (which is not denied in the PRC), it may nonetheless be denied in the PRC if it also has the **PRCExportControlPart1** tag.

Tag	`"us"`	`"eu"`	`"prc"`	`"all"`
Regulated lists:				
SelectAgentHhs[1]	**Denied**	Granted	Granted	**Denied**
SelectAgentUsda[2]	**Denied**	Granted	Granted	**Denied**
SelectAgentAphis[3]	**Denied**	Granted	Granted	**Denied**
AustraliaGroupHumanAnimalPathogen[4]	**Denied**	**Denied**	Granted	**Denied**
AustraliaGroupPlantPathogen[5]	**Denied**	**Denied**	Granted	**Denied**
PRCExportControlPart1[6]	Granted	Granted	**Denied**	**Denied**
PRCExportControlPart2[7]	Granted	Granted	**Denied**	**Denied**
EuropeanUnion[8]	Granted	**Denied**	Granted	**Denied**
Risk categorization/transmission pathways:				
ArthropodToHuman[9]	Granted	Granted	Granted	Granted
HumanToHuman[10]	Granted	Granted	Granted	Granted
PotentialPandemicPathogen[11]	**Denied**	**Denied**	**Denied**	**Denied**
RegulatedButPass	Granted	Granted	Granted	Granted

The two transmission pathway tags (**ArthropodToHuman** and **HumanToHuman**), along with the **PotentialPandemic-Pathogen** and **RegulatedButPass** tags, are SecureDNA-specific classifications:

- **ArthropodToHuman** and **HumanToHuman** do not themselves affect the `synthesis_permission` flag.
- **PotentialPandemicPathogen** *does* set the `synthesis_permission` to `denied`. The intent is to force human review and/or exemption-list handling for these organisms, even though many are not (yet) in the various lists above.
- **RegulatedButPass** is a tag with special behavior. It isn't inherent to an organism, but instead is added based on what a hit (item in `hits_by_hazard`) matches. If the SecureDNA system detects that a hit is likely low-risk (for example, matching against non-pathogenic or non-toxin-producing genes), the tag will be added to relevant organisms in the `organisms` list for that hit, based on internal data. **RegulatedButPass** doesn't affect the `synthesis_permission` flag, but is instead meant as a signal to human reviewers that a hit, especially one in an order made by a trusted customer, may be lower risk. If a hit matches *any* high-risk genes, the **RegulatedButPass** tag will not appear on *any* organism in that hit's `organisms` list. The **RegulatedButPass** tag has *no relation* to whether the DNA matched by a hit is legal to synthesize or export from a given region.

Warnings

Warnings are nonfatal. Synthesis may or may not proceed based on the value of `synthesis_permission`. Warnings may indicate to the customer that their input data may not be interpreted as they think or that some other condition has arisen to which the customer may want to attend. For example, we may issue warnings if a certificate

[1] Select Agent: US Department of Health and Human Services
[2] Select Agent: US Department of Agriculture
[3] Select Agent: USDA Animal and Plant Health Inspection Service
[4] Australia Group human and animal pathogens
[5] Australia Group plant pathogens
[6] People's Republic of China export control list (2002), part 1
[7] People's Republic of China export control list (2002), part 2
[8] European Union export and dual-use control list
[9] Arthropod-to-human transmissible pathogen
[10] Human-to-human transmissible pathogen
[11] Potential pandemic pathogen

will expire soon to encourage renewal before expiration.

```
{
  "synthesis_permission": "granted",
  "warnings": [
    {
      "diagnostic": "certificate_expiring_soon",
      "additional_info": "The provided certificate is expiring soon, at
       ↪  2024-02-14T04+00:00."
    }
  ]
}
```

Errors

Errors are fatal. Synthesis *MUST NOT* proceed. (Otherwise, a simple network interruption would allow trivial bypassing of screening and allow synthesizing anything.) Errors indicate serious problems either with the customer's input, with the screening system itself, or the reachability of the screening infrastructure from the provider attempting screening.

```
{
  "synthesis_permission": "denied",
  "errors": [
    {
      "diagnostic": "invalid_input",
      "additional_info": "Error parsing FASTA: error parsing record: bad
       ↪  nucleotide: '@'",
      "line_number_range": [103, 103]
    }
  ]
}
```

The `diagnostic` field contains a short string describing the error type. The current diagnostic codes are as follows.

- `not_found` — the request was made to an unknown URL.
- `internal_server_error` — the server encountered an error.
- `invalid_input` — the request is formatted incorrectly in some way.
- `request_too_big` — the request FASTA exceeds configured size limits.
- `too_many_requests` — a rate limit has been exceeded. (More information is available in the `additional_info` field.)
- `unauthorized` — the request lacks authorization. (This diagnostic is currently not produced by the `/screen` endpoint. If a request is made with invalid or expired certificates, the result is 400 `unable to verify the signature`.)

The `additional_info` field contains a longer description of what caused the error. Implementations *MUST NOT* depend on the contents of this field, as it is liable to change with system updates.

If applicable, the `line_number_range` field contains the line numbers (in the input FASTA) that caused the error.

Implementations must fail closed

Implementations *MUST* default to *DENY* permission for synthesis unless otherwise instructed. This means that bugs in either end of the protocol or in the implementation will cause immediate failures and be detected ("fail closed"), rather than being silently ignored and enabling the synthesis of dangerous organisms ("fail open").

This means that all of these potential situations, which are likely implementation bugs, should *DENY* permission:

- Failure to parse the resulting JSON at all.
- Failure to find the `synthesis_permission` field.
- Any `synthesis_permission` value which is not equal to the ASCII string `granted`. In particular, implementations *MUST NOT* assume that they should instead check for the string `denied` and deny synthesis; it is safer from an engineering perspective to only allow synthesis if the value `granted` is found.

In addition, implementations *MUST* obey the `synthesis_permission` value in determining whether to synthesize, and *MUST NOT* attempt to instead make this decision based on whether the `hits_by_hazard` field is present. Future versions of this software will enable a permission system, whereby authorized customers may be able to synthesize particular subsets of hazardous sequences if preemptively granted permission by an authorizing body, and such data returns *MAY* include a non-empty `hits_by_hazard` field while nonetheless granting permission to synthesize.