Trusted Contract Negotiation HTTP/TLS/JSON

Interface Design Description

Service ID: *"trusted-contract-negotiation"*

Abstract
This document describes a HTTP/TLS/JSON variant of the Contract Negotiation service that does not rely on cryptographic signatures to establish message authenticity and integrity.

ARTEMIS Innovation Pilot Project: Arrowhead
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Production and Energy System Automation Intelligent-Built environment and urban infrastructure for sustainable and friendly cities]

Contents

1 Overview

This document describes the HTTP/TLS/JSON variant of the Trusted Contract Negotiation (TrCoNe) Ecplipe Arrowhead service, which is meant to enable distinct parties to digitalize their contractual interactions. Examples of such interactions could be taking on the obligation to deliver a good, negotiating insurance, agreeing to pay by invoice in exchange for access to a particular Ecplipe Arrowhead service, among many other.

This document exists as a complement to the *Trusted Contract Negotiation – Service Description* (TrCoNeSD) document. For further details about how this service is meant to be used, please consult that document. The rest of this document describes how to realize the Trusted Contract Negotiation (TrCoNe) service using HTTP [1], TLS [2] and JSON [3], both in terms of its functions (Section 2) and its information model (Section 3).

2 Service Functions

This section lists the functions that must be exposed by TrCoNe services in alphabetical order. In particular, each subsection first names the HTTP method and path used to call the function, after which it names an abstract function from the CoNeSD document, as well as input and output types. All functions in this section respond with the HTTP status code `204 No Content` if called successfully.

2.1 POST /trusted-contract-negotiation/acceptances

Interface: Accept
Input: Acceptance

Called to accept a previously received Offer, as exemplified in Listing 1.

```
POST /trusted-contract-negotiation/acceptances HTTP/1.1

{
  "negotiationId": 23421,
  "offerorName": "A",
  "acceptorName": "B",
  "acceptedAt": "2020-06-04T11:14:45.345Z"
}
```

Listing 1: An Accept invocation.

2.2 POST /trusted-contract-negotiation/counter-offers

Interface: CounterOffer
Input: CounterOffer

Called to make a counter-offer, replacing a previously received Offer part of some identified negotiation session. An example is given in Listing 2.

```
POST /trusted-contract-negotiation/counter-offers HTTP/1.1

{
  "negotiationId": 23421,
  "offerorName": "B",
  "receiverName": "A",
  "validAfter": "2020-06-04T11:19:13.142Z",
  "validUntil": "2020-06-05T11:19:13.142Z",
  "contracts": [{
    "templateName": "SimplePurchase",
    "arguments": {
      "buyer": "A",
      "seller": "B",
      "quantity": "20",
      "articleNumber": "ABC-123",
      "price": "6150"
    }
  }],
  "offeredAt": "2020-06-04T11:19:13.142Z"
}
```

Listing 2: An CounterOffer invocation.

2.3 POST /trusted-contract-negotiation/offers

Interface: Offer
Input: Offer
Output: RandomID

Called to make an initial offer, meaning that it creates a new negotiation session, for the caller and callee to make one or more Contracts. An example is given in Listings 3 and 4.

```
POST /trusted-contract-negotiation/offers HTTP/1.1

{
  "offerorName": "A",
  "receiverName": "B",
  "validAfter": "2020-06-04T11:18:17.849Z",
  "validUntil": "2020-06-05T11:18:17.849Z",
  "contracts": [{
    "templateName": "SimplePurchase",
    "arguments": {
      "buyer": "A",
      "seller": "B",
      "quantity": "12",
      "articleNumber": "ABC-123",
      "price": "3750"
    }
  }],
  "offeredAt": "2020-06-04T11:18:17.849Z"
}
```

Listing 3: An Offer invocation.

```
HTTP/1.1 204 No Content
location: /trusted-contract-negotiation/negotiations/23421
```

Listing 4: A successful Offer invocation yields a response with a location header set, which, most significantly, contains the session identifier assigned to the negotiation session created by the request. In this example the identifier is 23421.

2.4 POST /trusted-contract-negotiation/rejections

Interface: Reject

Called to signal the desire to immediately terminate an identified negotiation session.

```
POST /trusted-contract-negotiation/rejections HTTP/1.1

{
  "negotiationId": 23421,
  "offerorName": "A",
  "rejectorName": "B",
  "rejectedAt": "2020-06-04T11:14:45.345Z"
}
```

Listing 5: A request to terminate negotion 23421.

3 Information Model

Here, all data objects that can be part of the service calls associated with this service are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.7, which are used to represent things like hashes, identifiers and texts.

3.1 struct Acceptance

An accepted Offer, meant to be legally binding.

Object Field	Value Type	Description
”negotiationId”	RandomID	Identifies negotiation session containing accepted candidate offer.
”offerorName”	Name	Offeror’s common name.
”acceptorName”	Name	Acceptor’s common name.
”acceptedAt”	DateTime	The time at which the offer was accepted.

3.2 struct Argument

A concrete Contract parameter value. In other words, this data structure identifies a variable parameter in a template and provides a concrete parameter JSON Value. Conceptually, the parameter specification in the template that matches the *”Name”* field could contain additional type information, such as range restrictions, required list lengths, or regular expressions.

As the *”arguments”* field of the Contract type is a JSON Object, which maps String names to String values, there is no need for a dedicated *Argument* type. This type exists only implicitly and is documented here only for the sake of completeness.

3.3 struct Contract

A parameterized reference to a *template*, representing certain rights and obligations accepted by two parties. In other words, a contract derives its meaning from both the template it refers to and the arguments it contains. Note that contract objects only become binding if they refer correctly to legally valid templates and become part of properly signed Acceptances.

Object Field	Value Type	Description
”templateName”	Name	Name of invoked *template*.
”arguments”	Object<String>	Arguments matching *parameters* defined in *template*.

3.4 struct CounterOffer

A counter-offer, meant to replace the candidate offer in a identified negotiation session. This data structure differs from Offer only in that it contains a negotiation identifier.

Field	Type	Description
”negotiationId”	RandomID	Identifies an existing or new negotiation session.
”offerorName”	Name	Offeror’s common name.
”receiverName”	Name	Receiver’s common name.
”validAfter”	DateTime	The time after which the offer may be accepted.
”validUntil”	DateTime	The time after which the offer may no longer be accepted.
”contracts”	Array<Contract>	Offered contracts.
”offeredAt”	DateTime	The time at which this offer was made.

3.5 struct Offer

An initial offer, meant to cause a new negotiation session to be created.

Field	Type	Description
”offerorName”	Name	Offeror’s common name.
”receiverName”	Name	Receiver’s common name.
”validAfter”	DateTime	The time after which the offer may be accepted.
”validUntil”	DateTime	The time after which the offer may no longer be accepted.
”contracts”	Array<Contract>	Offered contracts.
”offeredAt”	DateTime	The time at which this offer was made.

3.6 struct Rejection

Identifiers a negotiation session a sending party wishes to terminate.

Object Field	Value Type	Description
”negotiationId”	RandomID	Identifies rejected negotiation session.
”offerorName”	Name	Offeror’s common name.
”rejectorName”	Name	Rejector’s common name.
”rejectedAt”	DateTime	The time at which the negotiation session was terminated.

3.7 Primitives

As all messages are encoded using the JSON format [3], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

With these primitives now available, we proceed to define all the types specified in the CoNeSD document without a direct equivalent among the JSON types. Concretely, we define the CoNeSD primitives either as

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [4], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

aliases or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.7.1 alias DateTime = String

Pinpoints a moment in type by providing a formatted string that conforms to the RFC 3339 specification [5] (ISO 8601). Naively, the format could expressed as ”YYYY-MM-DDTHH:MM:SS.sssZ”, where ”YYYY” denotes year (4 digits), ”MM” denotes month starting from 01, ”DD” denotes day starting from 01, ”HH” denotes hour in the 24-hour format (00-23), ”MM” denotes minute (00-59), ”SS” denotes second (00-59) and ”sss” denotes second fractions (0-999). ”T” is used as separator between the date and the time, while ”Z” denotes the UTC time zone. At least three fraction digits should be used, which gives millisecond precision. An example of a valid date/time string is ”2019-09-19T15:20:50.521Z”. Other forms or variants, including the use of other time zones, is adviced against.

3.7.2 alias Name = String

A String that is meant to be short (less than a few tens of characters) and both human and machine-readable.

3.7.3 alias RandomID = Number

An integer Number, originally chosen from a secure source of random numbers. When new RandomIDs are created, they must be ensured not to conflict with any relevant existing random numbers.

4 References

[1] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” RFC 7230, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7230

[2] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC8446

[3] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[4] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

[5] “Date and Time on the Internet: Timestamps,” RFC 3339, 2002, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC3339

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

5.2 Quality Assurance

No.	Date	Version	Approved by
1			