Trusted Contract Negotiation

Service Description

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

Abstract

This document describes a 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 Trusted Contract Negotiation (TrCoNe) service, which enables systems to take part in contractual negotiations without the use of cryptographic signatures. In other words, the TrCoNe service provides a similar abstract interface and data types as the Contract Negotiation (CoNe) service, with the exception that signatures are not used at all. In addition, it uses simple Names instead of certificate fingerprints to identify the parties making and receiving offers. This means that using the TrCoNe requires trusting that the system hosting the service does not to abuse its position. Trusting a system in this manner might be important for performance reasons, or to be able to offload cryptographic identity management from systems that need to take direct part in contractual collaborations.

Readers of this document are assumed to be familiar with the CoNe service. For more information about that service, please refer to the Contract Negotiation Service Description (CoNeSD).

The rest of this document is organized as follows. In Section 2, we describe the abstract message functions provided by the service. In Section 3, we end the document by presenting the data types used by the mentioned functions.

2 Service Interfaces

This section lists the functions that must be exposed by TrCoNe services in alphabetical order. In particular, each subsection names an abstract interface, an input type and an output type, in that order. The input type is named inside parentheses, while the output type is preceded by a colon. Input and output types are only denoted when accepted or returned, respectively, by the interface in question.

All abstract data types named in this section are defined in Section 3. To better understand how these functions are meant to be used, please refer to the CoNeSD document.

2.1 function Accept (Acceptance)

Called to accept a previously received Offer, putting its Contracts into effect.

2.2 function CounterOffer (CounterOffer)

Called to make a counter-offer, replacing a previously received Offer part of some identified negotiation session.

2.3 function Offer (Offer) : 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. If the callee deems the Offer acceptable, it may proceed by wrapping the Offer that contains it into an Acceptance and call the Accept interface of the caller.

2.4 function Reject (Rejection)

Called to signal the desire to immediately terminate an identified negotiation session. The call is to be interpreted as if the caller is no longer interested in continuing the negotiation. If wanting to continue the negotiation, a counter-offer should be made instead.

3 Information Model

Here, all data objects that can be part of CoNe service calls are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is used to denote a collection of named fields, each with its own data type. As a complement to the explicitly defined types in this section, there is also a list of implicit primitive types in Section 3.7, which are used to represent things like hashes and identifiers.

An overview of the data object types is illustrated in Figure 1.

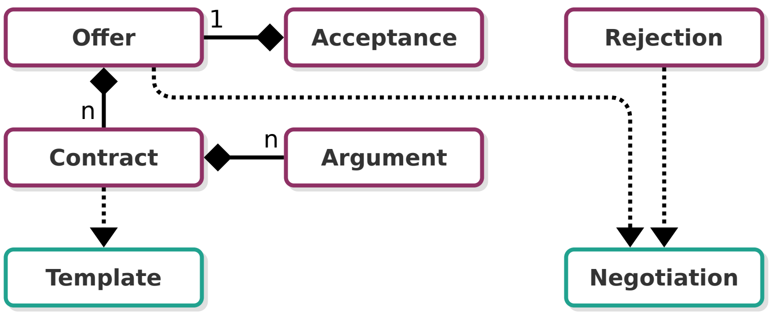

Figure 1: Information model as a UML class diagram. Black rhombuses signify data ownership, while dotted arrows denote references. The type closest to each rhombus is the one containing the reference or owning the data. Numbers are used to specify how many objects are referred to or owned, where n is to be regarded as an arbitrary positive number or zero. Note that *Template* and *Session* are only indirectly part of the information model, but are included for the sake of completeness.

3.1 struct Acceptance

An accepted Offer, meant to be legally binding.

Field	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 value. The type of the *Value* field should, in a non-naive implementation, be dynamic enough to be able to take on at least a few concrete types, such as integer, float, string, list, associative array, and so on. 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.

Field	Type	Description
name	Name	Human and machine-readable name of parameter.
value	Custom	Concrete value.

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 Acceptances.

Field	Type	Description
templateName	Name	Name of invoked *template*.
arguments	List<Argument>	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	List<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	List<Contract>	Offered contracts.
offeredAt	DateTime	The time at which this offer was made.

3.6 struct Rejection

Identifies a negotiation session a sending party wishes to terminate, as described in the CoNeSD document.

Field	Type	Description
negotiationId	RandomID	Identifies negotiation session to be terminated.
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

Types and structures mentioned throughout this document that are assumed to be available to implementations of this service. The concrete interpretations of each of these types and structures must be provided by any IDD document claiming to implement this service.

Type	Description
Custom	Any suitable type chosen by the implementor of the service.
DateTime	Pinpoints a specific moment in time.
List<A>	An *array* of a known number of items, each having type A.
Name	A string identifier that is intended to be both human and machine-readable.
RandomID	An identifier produced using a cryptographically secure random function.

4 Revision History

4.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

4.2 Quality Assurance

No.	Date	Version	Approved by
1			