Basismodul VAU-Kanal, Version 1.1.0

Überblick

Das vorliegende Modul zeigt exemplarisch, wie das im Spezifikationsdokument [gemSpec_Krypt] normativ festgelegte Kommunikationsprotokoll zwischen VAU und ePA-Clients implementiert werden kann.

Spezifikationsbasis ist "Release 3.1.2 Online-Produktivbetrieb".

Aufbau des Moduls

Die Implementierung erfolgt in Java (Version 11). Buildsystem ist Apache Maven (Version 3.6.1).

- **vauchannel** ist das maven reactor module, das die einzelnen Teilmodule in einem Build baut.
- **vauchannel-contract-2-java** legt das Format der ausgetauschten JSON-Nachrichten per JSON Schema (draft-04) und damit unabhängig von der Implementierungssprache fest. Aus den Schemas werden Java Klassen generiert.
- **vauchannel-protocol** implementiert das Protokoll mit der Verarbeitung der Nachrichten in der Javaklasse `VAUProtocol.java`. Die Definition des Kommunikationsprotokoll zwischen VAU und ePA-Clients aus [gemSpec_Krypt] Kapitel 6 werden in `VAUProtocol.java` detailliert kommentiert auf die Java-Implementierung gemappt. Abgedeckt sind
  - der Handshake,
  - der verschlüsselte Nutzdatentransport,
  - das Fehlerhandling. Alle benötigten kryptografischen Funktionen werden über ein Interface `VAUProtocolCrypto` angesprochen. Die konkrete

Implementierung des Interfaces ist damit unabhängig von der Protokollimplementierung. Sessioninformationen werden in der `VAUProtocolSession` abgelegt.

- **vauchannel-cxf** zeigt die Einbettung in die WebService-Bibliothek Apache CXF (XML/SOAP für die fachlichen Services und JSON/REST für das VAUProtokoll). Der VAU-Kanal startet und endet in dieser Einbettung in CXF-spezifischen Interceptoren. Die konkrete Implementierung des VAUProtokolls mit Kryptofunktionen und Ablage der Sessioninformationen wird als "dependency injected".
- **vauchannel-server** zeigt exemplarisch die Umsetzung der serverseitigen Endpunkte für Handshake und fachlichen Service (hier ein sayHello-Service) als Spring-Boot-Anwendung. Nicht thematisiert:
  - Das Ansprechen von Handshake-Service und OpenContext-Service unter einem Interface, das beispielsweise durch Trennung der Anfragen über den HTTP-Header `Content-Type` in einem vorgelagerten Gateway erfolgen kann.
  - Die Verwaltung mehrerer `VAUProtocolSession`.
- **vauchannel-client** implementiert einen Spring-Boot-Client, der gegen einen laufenden vauchannel-server Integrationstests ausführt.

Bauen

Das Modul **vauchannel** wird samt seinen Untermodulen wie folgt gebaut:

```
mvn clean install
```

Das Ergebnis des Builds sollte etwa so aussehen:

```
[INFO] ------------------------------------------------------------------
[INFO] Reactor Summary for vauchannel 1.1.0:
[INFO]
[INFO] vauchannel ............................ SUCCESS [  0.256 s]
[INFO] vauchannel-contract-2-java ............ SUCCESS [  2.333 s]
[INFO] vauchannel-protocol ................... SUCCESS [  6.351 s]
[INFO] vauchannel-cxf ........................ SUCCESS [  0.686 s]
[INFO] vauchannel-server ..................... SUCCESS [  7.882 s]
[INFO] vauchannel-client ..................... SUCCESS [  1.019 s]
```

```
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
```

Starten

Den Server starten

```
cd vauchannel-server/target
java -jar vauchannel-server-1.1.0.jar
```

und vom Client aus Integrationstests ausführen

```
cd vauchannel-client
mvn verify -Ptest
```

Release-Notes

V1.1.0, 2019.09.18

- Anpassung auf Spezifikationsstand zum Online-Produktivbetrieb, Release 3.1.2.
- Insbesondere wurde Change C_7029 implementiert, der für einen verschlüsselten Transport des Content-Type HTTP-Headers sorgt.

V1.0.0, 2019.09.03

- Initiale Implementierung gemäß der Spezifikationsdokumente zum Online-Produktivbetrieb, Release 3.1.1.