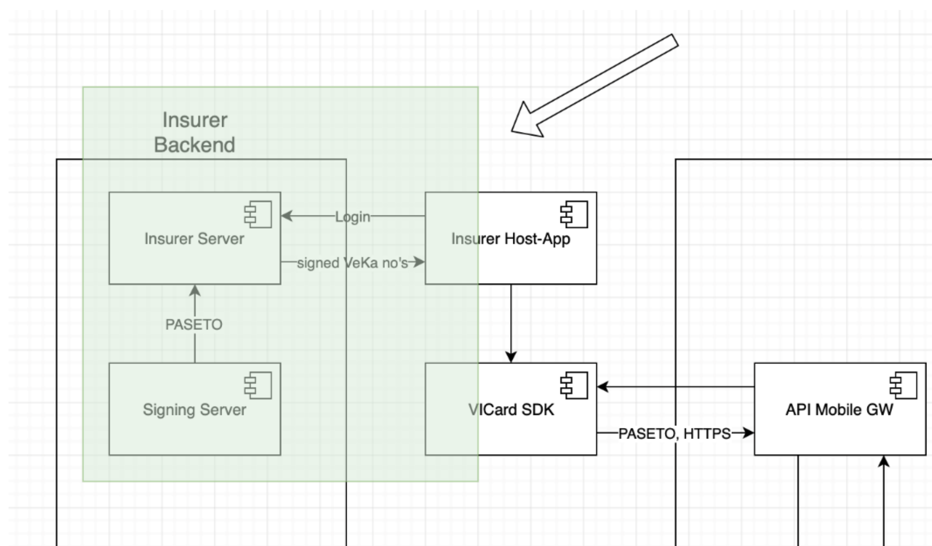
Insurer/Signing Server: Overview and Development Guide

Release Information	**1.00**
Target audience	This documentation is intended for developers and system integrators providing an Insurer Server for their App using the VICard SDK.
Summary	The Insurer Server is used to provide **vekaNumberClaim**-List**s** to access insurance card data in a secure manner.

Content

- 1 Overview
  - 1.1 Certificate Signing Process
  - 1.2 Key-pair creation by the insurer
  - 1.3 insurerAccessToken creation by Sasis
- 2 Development Guide
  - 2.1 Creation of an Insurer Server



(See Architecture / Documentation for more Detail)

Overview

The insurer server must be implemented by the insurer and is needed to create signed insurance number claims that are used by the VICARD SDK to get card- data for a specific insured person.

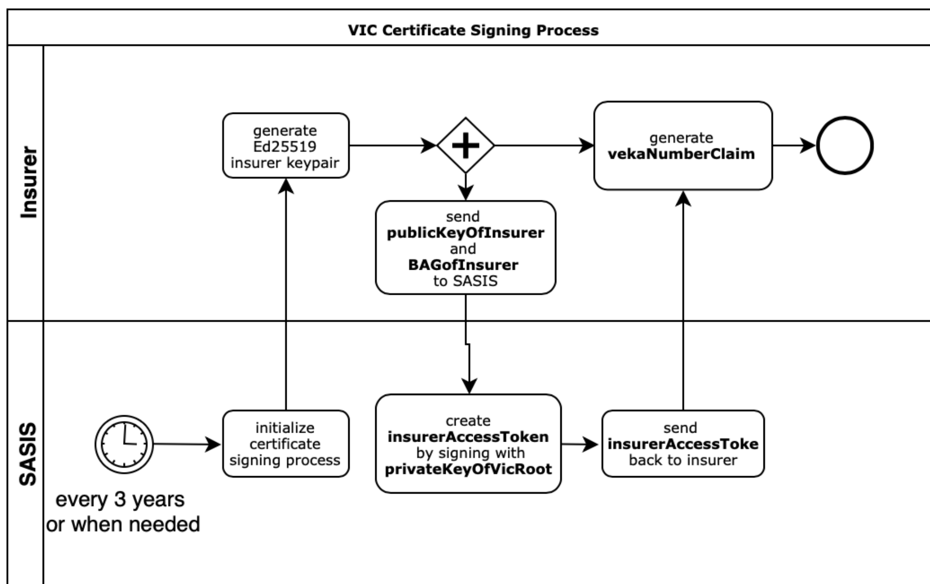
Certificate Signing Process

SASIS and the Insurer each create Ed25519 key-pairs compatible with paseto.

The insurer has to generate a key-pair and send his BAG-Nr. (BAGofInsurer) as well as the public key of the generated key-pair to Sasis.

Sasis then creates the insurerAccessToken via a local endpoint in the VicService and sends it to the insurer.

The insurer develops the Insurer Server using the insurerAccessToken to provide a vekaNrToken for the SDK.



Key-pair creation by the insurer

The insurer generates an Ed25519 keypair using the provided web app. The web app works locally in the browser.

Live Environment:

https://www.vvk-online.ch/virtual/keypair-generator.html

The resulting key-pair is stored in a Signing Server under control of the insurance. This function of this server is to perform paseto signatures.

The private key never leaves this location.

InsurerAccessToken creation by Sasis

This key-pair generated in the last section is used for requests and responses from the insurance. The public key part of the key-pair is required to generate the insurerAccessToken.

The insurerAccessToken is a public paseto token containing the **publicKeyOfInsurer** and **BAGofInsurer** (or other identification) of the insurance.

To get the **insurerAccessToken**, the insurer sends the previously generated **publicKeyOfInsurer** together with the **BAGofInsurer** to SASIS.

This is done via an email to Sasis:

To: vicard@sasis.ch
Subject: VICARD insurerAccessToken Request

Body:
Please generate and return an insurerAccessToken(s) for the following information:
BAGofInsurer: (Your "BAGofInsurer" here, can be more than one)
publicKeyOfInsurer: (Your "publicKeyOfInsurer" here, can be more than one)
SASIS Environment: (Live)

Sasis then creates the **insurerAccessToken** by using an endpoint in the VicService.

This endpoint is and should not be publicly available, and is available only for internal use by SASIS employees.

Sasis calls this endpoint passing the **BAGofInsurer** and the **publicKeyOfInsurer** The service internally uses the Sasis VIC Root key-pair to create the

insurerAccessToken. The VIC Root key-pair is versioned, and new tokens will always be signed with the latest version.

The **insurerAccessToken** is sent to the insurer.

At the insurer, the **insurerAccessToken** should be stored on the insurer server. It is used in the *Insurer Server* developed by the insurer.

Development Guide

The Insurer Server is called by the insurers' host app. It generates a set of **vekaNumberClaim**s described below. These claims are put in the SDK using an SDK-API.

The Insurer Server is just a basic REST server. It calls the Signing Server implementing paseto. Using https://paseto.io/ anyone could write such a server depending on the language needed.

Creation of an Insurer Server

You need to provide an endpoint of a rest service that runs on your server environment. You call this endpoint from your APP and put the result in the SDK via a method of the SDKs API. The result is a list of **vekaNrToken**s.

1. Create a keypair using: https://www.vvk-online.ch/virtual/keypair-generator.html
2. Store private key as **privateKeyOfInsurer** in your SigningServer
3. Store public key as **publicKeyOfInsurer** in your SigningServer
4. Send an email with the following contents to vicard@sasis.ch (you can use one single request for all of your BAG numbers)
    a. your **BAGofInsurer**
    b. your generated **publicKeyOfInsurer**
5. Wait for the response to the eMail
6. Store the contained **insurerAccessToken**
7. Implement a BASIC Insurer REST server
    a. Create an endpoint that returns a list of signed **vekaNrToken**. This list is called **vekaNumberClaim**-List here. Find a detailed description of the **vekaNrToken** structure below.
8. From your APP, you call your rest server endpoint and put the resulting **vekaNumberClaim**-List in the SDK using the API: VicSdk.setVekaNrClaims(**vekaNumberClaim**-List) (details in Android or iOS SDK)

vekaNrToken

The **vekaNrToken** is a paseto token that contains the **cardIdentificationNumber** of an insured person as well as the **insurerAccessToken** and is signed using the **privateKeyOfInsurer**. It also has a footer containing the **publicKeyOfInsurer.**

1. Each **vekaNrToken** must contain
   1. the **cardIdentificationNumber** of the insured person,
   2. the current date.
   3. the **publicKeyOfInsurer** you stored in step 3
   4. **publicKeyOfInsuredPerson** you get from SDK API: "VicSdk.getUserPublicKey" (details in Android or iOS SDK)
   5. the **insurerAccessToken** you received in step 6
2. Use your Signing Server to sign with PasetoV2.sign (as described in https://paseto.io/) with
   1. payload **vekaNrToken**
   2. secret key (sk) **privateKeyOfInsurer**
   3. footer **publicKeyOfInsurer**

The signing should be performed using your SigningServer. See the following pages for example implementations in Java and Node.js.

The completed JSON of a vekaNrToken should look as follows:

---- JSON
{
 iss: "did:vic-v1:" + $**publicKeyOfInsurer**
 sub: "did:vic-v1:" + $**publicKeyOfInsuredPerson**
 iat: "2007-12-03T10:15:30+01:00" // ISO-8601 Date as String
 cardIdentificationNumber: "1234567890" // insurance card number
 insurerAccessToken: + $**insurerAccessToken** // **insurerAccessToken** is provided by sasis
}

Footer
publicKeyOfInsurer (HEX-Encoded)

Definition of fields:

- **iss**

The Issuer
Public Key of the insurer in the format: "did:vic-v1:$**publicKeyOfInsurer**"

- **sub**

  The Subject
  This is **publicKeyOfInsuredPerson**. This public key is requested from the SDK.

- **iat**

  Issued at
  String representation of ISO-8601 Date such as "2007-12-03T10:15:30+01:00".

- **cardIdentificationNumber**

  The card number of the insured person.

- **insurerAccessToken**

  The **insurerAccessToken**

An example of implementation in node.js:

```
//Load HTTP module
const http = require("http");
const paseto = require('paseto')
const asn = require('asn1.js');
const _sodium = require('libsodium-wrappers');
const { createPublicKey, createPrivateKey } = require('crypto');

const hostname = '127.0.0.1';
const port = 3000;

const { V2: { verify, sign } } = paseto

//is used to convert private key from hex
var OneAsymmetricKey = asn.define('OneAsymmetricKey', function () {
    this.seq().obj(
        this.key('version').int(),
        this.key('algorithm').use(AlgorithmIdentifier),
        this.key('privateKey').use(PrivateKey)
    );
});
var PrivateKey = asn.define('PrivateKey', function () {
    this.octstr().contains().obj(
        this.key('privateKey').octstr()
    );
});

var AlgorithmIdentifier = asn.define('AlgorithmIdentifier', function () {
    this.seq().obj(
        this.key('algorithm').objid(),
    );
});

const pubKeyOfInsuredPerson = "872d7967dd5c931ebf57aedfaffe6b50d7f13ee1a3005ea45cd172ad297d3996";
const insurerAccessToken = "12345832324812381928312 89";
//See the description to the VeKa Token in the Insurer Server Overview and Development Guide
const vekaToken = (pubkeyofInsurance) => {
    return {
        sub: "did:vic-v1:" + pubKeyOfInsuredPerson,
        iss: "did:vic-v1:" + pubkeyofInsurance,
        cardIdentificationNumber: "1234567890",
        insurerAccessToken: insurerAccessToken
    };
}
```

```
const privateKeyAsBuffer = () => {
    const privateKey =
"cadeed6c1388c58a0bc1baeb0a2667bcccf0001a4a5773f77f71505a1e62d35dc4aa1ab99be5e6b8858891aff1fdc20479af0efc2
cf222c372873751e27100e9";
    const key = _sodium.from_hex(privateKey);

    let sk_cp = Buffer.from(key.slice(0, 32));
    //convert sodium Buffer to der ed25512 buffer
    var output = OneAsymmetricKey.encode({
        version: 0,
        privateKey: { privateKey: sk_cp },
        //ed25512 algorithm
        algorithm: { algorithm: '1.3.101.112'.split('.') }
    }, 'der');
    output.write('04', 12, 1, 'hex');
    return output;
}
const pubkeyHex = "c4aa1ab99be5e6b8858891aff1fdc20479af0efc2cf222c372873751e27100e9";

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    (async () => {
        {
            const priv = createPrivateKey({ key: privateKeyAsBuffer(), format: 'der', type: 'pkcs8' });
            const pub = createPublicKey({ key: priv, format: 'der', type: 'pkcs1' });
            const token = await sign(vekaToken(pubkeyHex), priv, { footer: pubkeyHex });
            res.end(token);
            console.log(await verify(token, pub));
        }
    })()
});

//listen for request on port 3000, and as a callback function have the port listened on logged
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

An example of implementation in Java:

```java
import dev.paseto.jpaseto.*;
import java.security.*;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.time.Instant;
import java.util.Arrays;
import org.apache.commons.codec.DecoderException;
import org.bouncycastle.asn1.DEROctetString;
import org.bouncycastle.asn1.edec.EdECObjectIdentifiers;
import org.bouncycastle.asn1.pkcs.PrivateKeyInfo;
import org.bouncycastle.asn1.x509.AlgorithmIdentifier;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

//Java JDK 15 would work without BouncyCastle
/*import java.security.spec.EdECPrivateKeySpec;
import java.security.spec.NamedParameterSpec;*/

public class Paseto {
    private static final String PUBLIC_KEY_OF_INSURER_HEX =
"c4aa1ab99be5e6b8858891aff1fdc20479af0efc2cf222c372873751e27100e9";
    private static final String PRIVATE_KEY_OF_INSURER_HEX =
"cadeed6c1388c58a0bc1baeb0a2667bcccf0001a4a5773f77f71505a1e62d35dc4aa1ab99be5e6b8858891aff1fdc20479af0efc2
cf222c372873751e27100e9";
    private static final String PUBLIC_KEY_OF_INSURED_PERSON_HEX =
"fc48204734b1cd52b83356543a0d5376432c7c54cb18ad45869caed72755546d";

    public static void main(String[] args) throws NoSuchAlgorithmException {
        generateVekaNrToken();
    }

    public static void generateVekaNrToken() throws NoSuchAlgorithmException {
        PrivateKey privateKey = getPrivateKeyFromECBigIntAndCurve(PRIVATE_KEY_OF_INSURER_HEX);
        final String vekaToken = Pasetos.V2.PUBLIC.builder()
                .setIssuer("did:vic-v1:" + PUBLIC_KEY_OF_INSURER_HEX)
                .setSubject("did:vic-v1:" + PUBLIC_KEY_OF_INSURED_PERSON_HEX)
                .setIssuedAt(Instant.now())
                .claim("cardIdentificationNumber", "01546874681748308641061")
                .claim("insurerAccessToken", "")
                .setFooter(PUBLIC_KEY_OF_INSURER_HEX)
                .setPrivateKey(privateKey)
                .compact();
```

```
        System.out.println(vekaToken);

        dev.paseto.jpaseto.Paseto decoded = Pasetos.parserBuilder().setKeyResolver(new
KeyResolverAdapter() {
            @Override
            public PublicKey resolvePublicKey(Version version, Purpose purpose, FooterClaims footer) {
                return getPublicKeyFromECBigIntAndCurve(footer.value());
            }
        }).build().parse(vekaToken);
        System.out.println(decoded.getClaims().get("cardIdentificationNumber", String.class));

    }

    public static byte[] decode(String s) {
        try {
            return org.apache.commons.codec.binary.Hex.decodeHex(s.toCharArray());
        } catch (DecoderException e) {
            e.printStackTrace();
        }
        return new byte[0];
    }

    private static PrivateKey getPrivateKeyFromECBigIntAndCurve(final String keyHexString) {
        Security.addProvider(new BouncyCastleProvider());
        var privateKeyBytes = Arrays.copyOfRange(decode(keyHexString), 0, 32);
        try {
            var privKeyInfo = new PrivateKeyInfo(new
AlgorithmIdentifier(EdECObjectIdentifiers.id_Ed25519),
                    new DEROctetString(privateKeyBytes));
            final KeyFactory keyFactory = KeyFactory.getInstance("Ed25519");
            var pkcs8KeySpec = new PKCS8EncodedKeySpec(privKeyInfo.getEncoded());
            return keyFactory.generatePrivate(pkcs8KeySpec);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    private static PublicKey getPublicKeyFromECBigIntAndCurve(final String keyHexString) {
        Security.addProvider(new BouncyCastleProvider());
        var publicKeyBytes = decode(keyHexString);
        try {
            var keyFactory = KeyFactory.getInstance("Ed25519");
            var pubKeyInfo = new SubjectPublicKeyInfo(new
AlgorithmIdentifier(EdECObjectIdentifiers.id_Ed25519),
```

```java
                publicKeyBytes);
            var x509KeySpec = new X509EncodedKeySpec(pubKeyInfo.getEncoded());
            return keyFactory.generatePublic(x509KeySpec);

        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    // Java JDK 15 would work without BouncyCastle should be released in september
    // 2020

    /*
     * private static PrivateKey getPrivateKeyFromECBigIntAndCurve(final String
     * keyHexString) {
     * NamedParameterSpec paramSpec = new NamedParameterSpec("Ed25519");
     * final EdECPrivateKeySpec privateKeySpec = new EdECPrivateKeySpec(paramSpec,
     * Arrays.copyOfRange(decode(keyHexString),0,32));
     * try {
     * final KeyFactory keyFactory = KeyFactory.getInstance("Ed25519");
     * return keyFactory.generatePrivate(privateKeySpec);
     * } catch (Exception e) {
     * e.printStackTrace();
     * return null;
     * }
     * }
     */
}
```