**TUTORIAL DE WALT-ID SSIKIT**

[1. INTRODUÇÃO 1](#_Toc171719553)

[2. Setup inicial 2](#_Toc171719554)

[3. Running locally 4](#_Toc171719555)

[4. Running with docker 9](#_Toc171719556)

[5. executando as Funcionalidades 10](#_Toc171719557)

[RESUMINDO 17](#_Toc171719558)

# INTRODUÇÃO

Este tutorial relata como usar o walt-id ssikit em ambiente Windows. Esta ferramenta foi desenvolvida para rodar em ambiente Linux, por isso iniciaremos descrevendo como instalar Ubuntu com o Windows Subsystem for Linux (WSL).

Na seção 3 “Running locally”, tenha em mente que a solução proposta para as falhas nos testes é inadequada. Além disso, o próprio ssi-kit foi recentemente descontinuado, sendo substituído por outras ferramentas, como explicado na página: <https://docs.walt.id/v/ssikit/ssi-kit/transition-to-the-community-stack>. Como trabalho futuro será realizado um novo tutorial para o uso das ferramentas mais atuais.

Aqui trataremos das funções mais básicas desta ferramenta, a geração de chaves criptográficas, de DIDs (Identificadores Descentralizados), emissão e verificação de VCs (Credenciais Verificáveis) e VPs (Apresentações Verificáveis).

Há diversas vantagens no uso de DIDs, VCs e VPs, principalmente nos quesitos segurança, privacidade e automatização de processos. Esses assuntos e conceitos fundamentais a respeito estão fora do escopo deste tutorial. Esses assuntos são tratados nas seguintes especificações:

<https://www.w3.org/TR/did-core/>

<https://www.w3.org/TR/vc-data-model-2.0/>

Este tutorial teve as seguintes referências:

<https://youtu.be/wJ-fQBSCwbA?si=uJlf22TKlRkJ4n09>

<https://docs.walt.id/v/ssikit/getting-started/cli-command-line-interface>

<https://github.com/walt-id/waltid-ssikit>

# Setup inicial

Rodar o comando “wsl --install” no CMD ou PowerShell. Após rodar este comando o Ubuntu será automaticamente instalado e você deve definir seu usuário e senha.

Instale o java sdk 19 com os comandos:

* sudo apt-get update
* sudo apt upgrade -y
* sudo apt install openjdk-19-jre-headless

Cheque a instalação:

* java --version

Instale o SDKMAN com os seguintes comandos:

* sudo apt install unzip
* sudo apt install zip
* curl -s "https://get.sdkman.io" | bash
* source "$HOME/.sdkman/bin/sdkman-init.sh"

Cheque a instalação:

* sdk version

Instale o gradle 8.8 com os seguintes comandos:

* sdk install gradle 8.8

Cheque a instalação:

* gradle --version

Instale o Docker com os seguintes comandos:

* sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
* curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
* sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"
* sudo apt update
* sudo apt install docker-ce -y

Escolha um diretório adequado para colocar o projeto ssikit, utilize os comandos:

* git clone https://github.com/walt-id/waltid-ssikit.git
* cd waltid-ssikit/

# Running locally

No diretório “waltid-ssikit”, abra o arquivo “build.gradle.kts”. Busque por “languageVersion”, esta propriedade estará setada como 17, troque para a sua versão do java, no caso a 19. Abaixo, faça a mesma coisa na propriedade “kotlinOptions.jvmTarget”, como ilustrado na Figura 1:

Figura 1 - Alteração no arquivo "build.gradle.kts"

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Salve o arquivo. O próximo passo é rodar o comando “./ssikit.sh build”, porém se rodar agora ocorreram falhas durante a fase de teste que impedirão o término da build, como mostrado na Figura 2.

Figura 2 - Testes falhos

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

A (péssima) solução que encontrei foi comentar os trechos de código apontados como falhos. Não comente o teste por completo porque haverá não dará certo, comente somente o código dentro do teste. Desta forma, mesmo não podendo utilizar o cheqd e a StatusList2021Service para lidar com revogações, poderá utilizar todas as demais funcionalidades. Existem 3 arquivos a serem editados no projeto:

* src\test\kotlin\id\walt\ecosystems\CheqdTest.kt (Figuras 3 e 4);
* src\test\kotlin\id\walt\services\did\DidCheqdTest.kt (Figura 5);
* src\test\kotlin\id\walt\signatory\revocation\StatusList2021ServiceTest.kt (Figura 6).

Figura 3 - Arquivo CheqdTest.kt parte 1

Tela de computador

Descrição gerada automaticamente

Figura 4 - Arquivo CheqdTest.kt parte 2

Tela de computador

Descrição gerada automaticamente

Figura 5 - Arquivo DidCheqdTest.kt

Tela de computador

Descrição gerada automaticamente

Figura 6 - Arquivo StatusList2021ServiceTest.kt

Tela de computador

Descrição gerada automaticamente

Salve o arquivo e rode o comando:

* ./ssikit.sh build

Aguarde, este comando pode demorar. Além disso, a build ainda pode falhar, ocorrendo a exceção “Exception in thread "main" java.io.IOException: Downloading from https://services.gradle.org/distributions/gradle-8.0.2-bin.zip failed: timeout”. Neste caso apenas rode o comando novamente e a build deve avançar. Finalizado o comando, um diretório chamado “data” será criado. Nele serão armazenados os arquivos gerados pelas funcionalidades do SSIKIT (DIDs, VCs, keys e etc).

# Running with docker

Para rodar o SSIKIT com o Docker execute os comandos a seguir:

* sudo docker pull waltid/ssikit
* sudo docker container run -p 7000-7004:7000-7004 -itv $(pwd)/data:/app/data docker.io/waltid/ssikit

Toda vez que for executar uma funcionalidade do SSIKIT deve-se executar este último comando acrescido da palavra-chave designada para a funcionalidade acompanhada de seus parâmetros.

# executando as Funcionalidades

Você pode optar entre a versão local ou Docker do ssikit. Aqui utilizaremos a versão local por simplicidade. Rodando o comando “./ssikit.sh --help” você observará as funcionalidades:

key: Gerenciamento de chaves;

did: Identificadores Decentralizados (DIDs);

vc: Credenciais Verificáveis (VCs);

essif: operações especificas ESSIF;

gaiax: operações especificas Gaia-X;

oidc: OIDC para apresentações verificáveis e emissão de credenciais; e

serve: Roda o walt.id ssikit como um serviço RESTful.

Sobre a geração de chaves, rodando o comando “./ssikit.sh key gen --help” você encontra os seguintes algoritmos de geração de chaves disponíveis: Ed25519; Secp256k1; RSA; Secp256r1. O algoritmo padrão é o Ed25519, para definir outro basta usar o parâmetro “--algorithm" ou “-a”, como no exemplo:

* ./ssikit.sh key gen --algorithm Secp256k1

Retornando o resultado:

Generating Secp256k1 key pair...

Key "a38d286571614dda91de81aca0a43e79" generated.

Executando o comando a seguir serão listadas todas as Keys já criadas:

* ./ssikit.sh key list

Retornando como exemplo:

Listing keys ...

Results:

- 1: "a38d286571614dda91de81aca0a43e79" (Algorithm: "ECDSA\_Secp256k1", provided by "SUN")

Agora vamos criar uma DID. Para o comando “did” existem as seguintes funcionalidades:

create Criação de DID

resolve Resolução de DID

list Listagem de DIDs

import Importação de DID para uma custodian store

delete Deletar DID em uma custodian store

O comando “did create” tem como parâmetro obrigatório “--did-method” ou “-m”, que aceita os seguintes valores: “key”, “web”, “ebsi”, “iota”, “jwk” e “cheqd”. Como já criamos uma chave criptográfica, utilizaremos ela para definir o id da did que criaremos, utilizando o parâmetro “--key” ou “-k”:

* ./ssikit.sh did create --did-method ebsi --key a38d286571614dda91de81aca0a43e79

Neste exemplo, a DID criada foi:

DID created: did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv

DID document (below, JSON):

{

"assertionMethod" : [ "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv#a38d286571614dda91de81aca0a43e79"

],

"authentication" : [ "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv#a38d286571614dda91de81aca0a43e79"

],

"@context" : "https://www.w3.org/ns/did/v1",

"id" : "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv",

"verificationMethod" : [

{

"controller" : "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv",

"id" : "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv#a38d286571614dda91de81aca0a43e79",

"publicKeyJwk" : {

"alg" : "ES256K",

"crv" : "secp256k1",

"kid" : "a38d286571614dda91de81aca0a43e79",

"kty" : "EC",

"use" : "sig",

"x" : "4GX07DjvymVXdU4G1Jm3zlHqXOdvTM-11esv6V8MAZ4",

"y" : "ohu1dj1qOMJscMnlcgbt3GUoLQIT4MbEpGLaQHe62nA"

},

"type" : "EcdsaSecp256k1VerificationKey2019"

}

]

}

O próximo passo é criar uma VC. Para tanto, é necessário ter duas DIDs, uma fará o papel de emissor (issuer) da credencial e a outra de titular (holder) da credencial. Neste exemplo criaremos a próxima DID com o comando:

* ./ssikit.sh did create --did-method key

A DID resultante foi:

“did:key:z6Mkga7xahVky5PuoTzaP9g4PVVFC2EhdYoYuxcQMBCYd7an”

As funcionalidades disponíveis para o comando “vc” são:

* issue: Emite e salva uma VC;
* present: Apresenta uma VC;
* verify: Verifica uma VC ou uma VP (Apresentação Verificável);
* list: Lista as VCs existentes;
* policies: Gerencia as políticas de verificação;
* templates: Gerencia os templates de VC;
* import: Importa VCs para uma custodian store;
* revocation: Lida com revogações de VC;
* parse: Analisa VCs com representações JWT ou SD-JWT e retorna um JSON body;

Os parâmetros obrigatórios para a emissão de uma credencial verificável são:

Template: “-t” ou “--template=<string>”;

DID do emissor (issuer): “-i” ou “--issuer-did=<string>”;

DID do titular/assunto (holder): “-s” ou “--subject-did=<string>”.

O template se trata da estrutura do arquivo para uma credencial específica, por exemplo: certificados de vacinação, diplomas, atestados, etc. Executando o comando “./ssikit.sh vc templates list” será retornada uma lista com todos os templates disponíveis. Também é possível criar seus próprios templates e importá-los com o comando “./ssikit.sh vc templates import”. Aqui utilizaremos o template “VerifiableVaccinationCertificate”. Rode o comando:

* ./ssikit.sh vc issue --template=VerifiableVaccinationCertificate --issuer-did=did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv --subject-did=did:key:z6Mkga7xahVky5PuoTzaP9g4PVVFC2EhdYoYuxcQMBCYd7an

Será retornada a VC como o conteúdo de um arquivo JSON:

{

"type" : [ "VerifiableCredential", "VerifiableAttestation", "VerifiableVaccinationCertificate" ],

"@context" : [ "https://www.w3.org/2018/credentials/v1", "https://w3id.org/security/suites/jws-2020/v1" ],

"id" : "urn:uuid:0226763a-1327-40f3-b03d-7c8ce8bf11bb",

"issuer" : "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv",

"issuanceDate" : "2024-07-11T16:15:09Z",

"issued" : "2024-07-11T16:15:09Z",

"validFrom" : "2024-07-11T16:15:09Z",

"expirationDate" : "2022-08-31T00:00:00Z",

"credentialSchema" : {

"id" : "https://raw.githubusercontent.com/walt-id/waltid-ssikit-vclib/master/src/test/resources/schemas/VerifiableVaccinationCertificate.json",

"type" : "JsonSchemaValidator2018"

},

"credentialSubject" : {

"id" : "did:key:z6Mkga7xahVky5PuoTzaP9g4PVVFC2EhdYoYuxcQMBCYd7an",

"dateOfBirth" : "1993-04-08",

"familyName" : "DOE",

"givenNames" : "Jane",

"personIdentifier" : "optional The type of identifier and identifier of the person, according to the policies applicable in each country. Examples are citizen ID and/or document number (ID- card/passport) or identifier within the health system/IIS/e-registry.",

"personSex" : "optional",

"uniqueCertificateIdentifier" : "UVCI0904008084H",

"vaccinationProphylaxisInformation" : [ {

"administeringCentre" : "Name/code of administering centre or a health authority responsible for the vaccination event",

"batchNumber" : "optional 1234",

"countryOfVaccination" : "DE",

"dateOfVaccination" : "2021-02-12",

"diseaseOrAgentTargeted" : {

"code" : "840539006",

"system" : "2.16.840.1.113883. 6.96",

"version" : "2021-01-31"

},

"doseNumber" : "1",

"marketingAuthorizationHolder" : "Example Vaccine Manufacturing Company",

"nextVaccinationDate" : "optional - 2021-03-28",

"totalSeriesOfDoses" : "2",

"vaccineMedicinalProduct" : "VACCINE concentrate for dispersion for injection",

"vaccineOrProphylaxis" : "1119349007 COVID-19 example vaccine"

} ]

},

"credentialStatus" : {

"id" : "https://essif.europa.eu/status/covidvaccination#392ac7f6-399a-437b-a268-4691ead8f176",

"type" : "CredentialStatusList2020"

},

"evidence" : {

"documentPresence" : [ "Physical" ],

"evidenceDocument" : [ "Passport" ],

"id" : "https://essif.europa.eu/tsr-va/evidence/f2aeec97-fc0d-42bf-8ca7-0548192d5678",

"subjectPresence" : "Physical",

"type" : [ "DocumentVerification" ],

"verifier" : "did:ebsi:2962fb784df61baa267c8132497539f8c674b37c1244a7a"

},

"proof" : {

"type" : "JsonWebSignature2020",

"creator" : "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv",

"created" : "2024-07-11T16:15:09Z",

"proofPurpose" : "assertionMethod",

"verificationMethod" : "did:ebsi:zhPFWeXCWrnDC9UBSE3GVnv#a38d286571614dda91de81aca0a43e79",

"jws" : "eyJiNjQiOmZhbHNlLCJjcml0IjpbImI2NCJdLCJhbGciOiJFUzI1NksifQ..sC\_qIidFcT3QewNy8-TYYHM3dG4q0CnLwf9gLlc9fkhpF6eE4eYPB74b8yVnzsSqlBOF0zjjZ9BO3tZP2mRvCg"

}

}

Este conteúdo deve ser salvo em um diretório adequado. Por simplicidade apenas rodaremos o seguinte comando “nano vc.json”, colaremos o conteúdo e salvaremos o arquivo. Agora já é possível realizar a verificação com o comando:

* ./ssikit.sh vc verify vc.json

O comando de retornar uma verificação bem-sucedida, com a mensagem:

Results:

SignaturePolicy: passed

Verified: true

Existem outras formas de verificação de credenciais chamadas de políticas. Para listá-las basta rodar o comando “./ssikit.sh vc policies list”. Para checar uma VC de acordo com essas políticas basta usar o parâmetro “--policy=<string>” ou “-p”. Você pode checar múltiplas políticas, como no exemplo abaixo:

* ./ssikit.sh vc verify vc.json --policy=JsonSchemaPolicy -p SignaturePolicy

Results:

JsonSchemaPolicy: passed

SignaturePolicy: passed

Verified: true

Para criar uma VP (Apresentação Verificável) o único parâmetro é obrigatório é o titular (holder): “--holder-did=<string>” ou “-i”. Você pode criar uma VP unicamente por meio do holder, mas pode ser mais interessante incluir um arquivo de VC, como no exemplo de comando abaixo:

* ./ssikit.sh vc present vc.json --holder-did=did:key:z6Mkga7xahVky5PuoTzaP9g4PVVFC2EhdYoYuxcQMBCYd7an

As apresentações geradas podem ser encontradas no diretório “waltid-ssikit\data\vc\presented”. Pode-se verificar a VP da mesma forma que se verifica uma VC, como no comando exemplo:

./ssikit.sh vc verify "data/vc/presented/vp-1720718079074.json"

# RESUMINDO

Neste tutorial, além da instalação do walt-id SSI Kit, foram exemplificadas as funcionalidades:

1. Geração de chaves criptográficas, comando: ssikit key gen --algorithm;
2. Criação de DIDs, comando: ssikit did create --did-method;
3. Emissão de VCs, comando: ssikit vc issue --template=<string> --issuer-did=<string> --subject-did=<string>;
4. Emissão de VPs, comando: ssikit vc present <file.json> --holder-did=<string>;
5. Verificação de VCs/VPs, comando: ssikit.sh vc verify <file.json> --policy=<string>.

A Figura 7 exemplifica o funcionamento da ferramenta de acordo com os atores envolvidos.

Figura 7 - Uso de comandos por ator