Token Sign, Verify, Key Management Design

10 Jan 2023

For keycloak v22

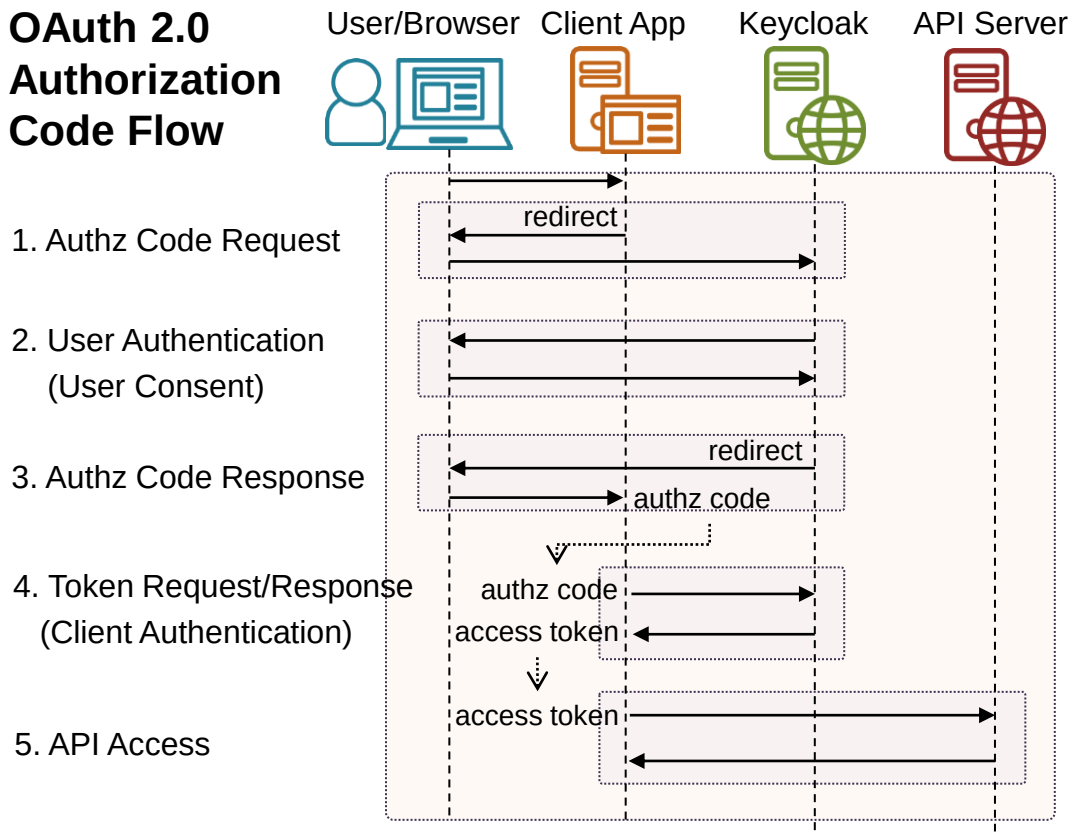
PROPOSED DRAFT

Table of Contents

Existing mechanism

Patterns for signing and verifying a JWT

Patterns for signing and verifying a JWT



Pattern 1: Signed by Keycloak

Pattern 1-a:
- Signer: Keycloak
- Verifier: Keycloak
  - Refresh Token
  - Access Token

Pattern 1-b:
- Signer: Keycloak
- Verifier: Client App
  - ID Token
  - Access Token
  - UserInfo Response
  - Authz Response (JARM)

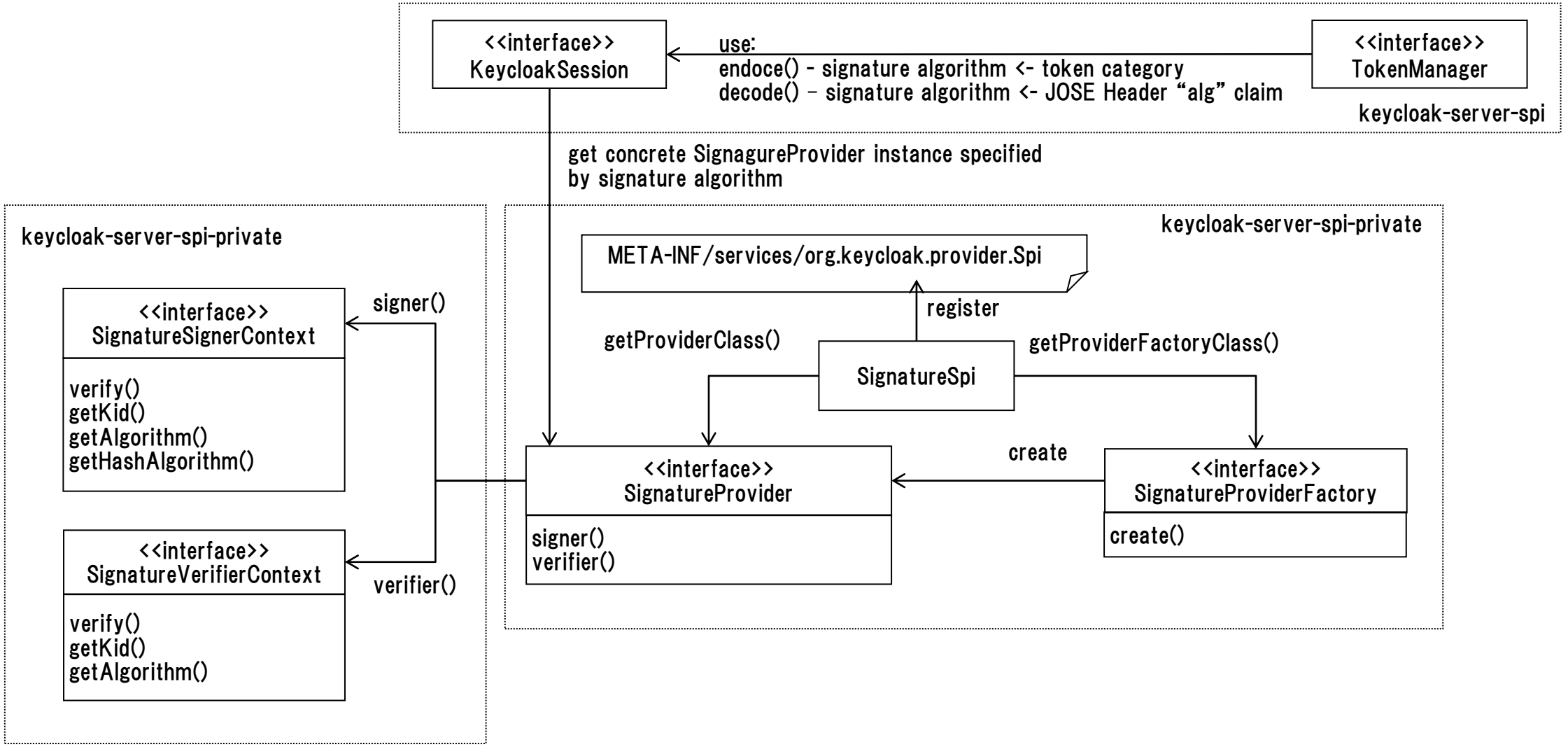
Pattern 1-c:
- Signer: Keycloak
- Verifier: API Server
  - Access Token

Pattern 2: Signed by Client App

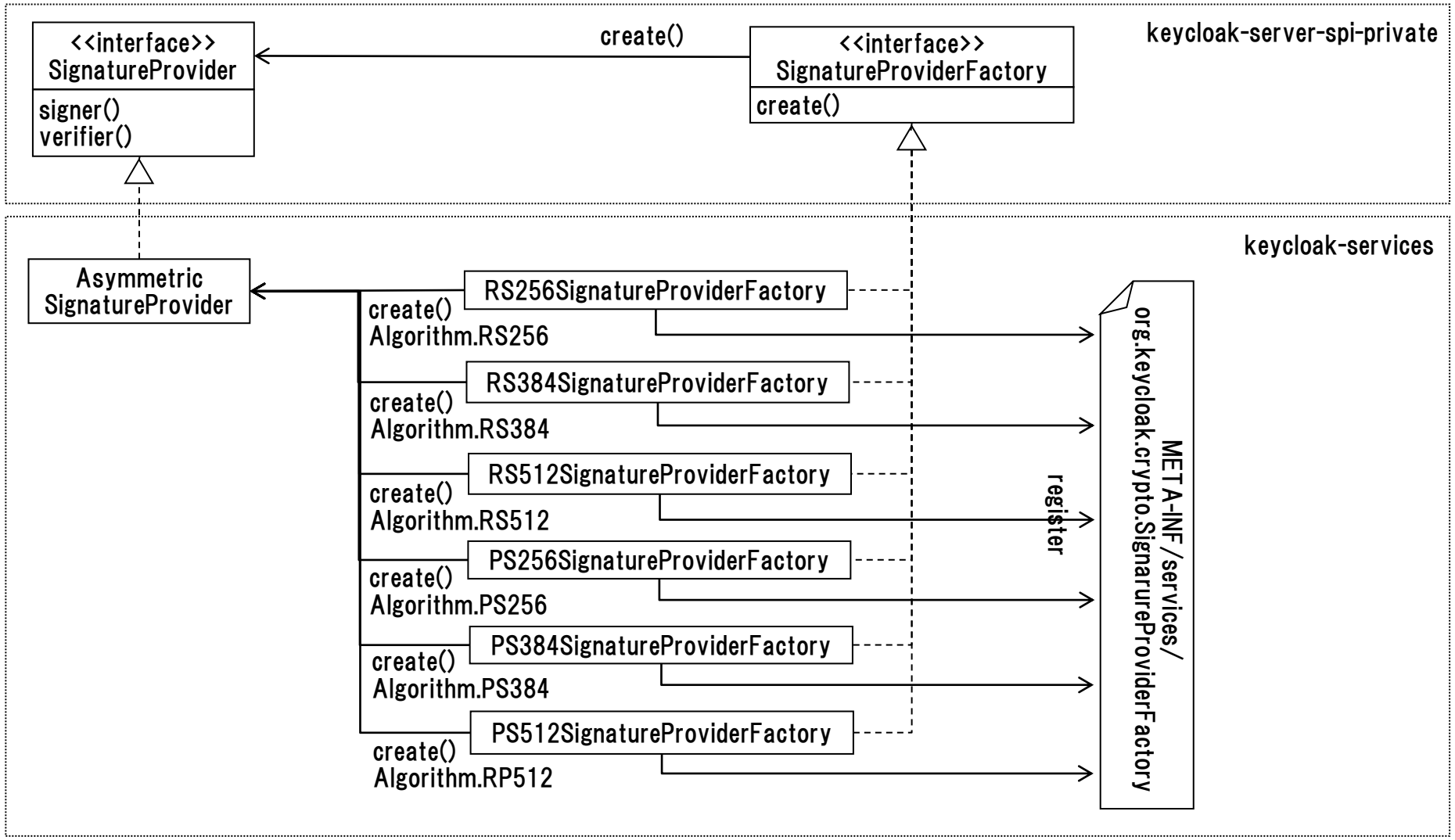
Pattern 2:
- Signer: Client App
- Verifier: Keycloak
  - JWS Client Assertion
  - Request Object/JAR
  - CIBA Backchannel Request

The framework for signing and verifying a JWT by Keycloak

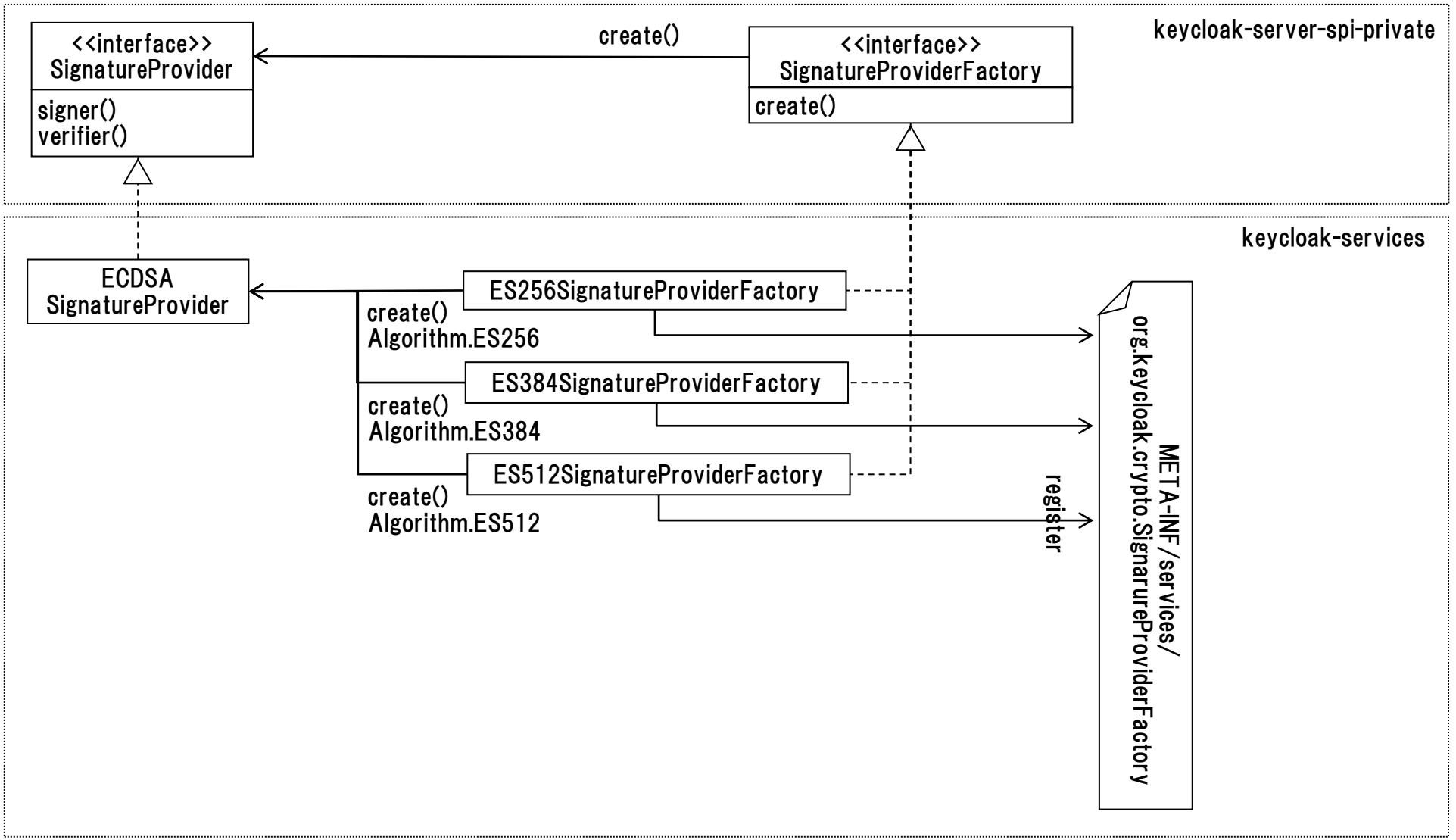
Sign and verify by keycloak: Framework



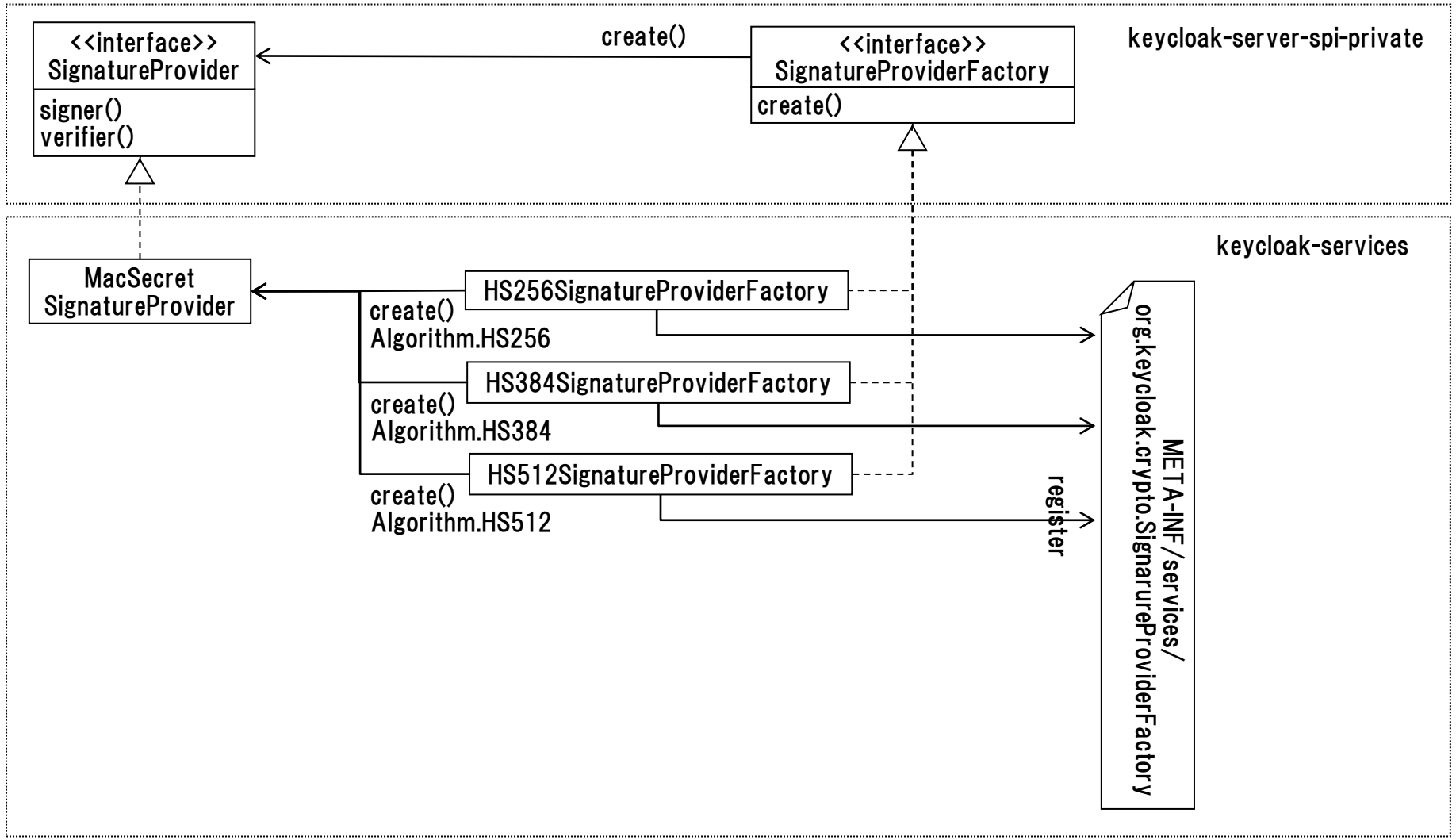
Sign and verify by keycloak: Signature Provider - RSA-SSA



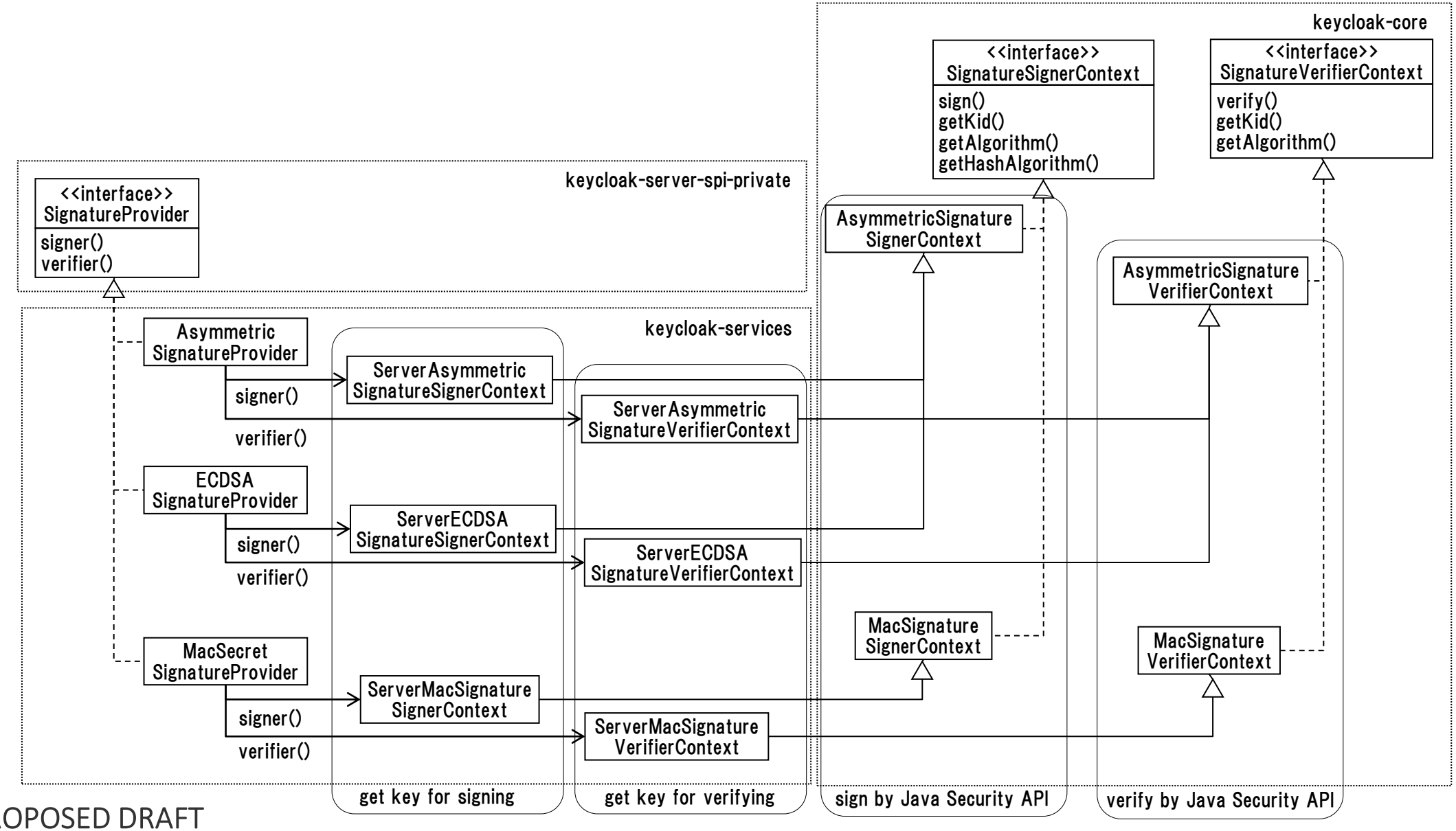
Sign and verify by keycloak: Signature Provider - ECDSA



Sign and verify by keycloak: Signature Provider - HMAC

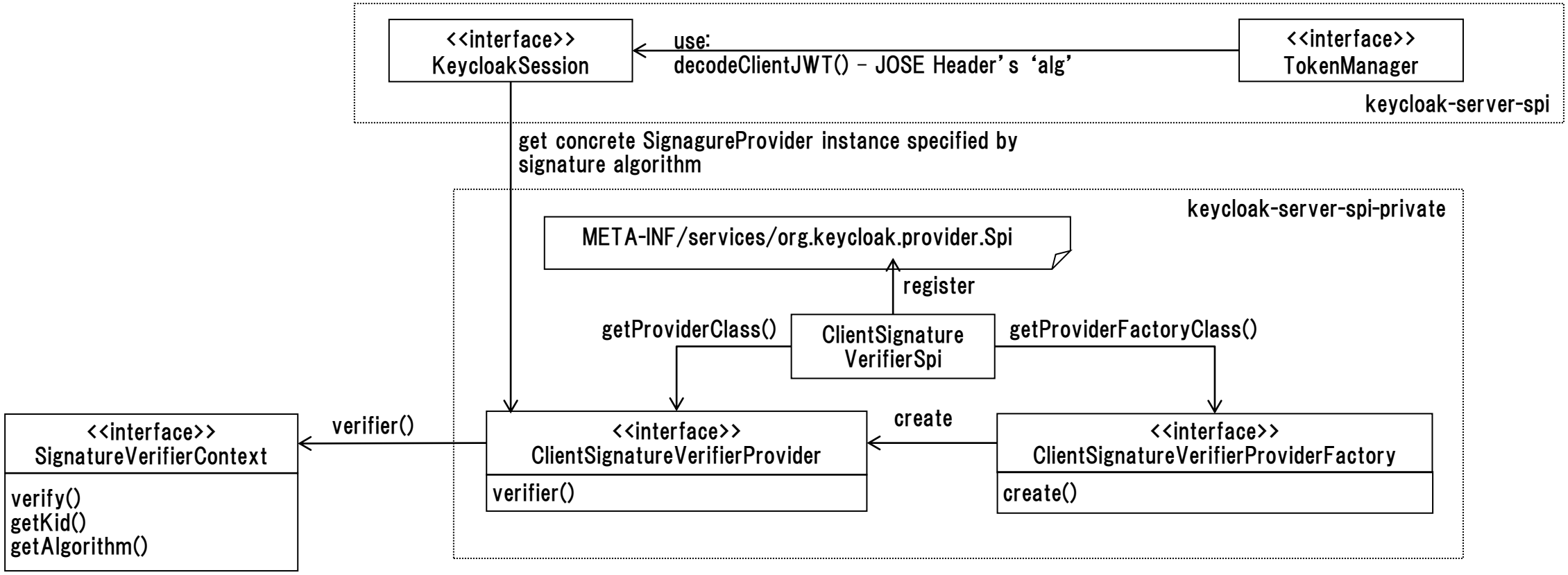


Sign and verify by keycloak: Signature Signer / Verifier Context

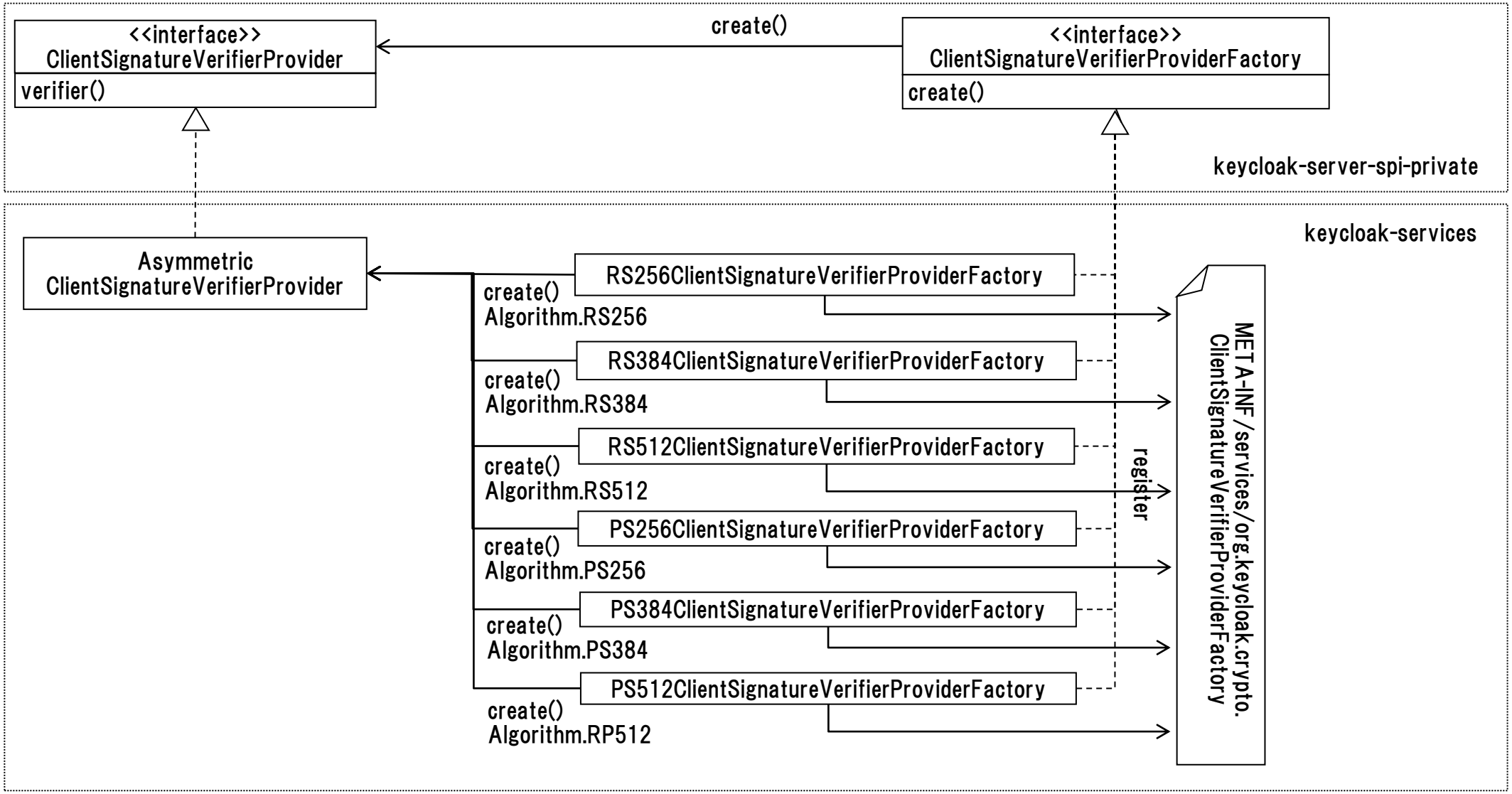


The framework for signing a JWT by Client App, verifying it by Keycloak

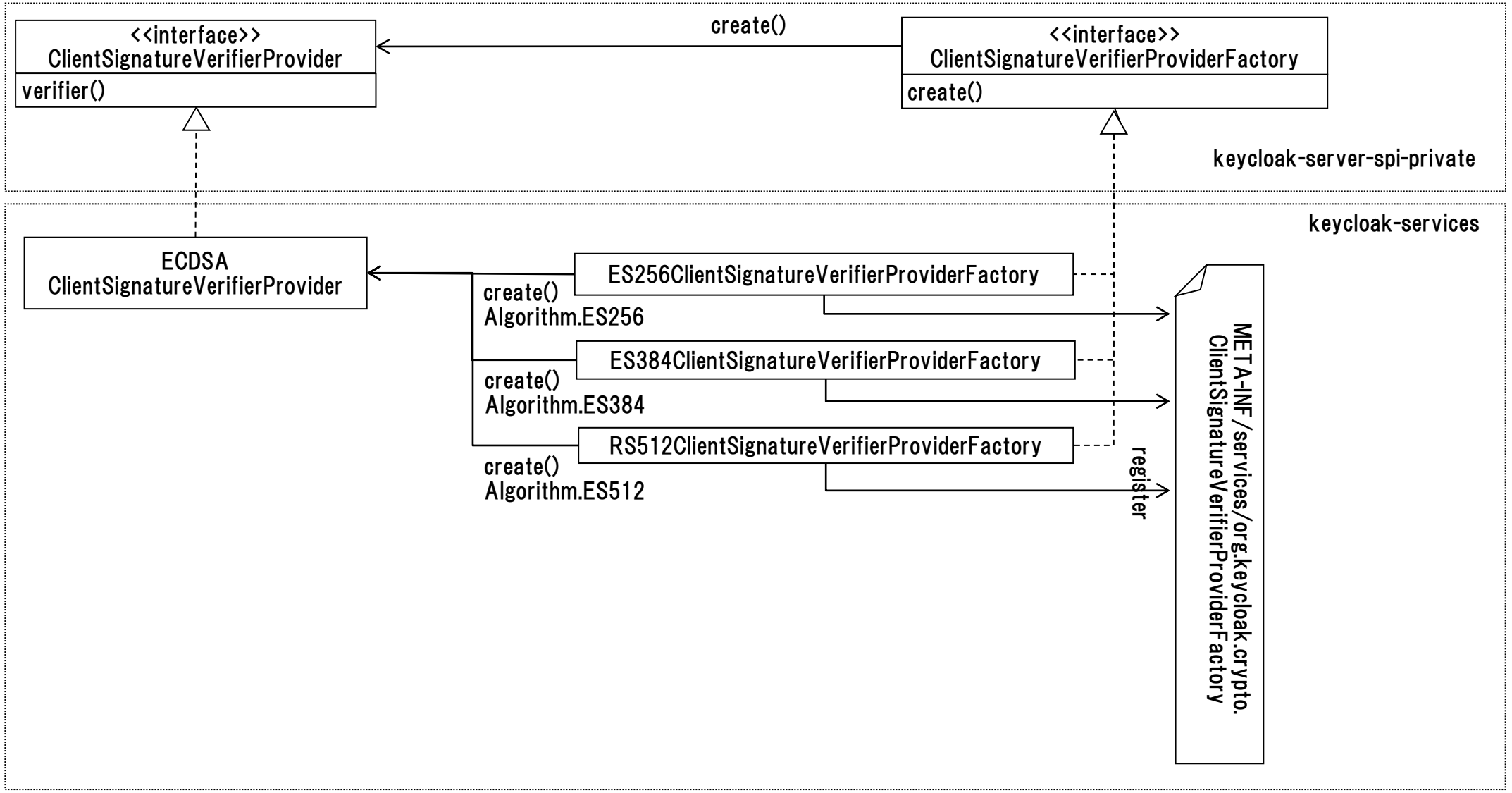
Framework: sign by a client, verify by keycloak



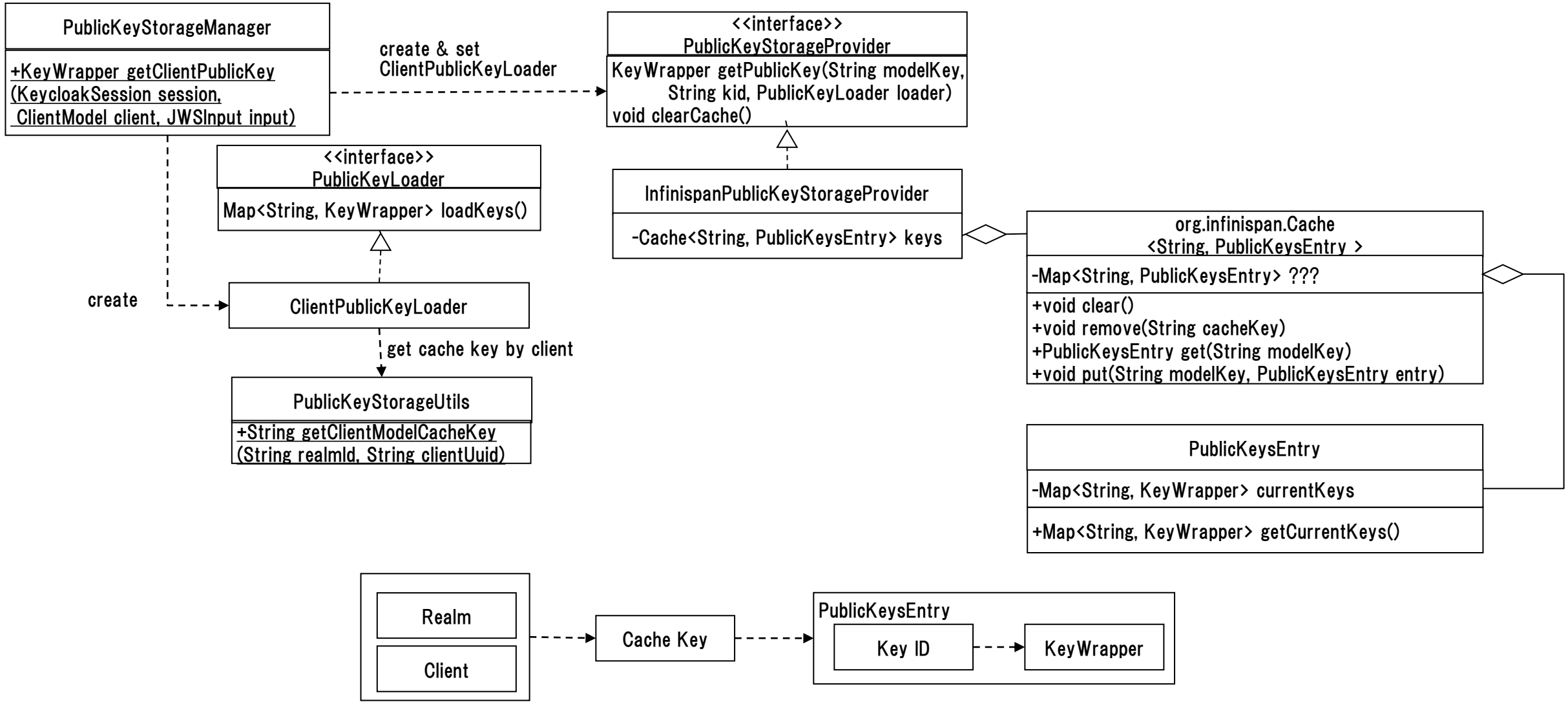
Sign by a client, verify by keycloak: Signature Provider - RSA-SSA



Sign by a client, verify by keycloak: Signature Provider - ECDSA



Client's key management



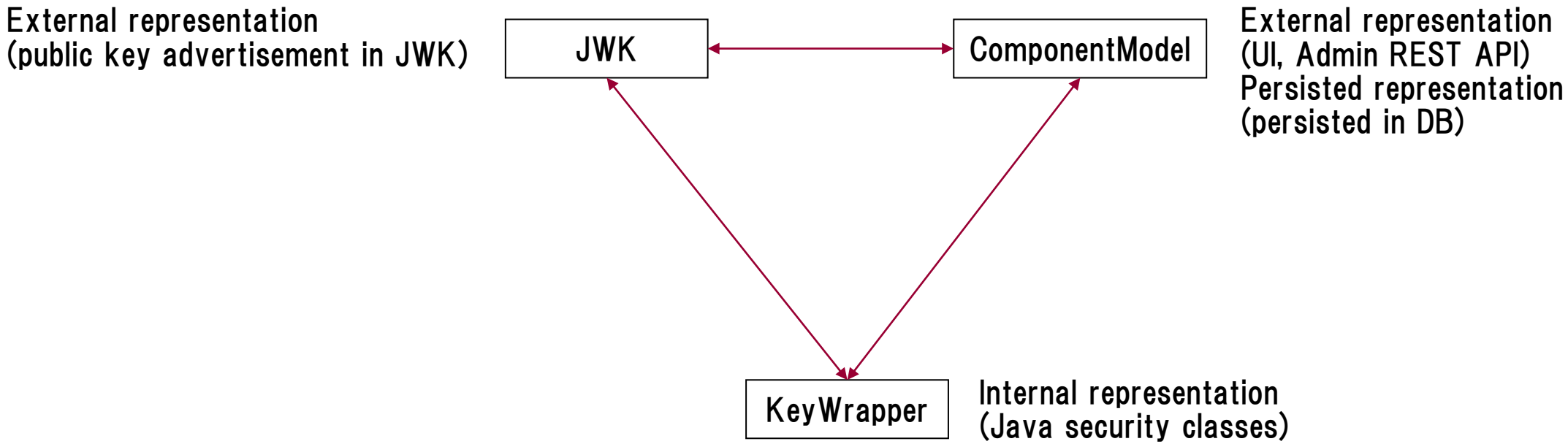
Key

Supported key types

keycloak-core: org.keycloak.crypto.KeyType

#	Description	Field	JWA representation
1	Asymmetric key for RSA based algorithms	RSA	RSA
2	Asymmetric key for elliptic curve-based algorithms	EC	EC
3	Symmetric key	OCT	OCT

Key representation



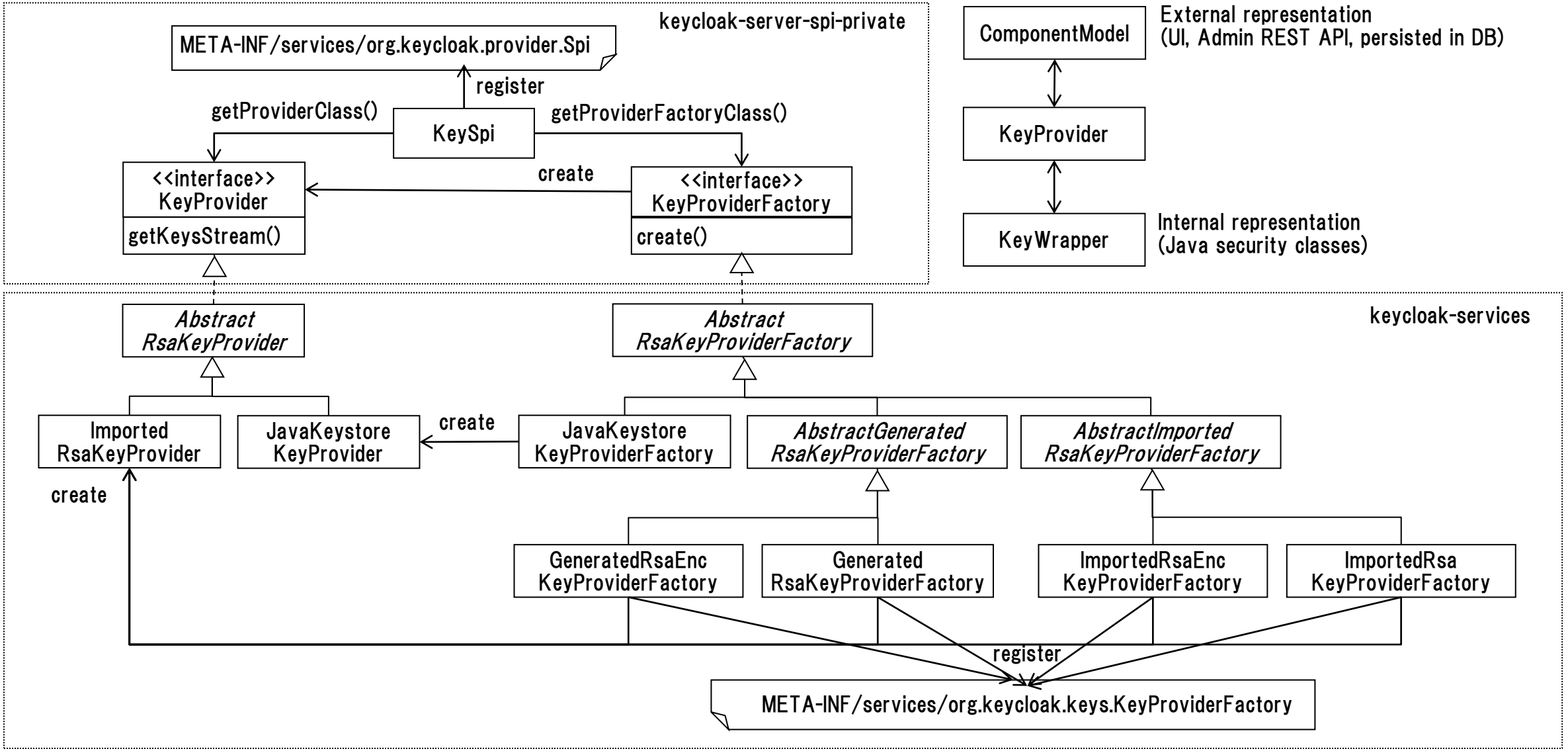
Advertising Keycloak's public keys

[Key representation in JWK]
project: keycloak-core
package: org.keycloak.jose.jwk
class: JWKBuilder

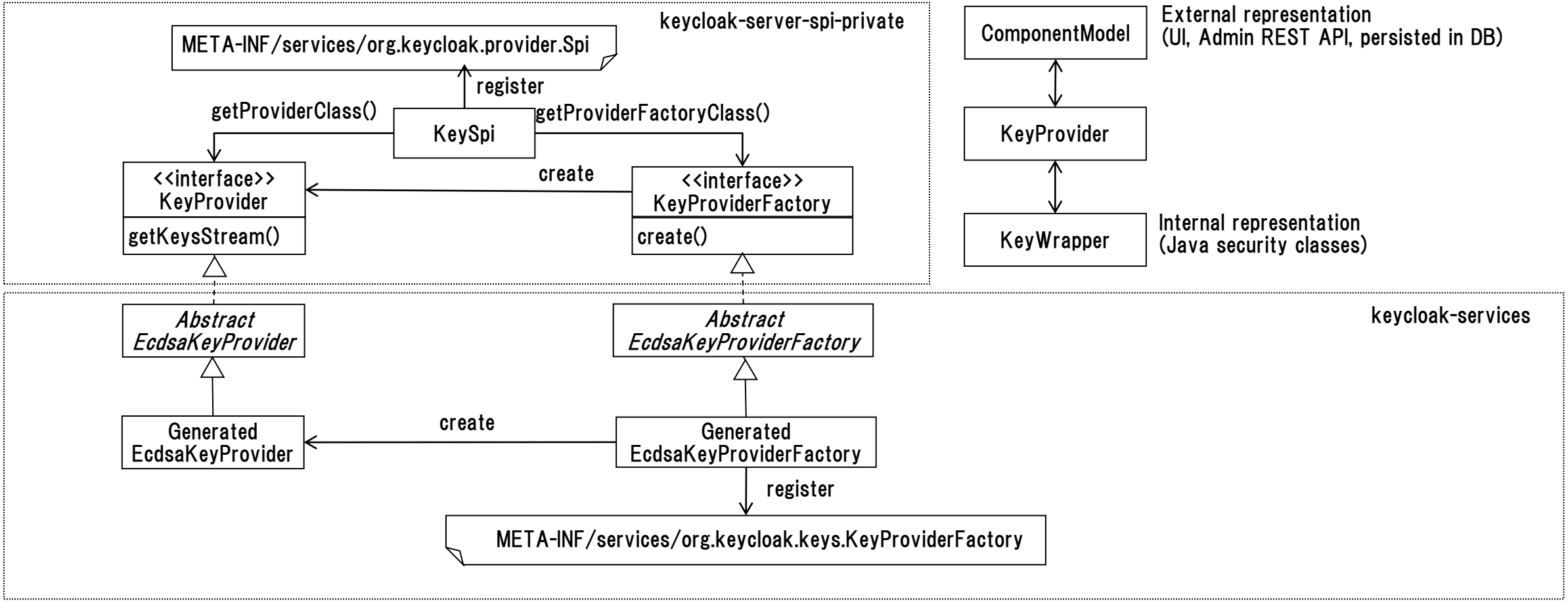
[JSON representation]:
project: keycloak-services
package: org.keycloak.protocol.oidc
class: OIDCLoginProtocolService
method: certs


```
{ "keys":[
 { "kid":"dsf9rL5TGxnxJMiciTb-VVF7oy2CFXRsvdHyTplZRms",
  "kty":"EC",
  "alg":"ES256",
  "use":"sig",
  "crv":"P-256",
  "x":"ri1I124IGmuvVhNvR--bT_MTUKuXnUGOWxUUxty6ZjE",
  "y":"DYAD6RbOUFFRGVdd5p7H1MbFqqOJDBJNvUKkA_gT4iY"},
 { "kid":"9KCv_i09JDKf0we2FbH-q3JTO2kw8omz-YICmhLgyVM",
  "kty":"RSA",
  "alg":"RS256",
  "use":"sig",
  "n":"608u-SfeobdPcij..._NRgLG7d2c",
  "e":"AQAB",
  "x5c":["MIIBkTC..."],
  "x5t":"p5QxeT6TPQ1AKLrVbdUW3BqNx3c",
  "x5t#S256":"OZyzV54QNb5aMoWxGTODA9-hBU2yJYGIzMNCRsdFyY8"
 }]}
```

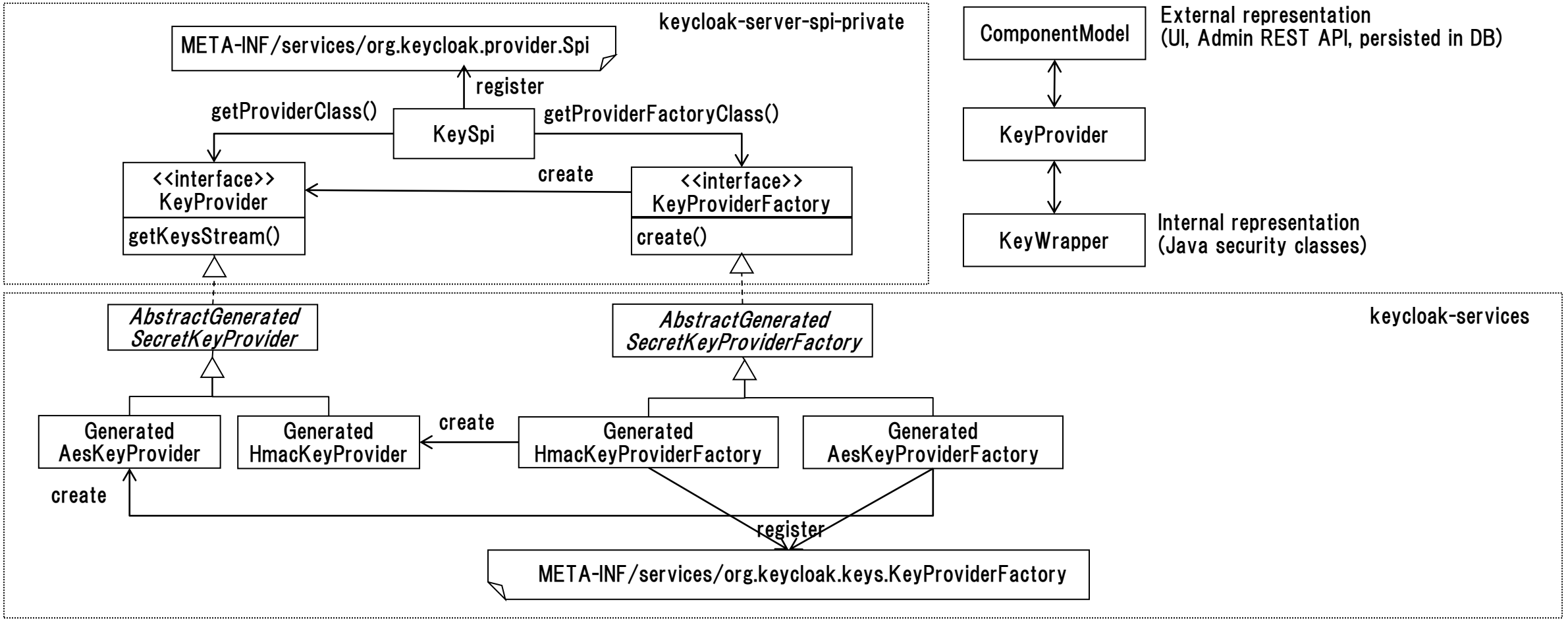
Key type: RSA


keycloak-server-spi-private
META-INF/services/org.keycloak.provider.Spi
register
getProviderClass()
KeySpi
getProviderFactoryClass()
<<interface>>
KeyProvider
getKeysStream()
create
<<interface>>
KeyProviderFactory
create()
ComponentModel
External representation
(UI, Admin REST API, persisted in DB)
KeyProvider
KeyWrapper
Internal representation
(Java security classes)
keycloak-services
Abstract
RsaKeyProvider
Abstract
RsaKeyProviderFactory
Imported
RsaKeyProvider
JavaKeystore
KeyProvider
create
JavaKeystore
KeyProviderFactory
AbstractGenerated
RsaKeyProviderFactory
AbstractImported
RsaKeyProviderFactory
create
GeneratedRsaEnc
KeyProviderFactory
Generated
RsaKeyProviderFactory
ImportedRsaEnc
KeyProviderFactory
ImportedRsa
KeyProviderFactory
register
META-INF/services/org.keycloak.keys.KeyProviderFactory

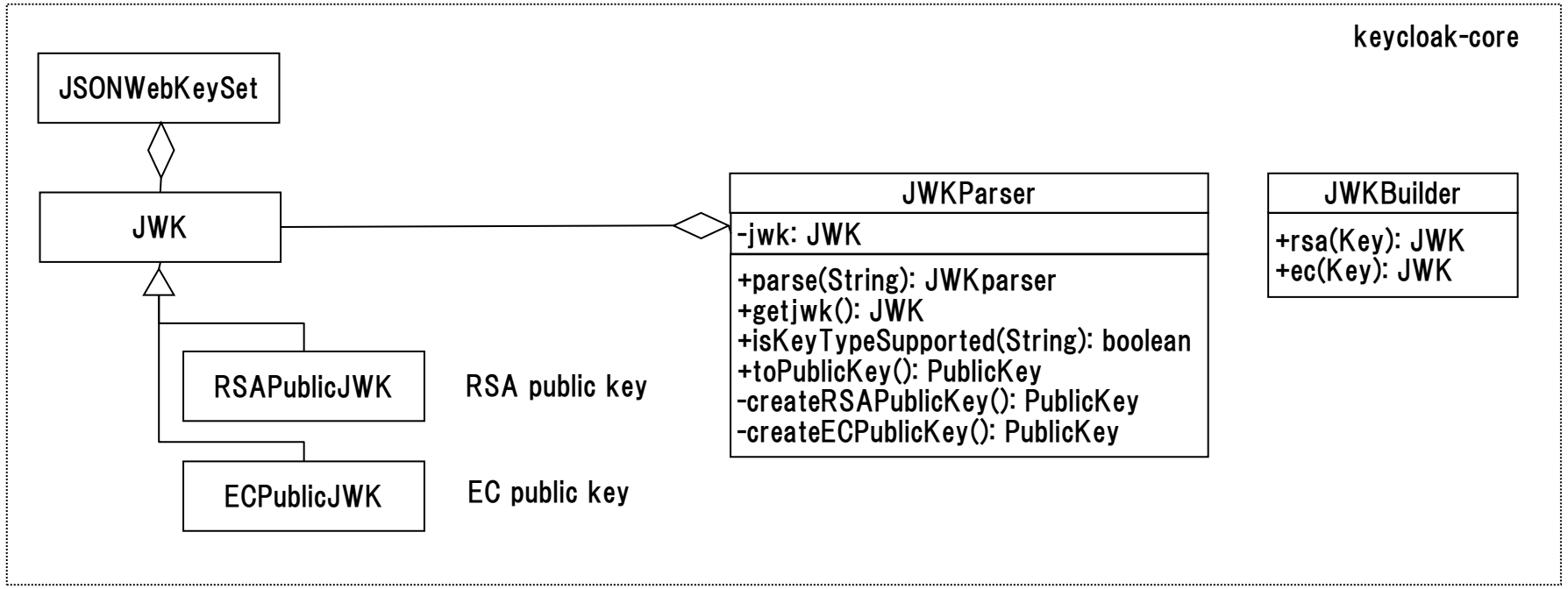
Key type: EC


keycloak-server-spi-private
META-INF/services/org.keycloak.provider.Spi
register
getProviderClass()
KeySpi
getProviderFactoryClass()
<<interface>>
KeyProvider
getKeysStream()
create
<<interface>>
KeyProviderFactory
create()
ComponentModel
External representation
(UI, Admin REST API, persisted in DB)
KeyProvider
KeyWrapper
Internal representation
(Java security classes)
keycloak-services
Abstract
EcdsaKeyProvider
Abstract
EcdsaKeyProviderFactory
Generated
EcdsaKeyProvider
create
Generated
EcdsaKeyProviderFactory
register
META-INF/services/org.keycloak.keys.KeyProviderFactory

Key type: OCT


keycloak-server-spi-private
META-INF/services/org.keycloak.provider.Spi
register
getProviderClass()
KeySpi
getProviderFactoryClass()
<<interface>>
KeyProvider
getKeysStream()
create
<<interface>>
KeyProviderFactory
create()
ComponentModel
External representation
(UI, Admin REST API, persisted in DB)
KeyProvider
KeyWrapper
Internal representation
(Java security classes)
keycloak-services
AbstractGenerated
SecretKeyProvider
AbstractGenerated
SecretKeyProviderFactory
Generated
AesKeyProvider
Generated
HmacKeyProvider
create
Generated
HmacKeyProviderFactory
Generated
AesKeyProviderFactory
create
register
META-INF/services/org.keycloak.keys.KeyProviderFactory

JWK public key representation



Misc

JCA <-> JWA representation conversion

keycloak-core: org.keycloak.crypto.JavaAlgorithm

#	Description	JWA	JCA	Key Type
1	RSASSA-PKCS1-v1_5 using SHA-256	RS256	SHA256withRSA	RSA
2	RSASSA-PKCS1-v1_5 using SHA-384	RS384	SHA384withRSA	RSA
3	RSASSA-PKCS1-v1_5 using SHA-512	RS512	SHA512withRSA	RSA
4	RSASSA-PSS using SHA-256 and MGF1 with SHA-256	PS256	SHA256withRSAandMGF1	RSA
5	RSASSA-PSS using SHA-384 and MGF1 with SHA-384	PS384	SHA384withRSAandMGF1	RSA
6	RSASSA-PSS using SHA-512 and MGF1 with SHA-512	PS512	SHA512withRSAandMGF1	RSA
7	ECDSA using P-256 and SHA-256	ES256	SHA256withECDSA	EC
8	ECDSA using P-384 and SHA-384	ES384	SHA384withECDSA	EC
9	ECDSA using P-521 and SHA-512	ES512	SHA512withECDSA	EC
10	HMAC using SHA-256	HS256	HMACSHA256	OCT
11	HMAC using SHA-384	HS384	HMACSHA384	OCT
12	HMAC using SHA-512	HS512	HMACSHA512	OCT

Initialization when Keycloak booted

keycloak-server-spi-private: org.keycloak.models.utils.DefaultKeyProviders

#	Provider Name	JWA	Key Use
1	rsa-generated	RS256	SIG
2	rsa-enc-generated	RSA-OAEP	ENC
3	hmac-generated	HS256	SIG
4	aes-generated	AES	ENC

Advertising supported signature algorithms by Server Metadata

[Server Metadata]
project: keycloak-services
package: org.keycloak.protocol.oidc
class: OIDCWellKnownProvider

{ ...
"id_token_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512"]
"userinfo_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256 ","PS512","RS512"]
"request_object_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","none"],
"token_endpoint_auth_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512"],
"introspection_endpoint_auth_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512"],
"authorization_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512"],
"revocation_endpoint_auth_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384 ","ES512","PS256","PS512","RS512"],
"backchannel_authentication_request_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384 ","ES512","PS256","PS512","RS512"],
..."}}

PROPOSED DRAFT

Integration tests

Integration tests

- REST API for generating keys

testsuite/integration-arquillian/servers/auth-server/services/testsuite-providers/src/main/java/org/keycloak/testsuite/rest/resource/TestingOIDCEndpointsApplicationResource.java

- Test for generating EC key

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/keys/GeneratedEcdsaKeyProviderTest.java

- Test for server metadata

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/OIDCWellKnownProviderTest.java

Integration tests

- Test for access token

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oauth/AccessTokenTest.java

- Test for OIDC flows

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/flows/AbstractOIDCResponseTypeTest.java

- Test for UserInfo response

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/UserInfoTest

- Test for authorization response (JARM)

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/AuthorizationTokenEncryptionTest.java

Integration tests

- Test for JWT client authentication

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oauth/ClientAuthSignedJWTTest.java

- Test for request object/JAR

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/OIDCAdvancedRequestParamsTest

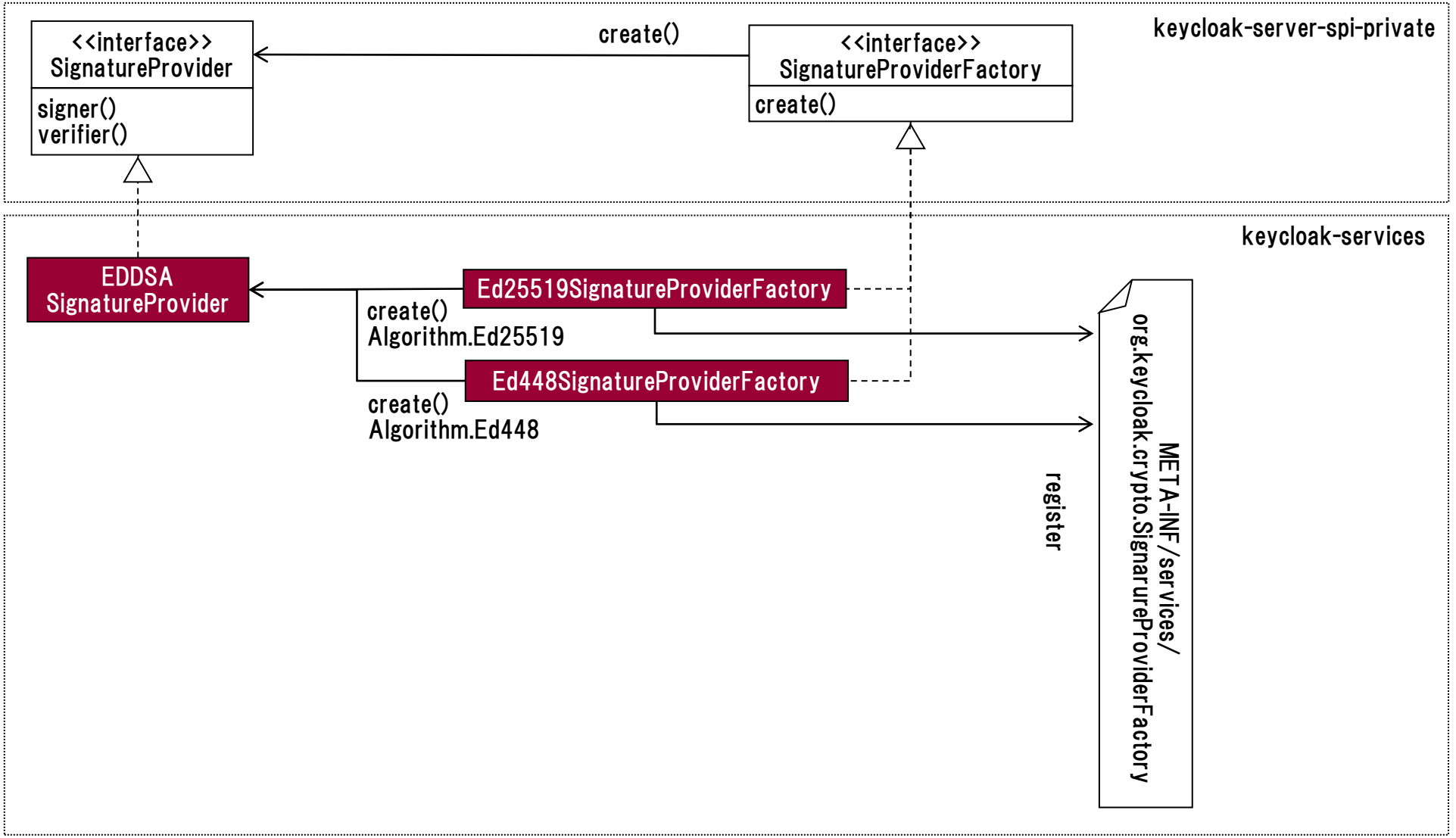
- Test for CIBA authentication request

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/client/CIBATest

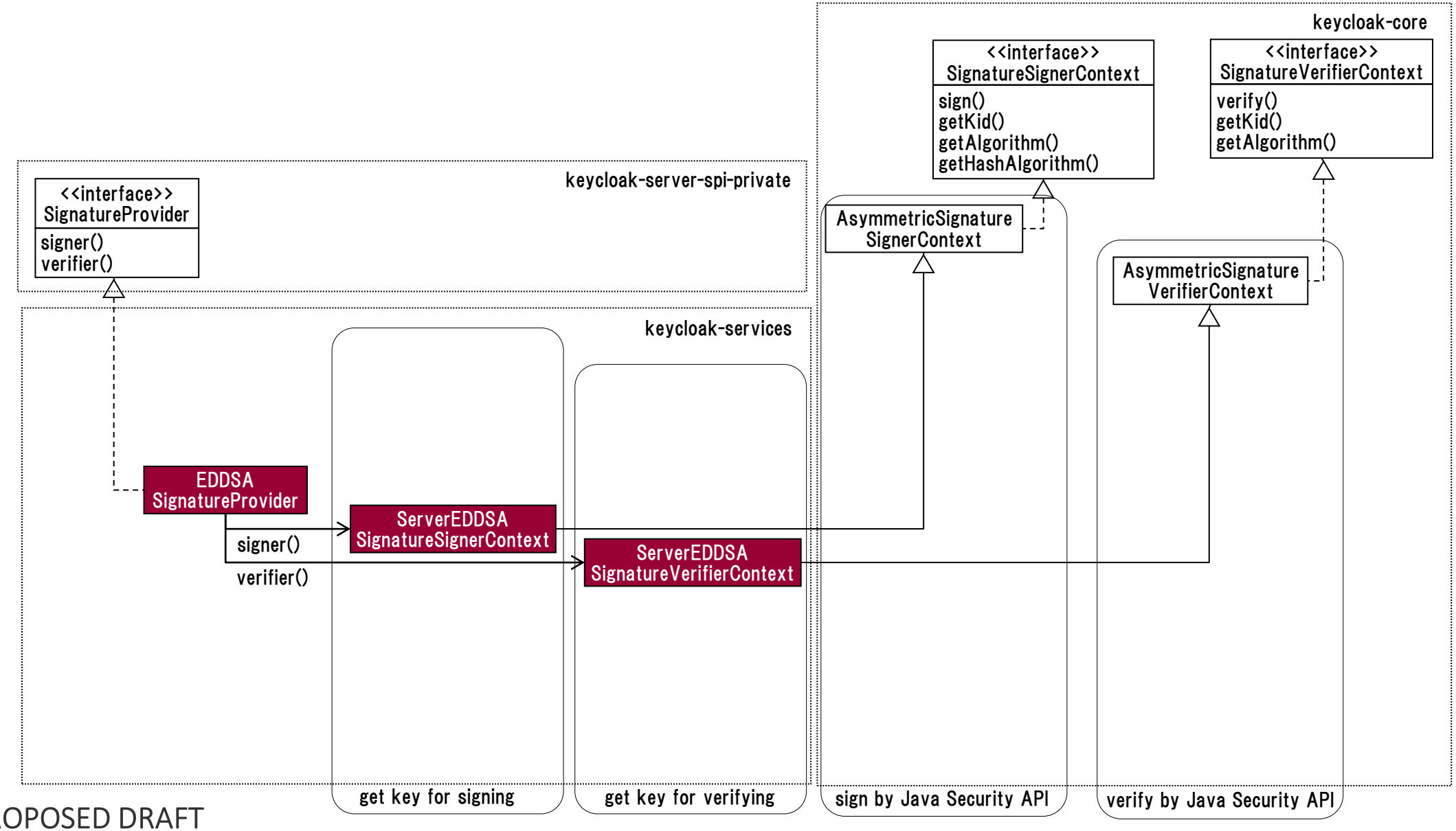
EdDSA support

The framework for signing and verifying a JWT by Keycloak

Sign and verify by keycloak: Signature Provider - EDDSA

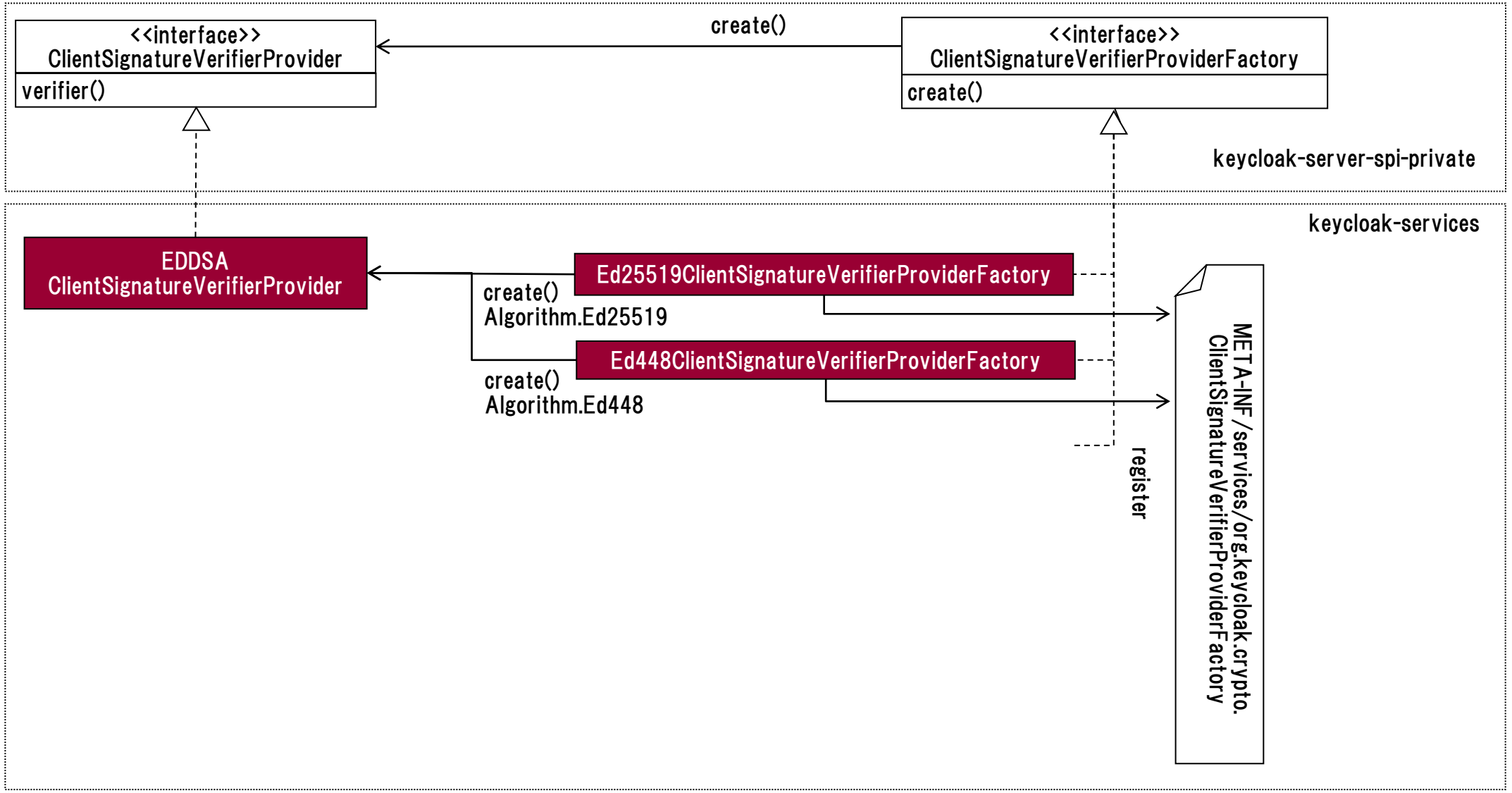


Sign and verify by keycloak: Signature Signer / Verifier Context



The framework for signing a JWT by Client App, verifying it by Keycloak

Sign by a client, verify by keycloak: Signature Provider - EDDSA



Key

Supported key types

keycloak-core: org.keycloak.crypto.KeyType

#	Description	Field	JWA representation
1	Asymmetric key for RSA based algorithms	RSA	RSA
2	Asymmetric key for elliptic curve-based algorithms	EC	EC
3	Symmetric key	OCT	OCT
4	**Octet Key Pair**	**OKP**	**OKP**

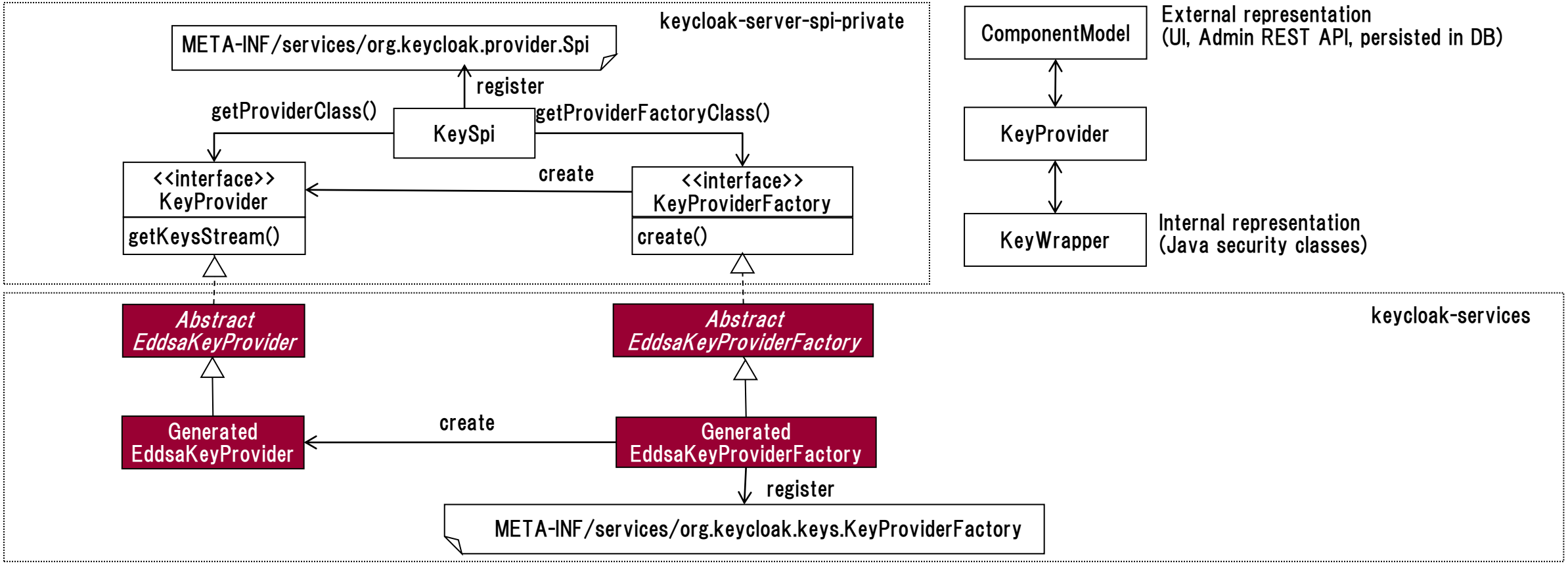
Advertising Keycloak's public keys

[Key representation in JWK]
project: keycloak-core
package: org.keycloak.jose.jwk
class: JWKBuilder

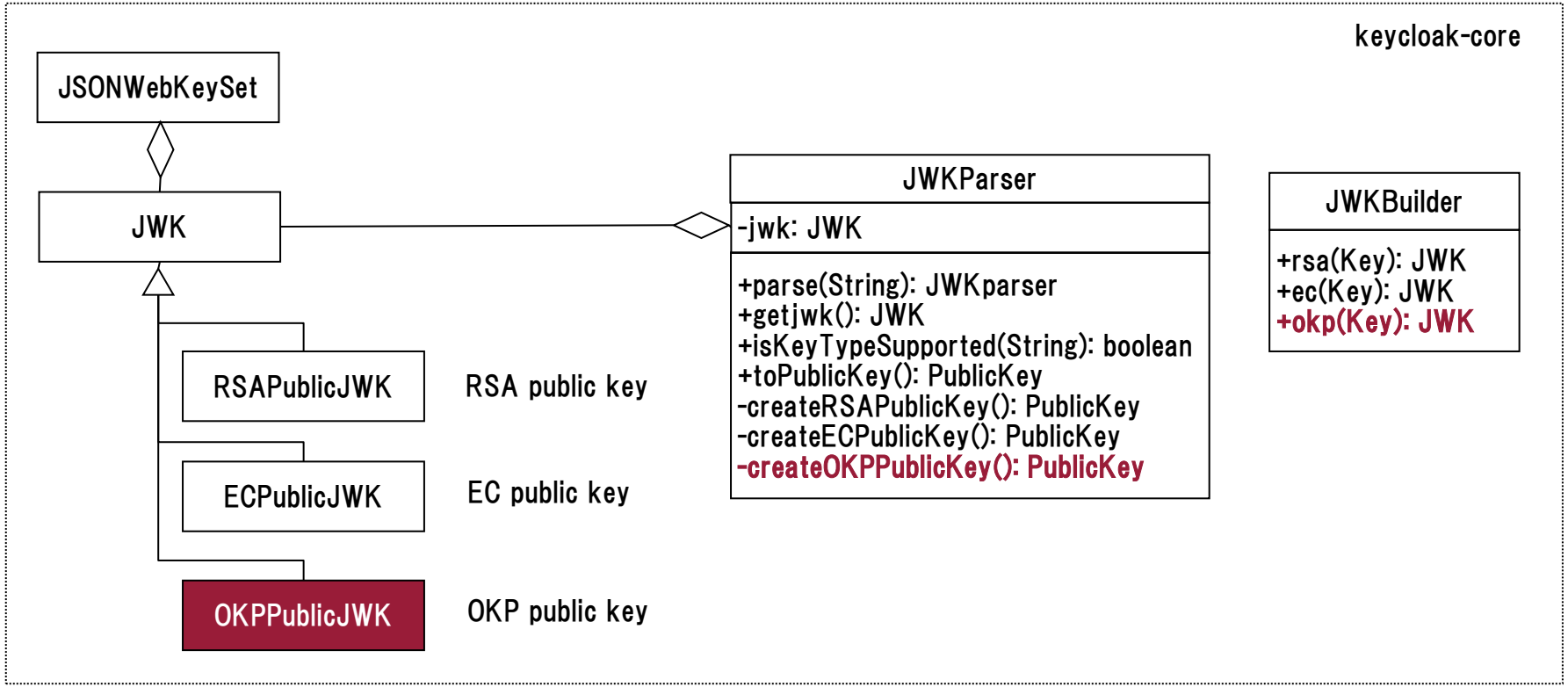
[JSON representation]:
project: keycloak-services
package: org.keycloak.protocol.oidc
class: OIDCLoginProtocolService
method: certs


```
{ "keys":[
  { "kid":"dsf9rL5TGxnxJMiciTb-VVF7oy2CFXRsvdHyTplZRms",
    "kty":“OKP",
    "alg":“Ed25519",
    "use":"sig",
    "crv":"Ed25519",
    "x":"ri1l124lGmuvVhNvR--bT_MTUKuXnUGOWxUUxty6ZjE“
  }
]}
```

Key type: OKP


keycloak-server-spi-private
META-INF/services/org.keycloak.provider.Spi
register
getProviderClass()
KeySpi
getProviderFactoryClass()
<<interface>>
KeyProvider
getKeysStream()
create
<<interface>>
KeyProviderFactory
create()
ComponentModel
External representation
(UI, Admin REST API, persisted in DB)
KeyProvider
KeyWrapper
Internal representation
(Java security classes)
keycloak-services
Abstract
EddsaKeyProvider
Abstract
EddsaKeyProviderFactory
Generated
EddsaKeyProvider
create
Generated
EddsaKeyProviderFactory
register
META-INF/services/org.keycloak.keys.KeyProviderFactory

JWK public key representation



Misc

JCA <-> JWA representation conversion

keycloak-core: org.keycloak.crypto.JavaAlgorithm

#	Description	JWA	JCA	Key Type
1	RSASSA-PKCS1-v1_5 using SHA-256	RS256	SHA256withRSA	RSA
2	RSASSA-PKCS1-v1_5 using SHA-384	RS384	SHA384withRSA	RSA
3	RSASSA-PKCS1-v1_5 using SHA-512	RS512	SHA512withRSA	RSA
4	RSASSA-PSS using SHA-256 and MGF1 with SHA-256	PS256	SHA256withRSAandMGF1	RSA
5	RSASSA-PSS using SHA-384 and MGF1 with SHA-384	PS384	SHA384withRSAandMGF1	RSA
6	RSASSA-PSS using SHA-512 and MGF1 with SHA-512	PS512	SHA512withRSAandMGF1	RSA
7	ECDSA using P-256 and SHA-256	ES256	SHA256withECDSA	EC
8	ECDSA using P-384 and SHA-384	ES384	SHA384withECDSA	EC
9	ECDSA using P-521 and SHA-512	ES512	SHA512withECDSA	EC
10	HMAC using SHA-256	HS256	HMACSHA256	OCT
11	HMAC using SHA-384	HS384	HMACSHA384	OCT
12	HMAC using SHA-512	HS512	HMACSHA512	OCT
13	**Ed25519 signature algorithm**	**Ed25519**	**Ed25519**	**OKP**
14	**Ed448 signature algorithm**	**Ed448**	**Ed448**	**OKP**

Advertising supported signature algorithms by Server Metadata

[Server Metadata]
project: keycloak-services
package: org.keycloak.protocol.oidc
class: OIDCWellKnownProvider

{ ...
"id_token_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","Ed25519","Ed448"]
,"userinfo_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","Ed25519","Ed448"]
"request_object_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","none","Ed25519","Ed448"],
"token_endpoint_auth_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","Ed25519","Ed448"],
"introspection_endpoint_auth_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","Ed25519","Ed448"],
"authorization_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384","ES512","PS256","PS512","RS512","Ed25519","Ed448"],
"revocation_endpoint_auth_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384 ","ES512","PS256","PS512","RS512","Ed25519","Ed448"],
"backchannel_authentication_request_signing_alg_values_supported":
["PS384","ES384","RS384","HS256","HS512","ES256","RS256","HS384 ","ES512","PS256","PS512","RS512","Ed25519","Ed448"],
..."}}

PROPOSED DRAFT

Integration tests

- Test for access token signed by Ed25519, Ed448

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oauth/AccessTokenTest.java

- Test for OIDC flows

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/flows/AbstractOIDCResponseTypeTest.java

- Test for UserInfo response

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/UserInfoTest

- Test for authorization response (JARM)

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/AuthorizationTokenEncryptionTest.java

Integration tests

Integration tests

- REST API for generating OKP keys

testsuite/integration-arquillian/servers/auth-server/services/testsuite-providers/src/main/java/org/keycloak/testsuite/rest/resource/TestingOIDCEndpointsApplicationResource.java

- Test for generating OKP key

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/keys/GeneratedEddsaKeyProviderTest.java

- Test for server metadata (supporting Ed25519, Ed448)

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/OIDCWellKnownProviderTest.java

Integration tests

- Test for access token

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oauth/AccessTokenTest.java

- Test for OIDC flows

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/flows/AbstractOIDCResponseTypeTest.java

- Test for UserInfo response

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/UserInfoTest.java

- Test for authorization response (JARM)

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/AuthorizationTokenEncryptionTest.java

Integration tests

- Test for JWT client authentication

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oauth/ClientAuthSignedJWTTest.java

- Test for request object/JAR

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/oidc/OIDCAdvancedRequestParamsTest.java

- Test for CIBA authentication request

testsuite/integration-arquillian/tests/base/src/test/java/org/keycloak/testsuite/client/CIBATest.java

Key representation by RFC 8032

Twisted Edwards curve used by EdDSA

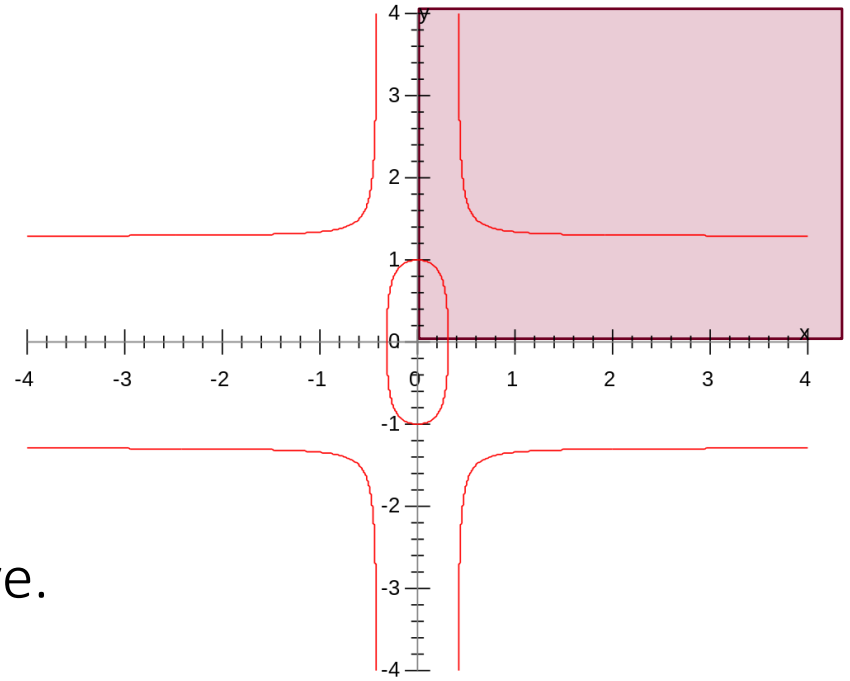
$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

E: twisted Edwards curve, additive group

p: prime number

$GF(p)$: Galois Field(Finite Field) with p elements

$\rightarrow x, y \in GF(p) = \{0,1, \dots, p-1\}$



If (x, y) satisfies the twisted Edwards curve,
then, $(p - x, y)$ is also satisfies the twisted Edwards curve.

Proof:

$x^2 = (y^2 - 1)/(dx^2 - a)\ (mod\ p)$

$(p - x)^2 = (x^2 - 2px + p^2)\ (mod\ p) = ((x^2\ mod\ p) + (p(p - 2x)\ mod\ p))\ (mod\ p)$

$= x^2\ mod\ p$

$= (y^2 - 1)/(dx^2 - a)\ (mod\ p)$

Twisted Edwards curve used by EdDSA

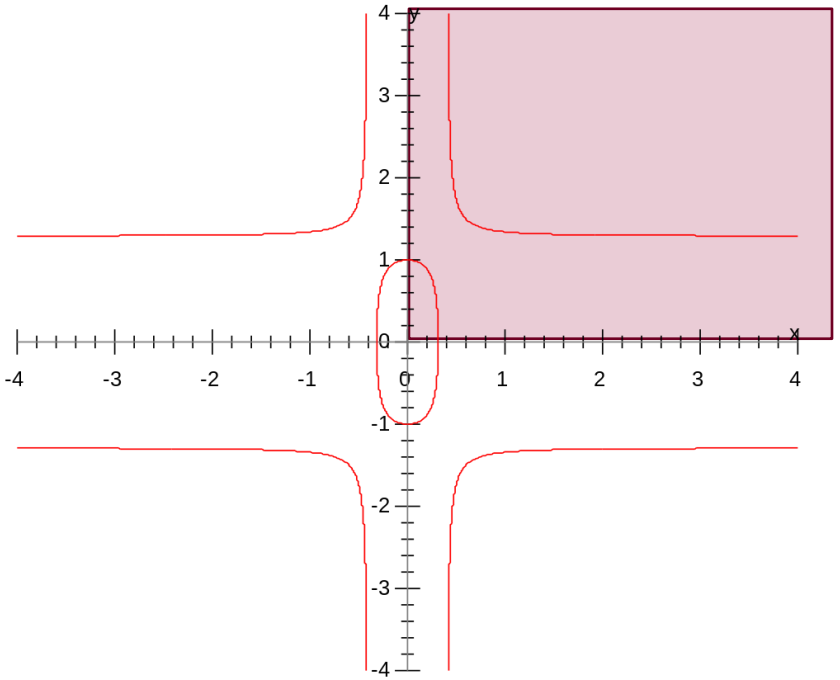
$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

E: twisted Edwards curve, additive group

p: prime number

$GF(p)$: Galois Field(Finite Field) with p elements

$\rightarrow x, y \in GF(p) = \{0,1, \dots, p-1\}$



If x is odd, then $p - x$ is even and vise versa.

Proof:

p is a prime number lager than 2, so p is odd.

If x is odd, then $p - x\ = 2k + 1\ - (2l + 1) = 2(k - l)$

If x is even, then $p - x\ = 2k + 1\ - 2l = 2(k - l) + 1$

Twisted Edwards curve used by Ed25519

$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

$A = [s]B, A, B \in E, [s] = f(s)$, s: octet string for integer

Private key: s

RFC 8032 encoding of private key for Ed25519:
https://www.rfc-editor.org/rfc/rfc8032#section-5.1.5

Private key := 32 octets of cryptographically secure random data

Twisted Edwards curve used by Ed25519

$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

$A = [s]B, A, B \in E, [s] = f(s)$, *s:* octet string for integer

Public key: $Enc(A)$ in octet string

RFC 8032 encoding of public key for Ed25519:
https://www.rfc-editor.org/rfc/rfc8032#section-5.1.5

Public key := h as encoding [s]B from a curve point (x, y) in E to 32 octets

h[0]…h[31] := representing y-coordinate as little-endian string of 32 octets

$y = h[0]256^0 + h[1]\ 256^1\ + \ldots + h[31]256^{31} = \sum_{i=0}^{31} h[i]\ 256^i$

$= h[0]2^{0\times8} + h[1]\ 2^{1\times8}\ + \ldots + h[31]2^{31\times8} = \sum_{i=0}^{31} h[i]\ 2^{i\times8}$

$= \sum_{i=0}^{31}(\sum_{j=0}^{7} b_{h[i]}[j]\ 2^j)\ 2^{i\times8} = \sum_{i=0}^{31}\sum_{j=0}^{7} b_{h[i][}j]\ 2^{j+i\times8}$

$b_{h[i]}[0]\ldots b_{h[i]}[7]$:= representing bits of h[i]

$b_{h[31]}[7]$:= The most significant bit of the final octet of h. Due to the nature of Ed25519 (0 <= y < p), it is always 0.

$b_{h[31]}[7]$ (=0) is replaced with the least significant bit of the x-coordinate.

Twisted Edwards curve used by Ed25519

$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

$A = [s]B, A, B \in E, [s] = f(s), s:$ octet string for integer
Public key: $Enc(A)$ in octet string

To summarize, considering the following characteristics:
[1] Due to the nature of twisted Edwards curve,
there are two x-coordinates corresponding to one y-coordinate.
[2] If (x', y') is a curve point in E, (p - x', y) is also a curve point in E.
[3] If (x', y') is a curve point in E, if x' is odd, then p – x' is even, and vise versa.
[4] Due to the nature of Ed25519 (0 <= y < p),
the most significant bit of y-coordinate represented as little-endian string of 32 octet is always 0.

Utilizing these characteristics, the most significant bit of y-coordinate $b_{h[31]}[7]$ is used for specifying x-coordinate's parity.

Other octets h[0]…h[31] except for $b_{h[31]}[7]$ is y-coordinate of A = [s]B represented as little-endian string of 32 octets.

Twisted Edwards curve used by Ed448

$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

$A = [s]B, A, B \in E, [s] = f(s)$, s: octet string for integer

Private key: s

RFC 8032 encoding of private key for Ed448:
https://www.rfc-editor.org/rfc/rfc8032#section-5.2.5

Private key := 57 octets of cryptographically secure random data

Twisted Edwards curve used by Ed448

$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

$A = [s]B, A, B \in E, [s] = f(s)$, *s:* octet string for integer

Public key: $Enc(A)$ in octet string

RFC 8032 encoding of public key for Ed448:
https://www.rfc-editor.org/rfc/rfc8032#section-5.2.5

Public key := h as encoding [s]B from a curve point (x, y) in E to 57 octets

h[0]…h[56] := representing y-coordinate as little-endian string of 57 octets

$y = h[0]256^0 + h[1]\ 256^1\ + \ldots + h[56]256^{56} = \sum_{i=0}^{56} h[i]\ 256^i$

$= h[0]2^{0\times8} + h[1]\ 2^{1\times8}\ + \ldots + h[56]2^{56\times8} = \sum_{i=0}^{56} h[i]\ 2^{i\times8}$

$= \sum_{i=0}^{56}(\sum_{j=0}^{7} b_{h[i]}[j]\ 2^j)\ 2^{i\times8} = \sum_{i=0}^{56}\sum_{j=0}^{7} b_{h[i][}j]\ 2^{j+i\times8}$

$b_{h[i]}[0]\ldots b_{h[i]}[7]$:= representing bits of h[i]

$b_{h[56]}[7]$:= The most significant bit of the final octet of h. Due to the nature of Ed448 (0 <= y < p),, it is always 0.

$b_{h[56]}[7]$ (=0) is replaced with the least significant bit of the x-coordinate.

Twisted Edwards curve used by Ed448

$E: \{(x, y):\ ax^2 + y^2 = 1\ + dx^2y^2, x, y \in GF(p)\}$

$A = [s]B, A, B \in E, [s] = f(s), s:$ octet string for integer
Public key: $Enc(A)$ in octet string

To summarize, considering the following characteristics:
[1] Due to the nature of twisted Edwards curve,
there are two x-coordinates corresponding to one y-coordinate.
[2] If (x', y') is a curve point in E, (p - x', y) is also a curve point in E.
[3] If (x', y') is a curve point in E, if x' is odd, then p – x' is even, and vise versa.
[4] Due to the nature of Ed448 (0 <= y < p),
the most significant bit of y-coordinate represented as little-endian string of 57 octet is always 0.

Utilizing these characteristics, the most significant bit of y-coordinate $b_{h[56]}[7]$ is used for specifying x-coordinate's parity.

Other octets h[0]…h[56] except for $b_{h[56]}[7]$ is y-coordinate of A = [s]B represented as little-endian string of 57 octets.

END