Потенциальные правки..

логи

1. GostKeyProvider - заменить log.info("Строка %s".formatted(value)) на log.info("Строка {}", value) для единообразия и производительности.
2. CertificateUtils.generateSelfSignedGostCertificate() - срок действия сертификата в константу:

private static final long CERTIFICATE\_DAYS = 365L;

1. GostAccessTokenIntrospectionProvider - удалить закомментированный логгер, оставить только один стиль логирования.

С точки зрения безопасности

GostPemUtils.calculateGostThumbprint256() - добавить проверку на null:

if (x509CertificateChain == null || x509CertificateChain.length == 0) {

throw new IllegalArgumentException("Certificate chain cannot be null or empty");

}

Может освободить ресурсы?

GostSignatureVerifierContext.verify() - добавить освобождение ресурсов:

try {

Signature signature = Signature.getInstance(GostAlgorithm.GOST\_SIGNATURE\_JCA\_NAME, "BC");

signature.initVerify((PublicKey) key.getPublicKey());

signature.update(data);

return signature.verify(signatureBytes);

} catch (Exception e) {

throw new VerificationException("Failed to verify GOST signature", e);

} finally {

*// Если нужно, освободить ресурсы*

}

В методе ниже пришлось поразбираться и думаю можно добавить поддержку всех кривых, тут у тебя экспертность выше, потенциально сделаем ли мы такое?

.convertECGostCurveParamToBCRep() - добавить поддержку всех кривых:

public static String convertECGostCurveParamToBCRep(String ecInGostRep) {

switch (ecInGostRep) {

case "GC256A": return "Tc26-Gost-3410-12-256-paramSetA";

case "GC256B": return "Tc26-Gost-3410-12-256-paramSetB";

case "GC256C": return "Tc26-Gost-3410-12-256-paramSetC";

case "GC256D": return "Tc26-Gost-3410-12-256-paramSetD";

default: throw new IllegalArgumentException("Unknown curve name: " + ecInGostRep);

}

}

Еще GostJWSHeader - добавить аннотации @Override для методов, переопределяющих родительские, чтобы избежать проблем при обновлении Keycloak.

По оргранизации кода лучше так,

1. AbstractGeneratedGostKeyProviderFactory - удалить аннотацию @Log, если класс не использует логирование.
2. GeneratedGostKeyProviderFactory.validateConfiguration() - удалить закомментированный код.
3. GostPublicJWK - добавить документацию для полей отпечатков сертификатов:

*/\*\**

\* Отпечаток сертификата SHA-1 согласно RFC 7517

*\*/*

private String sha1x509Thumbprint;

*/\*\**

\* Отпечаток сертификата SHA-256 согласно RFC 7517

*\*/*

private String sha256x509Thumbprint;

*/\*\**

\* Отпечаток сертификата ГОСТ 34.11-2012 по МР.26.2.002-2024

*\*/*

private String gostThumbprint;

рефакторинг общий

1. GostKeyUtils.extractPublicKey() - использовать более современный синтаксис:

if (privateKey instanceof BCECGOST3410\_2012PrivateKey) {

BCECGOST3410\_2012PrivateKey ecPrivateKey = (BCECGOST3410\_2012PrivateKey) privateKey;

*// Существующий код*

} else if (privateKey instanceof BCECGOST3410PrivateKey) {

throw new RuntimeException("ГОСТ 34.10-2001 ключ не поддерживается! Ожидается ГОСТ 34.10-2012.");

} else {

throw new RuntimeException("Unexpected key type: " + privateKey.getClass().getName());

}

1. CustomJWKBuilder.getCertificateChain() - упростить метод:

private static List<X509Certificate> getCertificateChain(KeyWrapper key) {

if (key.getCertificateChain() != null && !key.getCertificateChain().isEmpty()) {

return key.getCertificateChain();

}

return key.getCertificate() != null ? List.of(key.getCertificate()) : List.of();

}

1. GostHashProvider - вынести "BC" в константу:

private static final String PROVIDER = "BC";

подумать над оптимизацией

GostPemUtils - кешировать KeyFactory для повторного использования:

private static final Map<String, KeyFactory> KEY\_FACTORY\_CACHE = new ConcurrentHashMap<>();

private static KeyFactory getKeyFactory(String algorithm) throws NoSuchAlgorithmException, NoSuchProviderException {

return KEY\_FACTORY\_CACHE.computeIfAbsent(algorithm, alg -> {

try {

return KeyFactory.getInstance(alg, "BC");

} catch (NoSuchAlgorithmException | NoSuchProviderException e) {

throw new RuntimeException("Failed to create KeyFactory", e);

}

});

}

Паттерны можешь внедрить, можешь и нет , решай сам

Фасад - создать GostCryptoFacade для упрощения работы с криптографическими операциями:

пример

public class GostCryptoFacade {

public KeyPair generateKeyPair(GostCurveParams curve) {

try {

return GostKeyUtils.generateGostKeyPair(curve.getCurveParam());

} catch (Exception e) {

throw new RuntimeException("Failed to generate key pair", e);

}

}

public X509Certificate createCertificate(KeyPair keyPair, String subject) {

return CertificateUtils.generateSelfSignedGostCertificate(

keyPair, subject, GostAlgorithm.GOST\_SIGNATURE\_JCA\_NAME);

}

public boolean verifySignature(byte[] data, byte[] signature, PublicKey publicKey) {

try {

Signature sig = Signature.getInstance(GostAlgorithm.GOST\_SIGNATURE\_JCA\_NAME, "BC");

sig.initVerify(publicKey);

sig.update(data);

return sig.verify(signature);

} catch (Exception e) {

throw new RuntimeException("Signature verification failed", e);

}

}

}

В самом баунсиКастл можно использовать Singleton - управление провайдером BouncyCastle:

Тут пример:

public class BouncyCastleProviderHolder {

private static final BouncyCastleProviderHolder INSTANCE = new BouncyCastleProviderHolder();

private final BouncyCastleProvider provider;

private BouncyCastleProviderHolder() {

provider = new BouncyCastleProvider();

if (Security.getProvider("BC") == null) {

Security.addProvider(provider);

}

}

public static BouncyCastleProviderHolder getInstance() {

return INSTANCE;

}

public BouncyCastleProvider getProvider() {

return provider;

}

}

+ еще можно

1. Добавить unit-тесты для GostKeyUtils с фикусом на:

генерацию ключей с разными параметрами

извлечение публичного ключа

обработку ошибок

1. Добавить тесты на обработку некорректных сертификатов и ключей для повышения надежности.

И еще, функциональные требования не проверял, только нефункциональные. П организации кода на все лайк, минимальные правки, при желании можешь оставить все в таком виде