There are few things, which can be improved with current implementation of authenticatedClientSessionModel, codeToToken endpoint and refresh token endpoint.

Here is some overview of the current issues and how we can address them:

Issues

1. The UserSessionModel currently contains at max 1 authenticatedClientSession per client. There is the map like this on UserSessionModel (keys of the map are client UUIDs):

```
Map<String, AuthenticatedClientSessionModel>
getAuthenticatedClientSessions();
```

AuthenticatedClientSessionModel keeps the list of roles, list of protocol mappers and client notes.

The issue is, that if there are multiple browser tabs for same client application, they may not have same notes, protocolMappers and roles (for example if we limit the roles based on "scope" parameter, the roles can be different in browser tab1 and tab2 as every tab can have different scope parameter. This is not big issue now, since we don't have proper support for "scope" parameter, but could be in the future..

2. The authenticatedClientSession is currently needed in code-to-token request. This is a bit of perf issue in cross-dc. The code-to-token request can happen in different DC then authentication. It will be ideal if code-to-token request doesn't need to wait until userSession with attached authenticatedClientSession is replicated from the second DC

3. There is just 1 valid code per authenticatedClientSession tracked in the attached note - in case that 2 user sends 2 concurrent requests to *http://localhost:8080/customer-portal*, it may happen that there are 2 SSO logins concurrently issued, however just 1 of them will successfully pass the code-to-token request (clientSession note will be overwritten by the second request, so first request will fail in code-to-token request due the different note value)

4. Refresh token is signed by same key like accessToken. This is not ideal for:

- performance (RSA is an unecessary overhead. Symmetric cryptography is sufficient)
- Security:
    - We had security issues that refresh tokens and access tokens were exchanged . Check on token types are easy to add, but they are not so robust like using different token format for both tokens.
    - Refresh token is JWT and can be seen by application (not a security threat, but may be better if it's hidden)

Possible solution proposal

- Move some stuff from authenticatedClientSessionModel to the *code* parameter and refreshToken itself.

- Code would be JWT containing just clientSession notes. Not the protocolMappers and roles

- Refresh token will contain both roles and protocolMappers (roles are here already. The protocolMappers will be new thing on refresh token)

- AuthenticatedClientSessionModel won't contain protocolMappers and roles anymore

- Ensure that both code and refreshToken are AES-128 encrypted. Just the keycloak server has the key and can decrypt them

- Use single-use cache to track whether the particular value of code parameter was already used.

Details

Use JWS for code? Likely not...

In first stage, I've tried the code to be JWS. The JSON data would contain some details about notes, protocol mappers, roles and other things added to client session. Something like this:

```
{
  "protMappers": [
     "4b1b73fa-5ca4-424a-aca8-5251ea3789e6",
     "11166acd-7b9c-4e25-938e-5eba1068158e",
     "d8f13d6a-593f-4eb9-8614-0be29edd5787",
     "3320bfb9-24b6-4bb1-9f76-ae65a360c1e6",
     "907dcdcc-2974-4356-a172-16a87fae9855",
     "b22a2542-a9d6-4d44-9111-ce6e2d3869d3"
  ],
  "aud": "d8f13d6a-593f-4eb9-8614-0be29edd5798",
  "redu": "http://localhost:8080/foo/bar/baz/dar",
  "notes": {
     "scope": "openid",
     "SSO_AUTH": "true",
     "iss": "http://localhost:8081/auth/realms/master",
     "response_type": "code",
     "code_challenge_method": "plain",
     "code_challenge": "foo",
     "redirect_uri":
"http://localhost:8081/auth/admin/master/console/",
     "state": "0833cdef-f188-45aa-9eb3-7cbaf1fd9b99",
     "nonce": "170bad50-c432-44ce-8939-b82d5bce67da",
     "response_mode": "fragment"
  },
  "uss": "d8f13d6a-593f-4eb9-8614-0be29edd5797",
  "roles": [
     "4c1b73fa-5ca4-424a-aca8-5251ea3789e6",
     "12166acd-7b9c-4e25-938e-5eba1068158e",
     "d8f13d6a-593f-4eb9-8614-0be29edd5797",
     "3320bfb9-24b6-4bb1-9f76-ae65b360c1e6",
     "907dcdcc-2984-4356-a172-16a87fae9855",
     "b22b2542-a9d6-4d44-9111-ce6e2d3869d3"
  ]
}
```

The issues of JWS are:
a. The JSON is readable by adapter and users.

There is (at least) one security issue that PKCE (https://tools.ietf.org/html/rfc7636) with code_challenge method "plain" will be broken.

That’s because we currently save the code_challenge as a note on client session. So assuming that the notes are saved on the code JWT, then anyone with access to code parameter will be able to look at JSON and get code_challenge note from it and use it as code_verifier parameter.

b. The code parameter is too big. There is limit 2000 characters for the maximum URL size to ensure that it safely works across various browsers/platforms.

The example JSON itself has 960 characters. Signature adds few hundreds more. With more roles and protocol mappers, it can growths further...

Use JWE instead of JWS (encrypted "code" instead of signed)

Using JWE helps with the (a). But (b) still applies.

I've encrypted the 960 characters long JSON from above with AES-128 and it takes 1300 characters. So encryption adds about 30% more chars.

How to solve big code? Skip protocol mappers and roles in code entirely

This should solve issue (b). In other words, just notes and few other basic things are part of the JWT, which makes the code size much lower. Both roles and protocolMappers will be removed from the authenticatedClientSessionModel on server side too.

More details:

- Currently TokenManager.attachAuthenticationSession is called after the authentication, requiredActions and consents are confirmed. It creates AuthenticatedClientSessionModel, pulls the stuff from AuthenticationSessionModel and attaches clientSession to userSession.
- We can just omit the protocolMappers and roles generation here and not put anything to the code JWT
- In code-to-token endpoint we will:
    - Regenerate the roles and protocolMappers based on scope parameter (known from the notes attached to code JWT)
    - In case that consent is required for client, we will check that just roles and protocolMappers with persisted grantedConsent are applied. It can happened that protocolMappers and roles membership changed at the time when user had consent screen shown. So this will ensure that just approved consents are applied
    - protocolMapper UUIDs will be added to the refresh token. Roles are there already.
- For the next refresh token requests, roles and protocolMappers are taken from the current refresh token instead of from authenticatedClientSessionModel.
- TokenManager.verifyAccess check during refreshToken endpoint will be still there (if user is not member of some role anymore, refreshToken will fail.) In case that

protocolMapper was removed from the client, it's just not applied. Newly added protocolMappers to client also won't be applied

- Refresh token endpoint will require authenticatedClientSessionModel exists on userSessionModel. This is to doublecheck that "Revoke grant" wasn't called for the user in the meantime (Note: Revoke-grant currently removes the attached clientSession from the userSession too.) Same for Introspection endpoint, UserInfo endpoint etc. Not sure about authorization functionalities?

- After code-to-token endpoint finished, new item added to single-use cache. The default lifetime is 1 minute (specified by the realm timeout for exchange code).
    - Possibility for user to prefer Security vs Performance (Eg. ability to use asynchronous cache with the risk that code-to-token with same code can pass if executed concurrently on 2 datacenters)? IMO Not a big priority for now? But can be easily added in the future.

- Protocol Mappers SPI signature change: AuthenticatedClientSessionModel will need to be removed and replaced with the “notes” and “roles” . Both are available in code-to-token and refreshToken endpoint. For SAML protocol mappers probably no change needed