Google/Apple token generating process

Workflow:

1. $tek_i \leftarrow CRNG(16)$.
    1. CRNG stands for cryptographic random number generator
2. $RPIK_i \leftarrow HKDF(tek_i, NULL, UTF8(\text{" } EN - RPIK \text{ "}), 16)$.
    1. RPIK stands for Rolling Proximity Identifier Key, will be used to generate RPI
    2. HKDF is a key derivation function based on a hash-based message authentication[1] code (HMAC). It says if two RPIK are the same, their corresponding tek should be the same
3. $RPI_{i,j} \leftarrow AES_{128}(RPIK_i, PaddedData_j)$
    1. $PaddedData_j$ is a time related data and encoded in 32 bits: $\frac{TimeStampInSeconds}{600}$
    2. AES_{128} is symmetric

What will happen after user receive a RPI:

1. Pre: Use tek list downloaded from server to generate $RPIK$ list.
2. $RPI_{i,j} \rightarrow RPIK_i$. This is done via AES_{128} and key used is $PaddedData_j$
3. See if there is a match of $RPIK$

What Google provides in Github:

- Google does not provide implementation details. Instead, it provides high level APIs

```
import com.google.android.gms.nearby.exposurenotification.ExposureConfiguration;
import com.google.android.gms.nearby.exposurenotification.ExposureInformation;
import
com.google.android.gms.nearby.exposurenotification.ExposureNotificationClient;
import com.google.android.gms.nearby.exposurenotification.ExposureSummary;
import com.google.android.gms.nearby.exposurenotification.TemporaryExposureKey;
```

My Plan accordingly:

- Declare all the methods involved. These methods will not change the input and will return directly. We are going to implement cryptographic details in the future.
- This week will focus on the workflow:
    - How to broadcast and scan periodically.
    - How to upload information to the server
    - How to download informaiton from server periodically.
    - Database design on the server