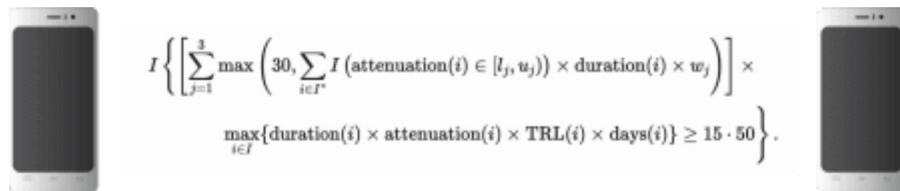


Abstract:

We attempt a mathematical description of the risk scoring algorithm used in the Google and Apple exposure notification framework (v1). This algorithm is used in many digital contact tracing apps for COVID-19, e.g., in Germany or Switzerland.


$$I \left\{ \left[\sum_{j=1}^3 \max \left(30, \sum_{i \in I^*} I(\text{attenuation}(i) \in [l_j, u_j]) \times \text{duration}(i) \times w_j \right) \right] \times \max_{i \in I} \{ \text{duration}(i) \times \text{attenuation}(i) \times \text{TRL}(i) \times \text{days}(i) \} \geq 15 \cdot 50 \right\}.$$



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/). The markdown+Rknitr source code of this blog is available under a [GNU General Public License \(GPL v3\)](https://www.gnu.org/licenses/gpl-3.0.html) license from github.

Motivation

My phone runs a digital COVID-19 contact tracing app based on the Apple and Google exposure notification (GAEN) framework [[Apple](#) | [Google](#) | [Wikipedia](#)]. Since my future health might depend on it, I am interested in the risk classification algorithm of the framework: when does it warn me about a possible exposure risk? Since algorithmic transparency is a key component in gaining acceptance for digital contact tracing apps, this blog post tries to give a mathematical description of the risk scoring algorithm in the GAEN framework – as far as I have been able to understand it. The description is accompanied by a small case study implemented in R.

The GAEN is used by country specific COVID-19 Apps in [several countries](#), e.g., Switzerland, Latvia, Denmark, Italy, Poland. In this post special focus is on the German [Corona-Warn-App](#) (CWA), which comes with a special commitment to transparency:

The Corona-Warn-App is an app that helps trace infection chains of SARS-CoV-2 (which can cause COVID-19) in Germany. The app is based on technologies with a decentralized approach and notifies users if they have been exposed to SARS-CoV-2. Transparency is key to both protect the app's end-users and to encourage adoption.

This means that all code and documentation of the CWA is available under an open-source license through [GitHub](#). A good overview of the technical foundation of the app is provided [here](#). However, important parts of the CWA risk scoring are taken from the GAEN framework, which is less transparent¹. In particular I was interested in the [API v1](#) form of the risk scoring, because this forms the basis of the CWA and also provides epidemiological flexibility through the use of the transmission risk level parameter.

The Risk Scoring

A phone using the GAEN framework scans the immediate surroundings in regular time intervals² using the [BLE](#) protocol. Doing so the phone collects anonymous identifier keys of the devices running BLE near you. If a GAEN user is tested positive, they can voluntarily decide to upload their anonymous identifier keys to a server. With regular intervals the app on your phone

downloads these keys and investigates whether there is a match between the keys you encountered and those available on the server. If there is a match, an algorithm decides based on attributes of the contact, e.g. duration, distance and infectiousness of the potential infector, whether the user should be displayed a warning about an increased risk for infection. It is this classification algorithm, which I am interested in.

Introducing mathematical notation, let (E) be the set of encounters recorded by your phone during the last 14 days. We assume each entry $(e \in E)$ consists of attributes³ containing the identifier key of the phone you met, $(\text{key}(e))$, (aka. *temporary exposure key* (TEK)), the calendar day the contact happened, $(\text{date}(e))$, the duration of the contact, $(\text{duration}(e))$, and the average attenuation of the signal during the contact, $(\text{antennuation}(e))$. The later serves as proxy for the distance between the two phones.

Furthermore, the server contains the set of uploaded keys by CWA users, who tested positive for COVID-19 and who gave consent to upload their keys. Uploaded keys of test positives are also called *diagnosis keys* – technically they are however just TEKs. For simplicity we shall assume in this document that a phone is using one key per calendar day and that an upload of keys consists of the keys used on the day of upload and the previous 13 days⁴. Let (S) denote the set of uploaded keys and for $(s \in S)$ let $(\text{key}(s))$ denote the diagnosis key, $(\text{day_valid}(s))$ the calendar day the key was in use and $(\text{TRL}(s))$ the transmission risk. The later is a numerical value for how infectious the potential infector was on $(\text{day_valid}(s))$. For simplicity, in what follows we shall consider only the so called **level** of the duration, attenuation and transmission risk, which in the GAEN API is a discretized version of the continuous quantity represented by an integer between 0-8.

We shall denote by (I) the subset of (E) , which consists of keys known to have been positively tested for COVID-19, that is the keys have been uploaded to the server. For simplicity we shall extend the elements of (E) with additional information provided by the server about the key, i.e. we let $I = \{ (e, s) : e \in E, s \in S, \text{key}(s) = \text{key}(e) \}$.

Case Study with Aisha, Anton and Betty

We shall illustrate the above using an example inspired by the [CWA documentation](#). In the example the CWA user Betty travels with Anton and Aisha together by bus to and from work on the 9th and the 16th of the month. Each trip lasts 10 minutes. Anton sat one meter away from Betty, while Aisha sat two meters away. Both Anton and Aisha become test positive on the 20th and decide to upload their keys to the server on the 20th and 21st, respectively. We perform Betty's risk calculation on the 20th and 21st, respectively. To make the example clearer, the original trip length of 10 minutes is changed to 11 minutes.

GAEN Configuration

Configuration of the risk scoring parameters as done for the CWA:

```
## Transmission risk level provided at key upload - see https://github.com/corona-warn-app/cwa-app-android/blob/fe9128fdb28384640d0a248b7ff46701d6a04f0e/Corona-Warn-App/src/main/java/de/rki/coronawarnapp/util/ProtoFormatConverterExtensions.kt#L12 for the google App
trl <- structure(c(5, 6, 8, 8, 8, 5, 3, rep(1, 7)), names=0:13)

# Breaks in dB for the attenuation - see https://github.com/google/exposure-
```

<https://github.com/google/exposure-notifications-internals/blob/e953a09dd3c20c04dfa29e362eef461890dac25c/exposurenotification/src/main/java/com/google/samples/exposurenotification/features/ContactTracingFeature.java#L515>

```
attenuation_breaks <- c(0, 10, 15, 27, 33, 51, 63, 73, Inf)
attenuation_include_lowest <- TRUE
# Level for each attenuation bucket, see https://github.com/corona-warn-app/cwa-server/blob/master/services/distribution/src/main/resources/master-config/exposure-config.yaml
```

```
attenuation_lvl <- tibble( bucket = c("[0,10]", "(10,15]",
"(15,27]", "(27,33]", "(33,51]", "(51,63]", "(63,73]", "(73,Inf]"),
lvl=c(2, 2, 2, 2, 2, 2, 2, 0))
```

Bucket limits in <https://github.com/google/exposure-notifications-internals/blob/e953a09dd3c20c04dfa29e362eef461890dac25c/exposurenotification/src/main/java/com/google/samples/exposurenotification/features/ContactTracingFeature.java#L502> (note that the buckets are closed to the right)

```
duration_breaks <- c(-Inf, 0, 5, 10, 15, 20, 25, 30, Inf)
duration_include_lowest <- FALSE
# Levels - see https://github.com/corona-warn-app/cwa-server/blob/f3d66840ef59e2ced10fb9b42f08a8938e76075a/services/distribution/src/main/resources/master-config/exposure-config.yaml#L50
```

```
duration_lvl <- tibble( bucket = c("(-Inf,0]", "(0,5]", "(5,10]",
"(10,15]", "(15,20]", "(20,25]", "(25,30]", "(30, Inf]"), lvl=c(0, 0,
0, 1, 1, 1, 1, 1))
```

Days parameters - see <https://github.com/corona-warn-app/cwa-server/blob/f3d66840ef59e2ced10fb9b42f08a8938e76075a/services/distribution/src/main/resources/master-config/exposure-config.yaml#L40>

```
days_lvl <- 5
```

```
# Time_attenuation_buckets (to be used later)
time_attenuation_bucket_breaks <- c(0, 55, 63, Inf)
time_attenuation_buckets <- levels(cut(0:100, breaks=time_attenuation_
bucket_breaks, include.lowest=TRUE, right=FALSE))
```

The `days(e)` parameter of an `(e|n E)` exposure reflects the possibility to provide additional risk relevant information about the CWA user at the day of the contact. However, the CWA currently keeps it at a fixed value of 5 and, hence, it plays little role in the risk calculations.

Part 1: Uploaded keys on the server

Anton had his CWA running since 2020-09-13, while Aisha has been running her app since 2020-09-01. For each of the two we build a tibble containing the uploaded keys at the day of transmission, i.e. on the 20th and 21st, respectively⁵. Furthermore, we construct two tibbles mimicking the state of the diagnosis key server on 2020-09-20 and 2020-09-21, respectively.

```
# Upload of Anton on the 2020-09-20 where his app has been running for 8 days
```

```
anton_ndays <- min(14, as.numeric(as.Date("2020-09-20") -
cwa_start_anton + 1))
```

```

anton <- tibble(key=rkey(anton_ndays),valid=seq(as.Date("2020-09-20"),
,length.out=anton_ndays,by="-1 day"), trl=trl[seq_len(anton_ndays)])

# Upload of Aisha on the 2020-09-21 where her app has been running for
14 days
aisha_ndays <- min(14, as.numeric(as.Date("2020-09-20") -
cwa_start_aisha + 1))
aisha <- tibble(key=rkey(aisha_ndays),valid=seq(as.Date("2020-09-21"),
,length.out=aisha_ndays,by="-1 day"), trl=trl[seq_len(aisha_ndays)])

# Upload to server consists of keys for the last 13 days (note: day
zero caveat is ignored). Hence dk consists of all keys who were active
during the last 13 days within
dk20200920 <- anton %>% filter(valid >= as.Date("2020-09-20") - 13)
dk20200921 <- rbind(anton, aisha) %>% filter(valid >=
as.Date("2020-09-21") - 13)

# Show diagnosis keys of 2020-09-20 (these are Anton's keys)
dk20200920

```

```

## # A tibble: 8 x 3
##   key      valid      trl
##
## 1 2a0a87 2020-09-20      5
## 2 051447 2020-09-19      6
## 3 f810f4 2020-09-18      8
## 4 17a0d6 2020-09-17      8
## 5 3d12e3 2020-09-16      8
## 6 aaf590 2020-09-15      5
## 7 22b909 2020-09-14      3
## 8 1be004 2020-09-13      1

```

Part 2: Betty's exposure history

Betty's sighting history on the 20th would look as follows:

```

## # A tibble: 6 x 7
##   key      date      attenuation duration attenuation_lvl
duration_lvl days_lvl
##
## 1 3d12e3 2020-09-16      40      11      2
1      5
## 2 3d12e3 2020-09-16      40      11      2
1      5
## 3 835292 2020-09-16      60      11      2
1      5
## 4 835292 2020-09-16      60      11      2
1      5
## 5 3d8508 2020-09-09      60      11      2
1      5
## 6 3d8508 2020-09-09      60      11      2
1      5

```

Hence, Betty's set of exposures with test positives provided by the server on the 20th is:

```
i_set <- inner_join(eh_betty, dk20200920, by="key")
i_set

## # A tibble: 2 x 9
##   key      date      attenuation duration attenuation_lvl
duration_lvl days_lvl valid          trl
##
## 1 3d12e3 2020-09-16          40         11              2
1      5 2020-09-16      8
## 2 3d12e3 2020-09-16          40         11              2
1      5 2020-09-16      8
```

Risk scoring

The description of the risk scoring in the CWA app is centered around [Fig. 12 and 13](#) of the solution architecture documentation. An illustrative example of how this computation looks can be found [here](#). For further details the source code for the risk scoring in the CWA can be consulted [[Android version](#) | [iOS](#)]. However, for the Android version, which is the one I checked most, it is [not 100% transparent](#) (to me) how the important `exposureSummary` object is generated by the API. Nevertheless, the computation of the GAEN risk scoring algorithm consists of two steps: a per-exposure risk score calculation followed by an aggregation over all exposures.

Risk Score

We compute the total risk for each exposure with a test positive, i.e. for each element $i \in I$, by:

$$\begin{aligned} \text{TR}(i) = & \text{duration}(i) \times \\ & \text{attenuation}(i) \times \\ & \text{TRL}(i) \times \\ & \text{days}(i) \end{aligned}$$

Exposure risks with a total risk below the [minimum risk score](#) of 11 are put to zero:

```
# Minimum risk score - see https://github.com/corona-warn-app/cwa-server/blob/f3d66840ef59e2ced10fb9b42f08a8938e76075a/services/distribution/src/main/resources/master-config/app-config.yaml#L13
minimum_risk_score <- 11

# Compute risk score and filter out those below the minimum
risk <- i_set %>%
  mutate(risk_score = attenuation_lvl * duration_lvl * trl * days_lvl)
  %>%
  filter(risk_score >= minimum_risk_score)

risk

## # A tibble: 2 x 10
##   key      date      attenuation duration attenuation_lvl
duration_lvl days_lvl valid          trl risk_score
```

```
##
## 1 3d12e3 2020-09-16      40      11      2
1      5 2020-09-16      8      80
## 2 3d12e3 2020-09-16      40      11      2
1      5 2020-09-16      8      80
```

From this we can compute the so called **Normalized Total Risk Score** defined as $\text{Normalized Total Risk Score} := \frac{\max_{i \in I} \text{TR}(i)}{50}$.
 I'm guessing, but the value of 50 appears to be a “baseline” for the transmission risk given by the risk test positives have on the day they upload their keys (first element of `tr1` is 5 and, hence, $(1 \times 2 \times 5 \times 5 = 50)$).

```
# Derive the normalized total risk score
maxTR <- max(risk$risk_score)
ntrs <- maxTR / 50

c(maxTR, ntrs)

## [1] 80.0 1.6
```

In order to reflect that the accumulation of several exposures with a test positive increases your risk, a weighted summation of time spent in three 3 attenuation buckets (distance: far, intermediate and close) is computed.

For each class $(j=1,2,3)$ (low, mid and high) a weighted time in the class is computed
$$wt_j = \max\left(30, \sum_{i \in I^*} \mathcal{I}\left(l_j \leq \text{attenuation}(i) < u_j\right) \times \text{duration}(i) \times w_j\right)$$

 where $\mathcal{I}(A)$ denotes the indicator function which is 1 if the condition (A) is fulfilled and zero otherwise. Furthermore, $(I^* = \{i \in I : \text{TR}(i) > 0\})$, $(w_1=0, w_2=0.5)$ and $(w_3=1)$ and the limits (l_j, u_j) of the three classes are [chosen](#) as $([63, \infty))$, $([55, 63))$ and $([0, 55))$ dB. Note that for reasons beyond my knowledge, the time in each class is capped at 30 minutes. The total time is then calculated as
$$\text{Total time} = \sum_{j=1}^4 wt_j,$$

 where $(wt_4 \geq 0)$ is denoted a bucket offset value, which is currently set to 0 in the CWA. Hence, the largest possible total time is $(0 \times 30 + 0.5 \times 30 + 1 \times 30 + 0 = 45)$ minutes.

```
# Accumulate time in the 3 attenuation buckets
ta_buckets <- risk %>%
  mutate( time_attenuation_bucket = cut(attenuation,
    breaks=time_attenuation_bucket_breaks, include.lowest=TRUE,
    right=FALSE)) %>%
  group_by(time_attenuation_bucket) %>%
  summarise(time = sum(duration)) %>%
  # Cap at 30 minutes
  mutate(time = pmin(30, time, na.rm=TRUE))

# Define the weights - https://github.com/corona-warn-app/cwa-server/blob/f3d66840ef59e2ced10fb9b42f08a8938e76075a/services/distribution/src/main/resources/master-config/attenuation-duration.yaml#L15
weights <- tibble( time_attenuation_bucket=time_attenuation_buckets,
```

```
weight=c(1, 0.5, 0))
ta_buckets_w <- right_join( ta_buckets, weights,
by="time_attenuation_bucket")

# Do the weighted summation and add the bucket offset
exposure_score <- ta_buckets_w %>% summarise(total = sum(time*weight,
na.rm=TRUE) + 0) %>% as.numeric()
```

The combined risk score is now given as \[

$$\text{Combined Risk Score} = \text{Exposure Score} \times \text{Normalized Total Risk Score}$$

\]

A warning is given by the app, if the combined risk is above a threshold value of 15.

In code:

```
# Compute combined risk score and classify
(combined_risk_score <- exposure_score * ntrs)

## [1] 35.2

(alarm <- combined_risk_score >= 15)

## [1] TRUE
```

Hence, Betty receives an alarm on the 20th.

The classifier in a nutshell

To summarise, the resulting binary classifier of the CWA can be expressed mathematically as

$$\begin{aligned} & \left(\sum_{j=1}^3 \max_{i \in [l_j, u_j]} \left(\text{attenuation}(i) \times \text{duration}(i) \times w_j \right) \right) \times \\ & \left(\max_{i \in I} \left(\text{duration}(i) \times \text{attenuation}(i) \times \text{TRL}(i) \times \text{days}(i) \right) \right) \geq 15 \cdot 50 \end{aligned}$$

The formula can also be assembled into one classifier function, which we will use to repeat the risk classification based on the available server information on 2020-09-21:

```
cwa_classifier <- function(i_set) {
  # Compute risk score and filter out those below the minimum
  risk <- i_set %>%
    mutate(risk_score = attenuation_lvl * duration_lvl * trl *
days_lvl) %>%
    filter(risk_score >= minimum_risk_score)

  # Derive the normalized total risk score
  maxTR <- max(risk$risk_score)
```

```

ntrs <- maxTR / 50

# Accumulate time in the 3 attenuation buckets
ta_buckets <- risk %>%
  mutate( time_attenuation_bucket = cut(attenuation,
breaks=time_attenuation_bucket_breaks, include.lowest=TRUE,
right=FALSE)) %>%
  group_by(time_attenuation_bucket) %>%
  summarise(time = sum(duration)) %>%
  # Cap at 30 minutes
  mutate(time = pmin(30, time, na.rm=TRUE))

# Define the weights - https://github.com/corona-warn-app/cwa-server/blob/f3d66840ef59e2ced10fb9b42f08a8938e76075a/services/distribution/src/main/resources/master-config/attenuation-duration.yaml#L15
weights <- tibble( time_attenuation_bucket=time_attenuation_buckets,
weight=c(1, 0.5, 0))
ta_buckets_w <- right_join( ta_buckets, weights,
by="time_attenuation_bucket")

# Do the weighted summation and add the bucket offset
exposure_score <- ta_buckets_w %>% summarise(total = sum(time*weight,
na.rm=TRUE) + 0) %>% as.numeric()

# Compute combined risk score and classify
(combined_risk_score <- exposure_score * ntrs)
(alarm <- combined_risk_score >= 15)

# Return individuals elements of the risk classification
return(list( risk=risk, ta_buckets_w=ta_buckets_w,
exposure_score=exposure_score, ntrs=ntrs, combined_risk_score=combined_
risk_score, alarm=alarm))
}

```

To illustrate use of the function we will repeat the classification with the server information available on 2020-09-21. Note that this information is slightly different than on 2020-09-20, because now also Aisha's keys are available:

```

i_set <- left_join(eh_betty, dk20200921, by="key")
cwa_classifier(i_set)

## $risk
## # A tibble: 4 x 10
##   key      date      attenuation duration attenuation_lvl
duration_lvl days_lvl valid          trl risk_score
##
## 1 3d12e3 2020-09-16      40          11              2
1      5 2020-09-16      8           80
## 2 3d12e3 2020-09-16      40          11              2
1      5 2020-09-16      8           80
## 3 835292 2020-09-16      60          11              2

```



```

1          5 2020-09-16      5          50
## 4 835292 2020-09-16      60          11          2
1          5 2020-09-16      5          50
##
## $ta_buckets_w
## # A tibble: 3 x 3
##   time_attenuation_bucket  time weight
##
## 1 [0,55)                22      1
## 2 [55,63)                22     0.5
## 3 [63,Inf]               NA      0
##
## $exposure_score
## [1] 33
##
## $ntrs
## [1] 1.6
##
## $combined_risk_score
## [1] 52.8
##
## $alarm
## [1] TRUE

```

In other words and not so surprising we also get an alarm, if Betty runs the risk scoring on the 21st. The combined risk score is now also higher due to the multiple exposures.

Discussion

Despite an honest effort to try to understand the code and the documentation, there are still specific details, which I'm unsure about. One reason is that only parts of the code of the GAEN framework v1 are public – this should be improved. Another reason is that I lack the technical skills to really build and execute the CWA from source on my phone or to evaluate the GAEN snippets. This would allow me to test different scenarios and gain insight based on a reverse engineering approach. A mathematical and high-level implementation documentation as the one attempted in this post is not easy to understand either, but math is a universal language with enough abstraction to not get lost in the nitty-gritty details of source code. If the source code is public, however, a ground truth can always be established – it may just take time... This post mixes the two approaches by supplementing the math with an illustrative example using R code...