Alias Generator

David Mis

March 10, 2014

This report describes the alias generator as implemented in the network prototype. The alias generator allows phones to generate pseudonyms that appear random to network administrators, but still allow the phone to be reached by anyone who was previously authorized by the phone to do so. The generator described in this report not only supplies unpredictable pseudonyms, but also generates new pseudonyms at unpredictable times (from the point of view of network administrators). The only restriction is that the time between pseudonym updates can not exceed a `GLOBAL_MAX_TIME` parameter. All pseudonyms older than `GLOBAL_MAX_TIME` can be discarded from the network location registers; otherwise the network would need to store stale pseudonyms indefinitely.

I see three reasons why random update intervals are preferable over a fixed-interval scheme: first, it prevents all phones from updating pseudonyms at the same time, which would put unnecessary load on the network. Second, with carefully chosen parameters, it will more difficult for network administrators to detect exactly how many phones are in a given location area at any particular time (I still need to prove this, and I suspect this advantage vanishes as the number of phones in area increases.) Third, it will be more difficult for a network administrator to know if a location update is coming from a phone that is being turned on, moving location areas, or just providing its periodic update.

Assume we have two phones, Alice and Bob. Each phone has two secrets it must share in order to be reachable for calls—a `timing_secret` and a `ID_secret`[1]. At the highest level, the alias generator is a one-way function F such that:

$$F(\texttt{timing\_secret}, \texttt{ID\_secret}, \texttt{current\_time}) = \texttt{current\_pseudonym}.$$

[1]I see no reason why the two secrets can not be the same value, but I keep them separate for clarity and generality.

The generator proceeds in two phases. In the first phase, it determines the last time the pseudonym was updated, `last_update_time`, based on `timing_secret` and `current_time`. In the second phase, the generator produces `current_pseudonym` based on `last_update_time` and `ID_secret`. Phones offer `current_pseudonym` to the network at `last_update_time` and when changing location areas in order to be reachable for calls. Also, phones can initiate an outgoing call by providing the network with the `current_pseudonym` of a friend. The rest of this report describes the two phases of the generator and concludes with a brief discussion. The entire generator process is illustrated in AliasGeneratorIllustration.pdf.

The alias generator has several parameters, described below. All times throughout this report are in milliseconds since the start of the UNIX Epoch[2]:

`GLOBAL_MAX_UPDATE`:	The network's timeout for aliases in the location registers. Phones must not have intervals longer than `GLOBAL_MAX_UPDATE` between alias updates or they may be unreachable. This is the only global parameter; all others can be set individually by phones and shared with friends (although it may be preferable for all phones to share the same parameters to avoid leaking information based on the timing of updates).
`PERIOD_LENGTH`:	A period is a relatively long time frame which provides a reference for computing `update_time`. In the prototype network, phones set this parameter to 1 day.
`MIN_UPDATE`:	A phone-specific minimum time between updates. The prototype implementation assumes this is a positive value, but it could be set to 0 with only a few small modifications.
`MAX_UPDATE`:	A phone-specific maximum time between updates. Must be less than or equal to `GLOBAL_MAX_UPDATE`.
`UPDATE_GRANULARITY`:	A relatively small length of time that gives the number of possible updates per period.

The first phase proceeds as follows. First, the generator determines the

[2]There might be an issue with this, look more into it.

start of the current period, named period_start, by computing

$$\texttt{period\_start} = \texttt{current\_time} - (\texttt{PERIOD\_LENGTH}\ \%\ \texttt{current\_time}).$$

It then produces an array of alias_update_scalars. This is done by hashing

$$\texttt{timing\_digest} = SHA{-}256(\texttt{timing\_secret}, \texttt{period\_start}, \texttt{iteration\_number})$$

where iteration_number is incremented for each hash necessary to fill the array. Each digest gives a number of alias_update_scalars by taking successive strings of bits from the digest. The generator can now recursively produce a sequence of alias_update_times by

$$\begin{aligned}\texttt{alias\_update\_time\_1} = {} & \texttt{period\_start} + \texttt{MIN\_UPDATE} + \\ & \left(\frac{\texttt{alias\_update\_scalar\_1}}{\texttt{max\_alias\_update\_scalar}}\right)(\texttt{MAX\_UPDATE} - \texttt{MIN\_UPDATE}),\end{aligned}$$

$$\begin{aligned}\texttt{alias\_update\_time\_n} = {} & \texttt{period\_start} + \texttt{MIN\_UPDATE} + \\ & \left(\frac{\texttt{alias\_update\_scalar\_n-1}}{\texttt{max\_alias\_update\_scalar}}\right)(\texttt{MAX\_UPDATE} - \texttt{MIN\_UPDATE}),\end{aligned}$$

where max_alias_update_scalar is the largest possible scalar (ie. for 16-bit scalars, this is 0xFFFF). The largest alias_update_time that does not exceed current_time is taken to be the last_update_time, and this concludes the first phase of the generator.

Note that there is one special case: If current_time is less than alias_update_time_1, then the first phase must be repeated using the previous period. This is a relatively rare corner case since it will only happen when a phone tried to make a call in the first few minutes of a period, and it can be detected quickly since alias_update_time_1 only depends on alias_scalar_1.

The second phase of the generator consists of a single step: the generator produces current_pseudonym by:

$$\texttt{current\_pseudonym} = SHA - 256(\texttt{last\_update\_time}, \texttt{ID\_secret}).$$

1 Discussion

We have some flexibility in choosing values for the five parameters. For the prototype network, I set the following parameter values:

$$\begin{aligned}
\texttt{GLOBAL\_MAX\_UPDATE} &= \texttt{MAX\_UPDATE} = 10 \text{ minutes},\\
\texttt{MIN\_UPDATE} &= 1 \text{ minute},\\
\texttt{PERIOD\_LENGTH} &= 1 \text{ day},\\
\texttt{UPDATE\_GRANULARITY} &= 1 \text{ second}.
\end{aligned}$$

The number of `alias_update_scalars` that the generator must produce is based on `PERIOD_LENGTH` and `MIN_UPDATE` since there must be at least enough scalars to fill a period even if every update happens after `MIN_UPDATE` milliseconds. If a user wishes to set `MIN_UPDATE` to 0, then the generator will need to produce scalars until it finds an `alias_update_time` that exceeds `current_time`. My current implementation does not operate this way, but this would be an easy modification.

The number of bits needed per scalar is determined by the `UPDATE_GRANULARITY` and the difference between `MAX_UPDATE` and `MIN_UPDATE`. Using the above parameter values, scalars are 16 bit values, and at most 90 hashes must be computed to find a pseudonym[3]. On a Nexus S Android phone, an unoptimized generator consistently computes an alias in less than a tenth of a second, so even performing hundreds of hashes at call-time will be unnoticeable to a user.

[3]The true worst case is 92 hashes if `current_time` falls at the very beginning of a period.