# Notes on the implication of dishonest witnesses

Niels Möller

2023-12-04

## 1 Introduction

These notes document the implications of dishonest witnesses. The objective is to find under which conditions a split-view attack can succeed, if the log and some of the witnesses are colluding. Or conversely, assuming a $k$-of-$n$ policy is used, specify how many honest witnesses are required.

A split-view attack is successful if the log can give two parties two distinct and inconsistent tree heads, and have both tree heads accepted.

An honest witness is assumed to cosign any tree head the log sends it, as long as that is consistent with previously cosigned tree heads. And the only property needed for this note is that an honest witness will never cosign two inconsistent tree heads. We assume that the attacker chooses which view of the log each honest witness cosigns.

We assume a compromised or dishonest witness is willing to sign any tree heads chosen by the attacker, and then cosignatures on different inconsistent tree heads can be passed to different parties. This setup is one kind of Byzantine fault tolerance.

## 2 Threshold policy

Assume we have two parties, e.g, a Sigsum monitor and a Sigsum verifier, using the same list of $n$ witnesses, and where policy requires at least $k_1$ and $k_2$ valid cosignatures, respectively.

First note that we must require $k_1 + k_2 > n$, otherwise we are vulnerable to split view attack even if all witnesses are honest.

If we assume at most $f$ of the witnesses are compromised, there can be up to $n + f$ cosignatures in total on the two views, where the compromised witnesses sign both views, and the attacker chooses which view is signed by each of the honest witnesses. We then have split view protection provided that

$$k_1 + k_2 > n + f. \tag{1}$$

To make the policies easy to understand, we could use symmetric policy where $k_1 = k_2 = k$, with a corresponding assumption that at least $k$ witnesses are honest. Then $f = n - k$, and the condition boils down to

$$k > 2n/3. \tag{2}$$

Readers familiar with byzantine fault tolerance will recognize this number. For those of us that are not familiar with that problem, the condition may appear surprisingly strict. E.g, the smallest configuration of this type that can tolerate a single compromised witness is 3-of-4, and the smallest that can tolerate 2 compromised witnesses is 5-of-7.

So with a small number of witnesses, we need rather high availability.

## 3 The case for asymmetric policy

When one of the parties is a verifier, with threshold $k_1$, and the other is a monitor, with threshold $k_2$, it may be beneficial with $k_2 > k_1$. The reason is that witnesses can be temporarily unavailable, so even if a witness is healthy it may not be able to cosign each and every tree head published by a log. Maybe for some reason it cosigns only every third tree head. A verifier only sees one tree head at a time, and is therefore likely to miss some cosignatures due to such temporary issues.

A monitor, on the other hand, repeatedly requests the log's tree head, and checks consistency between successive tree heads. It can keep track of the time-stamp of the most recent cosignature seen for each witness.

E.g., assume that the monitor has a 12 hour window before alerting on missing cosignatures. That means that at each point it time, it can count how many of the witnesses have cosigned a tree head in the last 12 hours. Since it also checks consistency, it can consider the tree head from 12 hours ago as cosigned by all those witnesses, and if the number is at least $k_2$, the monitor is happy and doesn't not raise any alert. In this way, the monitor is shielded from temporary problems, or witnesses that for some reason cosigns only some of the tree heads.

We can make use of this monitor advantage to define a *dual policy.* Assume that the verifier uses a $k_1$-of-$n$ policy, and the intention is to tolerate up to $f$ compromised witnesses (we must have $f < k_1$). If we apply (1) to this case, we find $k_2 > n + f - k_1$. The dual policy to be used by the monitor would then be

$$k_2 = n + f + 1 - k_1 \tag{3}$$

Unfortunately, this depends on $f$ which must in some way be specified by the user. It's not so user friendly to have a $f$-of-$k$-of-$n$ policy which would mean that we have $n$ witnesses, require $k$ valid cosignatures when verifying a Sigsum proof, and want to allow for up to $f$ compromised witnesses.

It's tempting to derive $f$ from $k_1$ by some fix rule. But, e.g., if we set $f = k_1 - 1$, then that leads to $k_2 = n$, i.e., monitor must alert if any cosignature is missing over its observation period. Such a configuration could perhaps be useful in some cases, e.g., if one can easily revoke or detrust witnesses from verifier policy if they appear to go offline, but doesn't look very useful for the general case.

Another alternative could be to set $f = \lceil k_1/2 \rceil - 1$, motivated by the byzantine fault tolerance condition when $k \approx 2n/3$. With this rule, we get

$$k_2 > n + \lceil k_1/2 \rceil - 1 - k_1 = n - 1 - \lfloor k_1/2 \rfloor \tag{4}$$

We should also require that $k_2 \geq k_1$. The dual policy would then be

$$k_2 = \max(k_1, n - \lfloor k_1/2 \rfloor) \tag{5}$$

| policy | dual | $f$, # compromised witnesses |
|---|---|---|
| 3-of-n | $(n-1)$-of-$n$ | 1 |
| 4-of-n | $(n-2)$-of-$n$ | 1 |
| 5-of-n | $(n-2)$-of-$n$ | 2 |
| 6-of-n | $(n-3)$-of-$n$ | 2 |
| 7-of-n | $(n-3)$-of-$n$ | 3 |
| $\vdots$ | $\vdots$ | $\vdots$ |

Figure 1: Dual policies

This give a family of reasonable pairs of policy and dual policy, with some examples in Figure 1. Maybe this is good enough to document and apply automatically when monitoring, to not require users to specify $f$ explicitly?