Consider a Lamport signature scheme where there are 256 participants, and each participant is required to reveal one of eight values that they have committed to (ie. 128 bytes total for a Merkle branch). Given a value with hash H (assume a 768 bit hash), take 3 bits of H and assign them to each user. That user is required to provide the specific value that's assigned to by the 3 hash bits. Any signature containing at least 170 valid preimages+Merkle branches passes as valid.

If you actually have 170 honest participants, then it's easy to see how the scheme can succeed. What if one signature is made (with all 256 participants) and you want to forge it? Then, in expectation, only 32 fitting values will be available, so to make a valid signature for a given random specified H' half the time you need ~158 participants to be colluding (of the ~98 non-colluding, ~12 will on average be available, getting you to 170). If you have 96 colluding participants, then you will only be able to make a signature for ~1 in 2^80 of possible values (ie. cryptographically infeasible, especially if a few rounds of PoW are done on H before using it as a source of query bits).

Hence, the size of a quantum-secure threshold signature is only ~96 bytes per participant for a committee of size 256, if we are willing to accept the ~1/3 slippage in safety (ie. safety reduces from 2/3 to 1/3); slippage reduces to ~1/5 if we increase from 96 bytes to 128 bytes, and to ~1/7 if we go up to 160 bytes per participant.

**Edit (2018.05.28): Hash ladder sigs**

We can trade off size for verification time by using a hash ladder sig instead of a regular sig. Each participant generates two values, x_1

and x_2

, and commits to H^n(x_1)

and H^n(x_2)

(eg. n = 16). To make a signature, take log(n) bits from the message hash, call this value d, and compute s_1 = H^d(x_1)

and s_2 = H^{n-d}(x_2)

, and output these values. Checking the signature involves taking H^{n-d}(s_1)

and H^n(s_2)

and checking that these values match the commitments. Sigs of this form are always 64 bytes; if we set n = 32

(leading to a sig verification time of 32 microseconds if we use blake2, which takes ~1 microsecond per round), then we get the same ~1/7 slippage.

A hybrid approach would be 96-byte signatures where we extract two values d_1

and d_2

from the message hash and put H^{d_1}(x_1)

, H^{d_2}(x_2)

, H^{2n - d_1 - d_2}(x_3)

in the signature; this could allow us only 8% slippage with 32 hashes to verify a signature, or 6% slippage with 64 hashes to verify a signature.

Note that slippage becomes much lower for larger sets; for an infinite sized set, slippage approaches \frac{2}{3N}

for 64-byte signatures and \frac{2}{3N^2}

for 96-byte signatures.