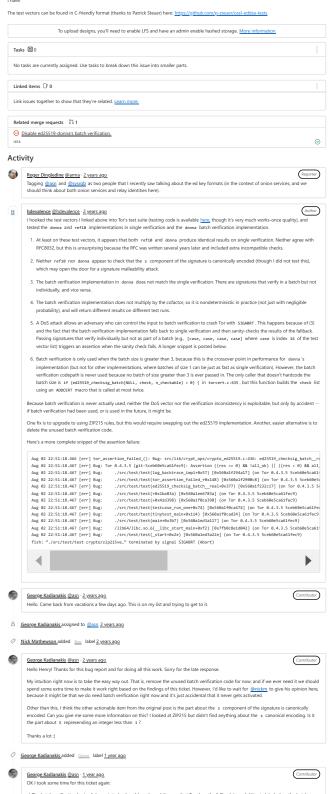
Potential consensus divergence from Ed25519 edge cases

Ed25519 poses risks in consensus-critical applications, because (a) the spec does not require that implementations agree on whether signatures are valid and (b) in practice, implementations differ from the spec and from each other.

In the context of working to address this issue in Zcash (resulting in Z<u>[[]215</u>). I created a set of 196 test vectors, consisting of hex-encoded (public key, signature) pairs on the message be "Zcash". Running these test vectors across various other Ed25519 implementations reveals a wide divergence in behaviour (see]. 2 for additional context

From a quick look at the Tor source and some tips from Teor, it looks like Tor has four different verification codepaths: ref10 open, ref10 open, batch, donna open, batch. But I'm not entirely sure whether these are all used, because that requires a deeper knowledge of the codebase than the codebase that the codebase



b) Henry's point (2) is also valid. We are indeed not doing the right check to see if the received 5 is canonical. In particular, ed2519-donna does the weak check for 5 canonicity (checking just the three most-significant bits) as described in the section. Checking for non-canonical 5 of 1 this range;

verification check falls) in existsis _checksis_batch() can indeed cause crashes. The good thing is that the batch verification is never actually called (it's only enabled if four or more sigs are passed which never happens). Fixing the math would be a pretty big move since upstream is also broken, so logited for removing the batching functionality. Using a proper library like ed2519-dialek in arti would allow us to do batch verification again. See Henry's excellent <u>bloop post</u> for more details.

See Lideced for the fix commit. I also pushed https://pitlab.toproject.org/ags/tor/.tree/hup40078-test-vectors which makes sure that the remaining ZP125 test vectors don't break tor. It's up to us whether we want to include the test vectors in tor. I saw that OpenSSL did not include them so I took the easy way out myself.

Our check will catch most non-canonical signatures but there are a bunch of them that can escape. In particular, the paper says that for honestly generoted signatures, the probability that the fourth most significant bit (ESCIA bit) is set is very smoll, roughly 1/2*1/128). This means that an attacker can craft signatures that are malleable (for each such evil signature, there is another one that verifies). And also there is an extremely small chance that non-attacker signatures can be malleable. I'm not sure what are the implications of e425519 signature malleability in our protocol. Certainly one can go around replay-caches, what other attacks are there? Fising this would require us to implement the full canonicity check as described by that paper. I didn't find any other C codebase that implements the full canonicity check (see Table 5 on the paper) so I opted to not try to do it at this point.

