[75] J. Plank and L. Xu. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. *NCA 2006.*
[76] PyECLib. https://pypi.python.org/pypi/PyECLib
[77] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.
[78] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Industrial Appl. Math*, 1960.
[79] M. K. Reiter and K. Birman. How to securely replicate services. *ACM TOPLAS*, vol. 16 issue 3, pp. 986–1009, ACM, 1994.
[80] M. Sathiamoorthy. et al. XORing elephants: novel erasure codes for big data. *Journal Proceedings of the VLDB Endowment* volume 6, issue 5, pp. 325–336, 2013.
[81] F. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surveys* 22(4): 299–319, 1990.
[82] V. Shoup. On fast and provably secure message authentication based on universal hashing. *CRYPTO '96*, pages 313–328, 1996.
[83] V. Shoup. Practical threshold signatures. *EUROCRYPT 2000.*
[84] V. Shoup. NTL: A library for doing number theory. http://shoup.net/ntl
[85] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *EUROCRYPT '98.*
[86] SingularityNET. https://singularitynet.io/
[87] J. Sousa, A. Bessani, and M. Vukolic. A Byzantine fault-tolerant ordering service for the Hyperledger Fabric blockchain platform. *DSN 2018.*
[88] Tendermint core. https://github.com/tendermint/tendermint
[89] H. Turki, F. Salgado, J. M. Camacho. HoneyLedgerBFT: Enabling Byzantine fault tolerance for the Hyperledger platform. Available: https://www.semanticscholar.org/
[90] R. van Renesse, C. Ho, and N. Schiper. Byzantine chain replication. *OPODIS 2012.*
[91] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung. Spin one's wheels? Byzantine fault tolerance with a spinning primary. *SRDS 2009.*
[92] Walmart, JD.com, IBM and Tsinghua University Launch a Blockchain Food Safety Alliance in China. https://www-03.ibm.com/press/us/en/pressrelease/53487.wss
[93] Z. Wilcox-O'Hearn. Zfec 1.5.2. https://pypi.python.org/pypi/zfec
[94] L. Zhou, F. B. Schneider, R. van Renesse. APSS: proactive secret sharing in asynchronous systems. *ACM Trans. Inf. Syst. Secur.* 8(3): 259–286 (2005)

A CORRECTNESS PROOF

Proof of Theorem 9.2. Termination is simple, as in AVID-FP. If a correct server initiates disperse, the server erasures codes the transaction, and sends fragments and the fingerprinted cross-checksum to each server. As the server initiating disperse is correct, at least $n - f \geq m + g_0(L-1) + f$ correct servers receive disperse messages, and send echo messages to all servers. Each server will eventually receive $m + g_0(L-1) + f$ echo messages, and then sends a ready message, if it has not done so. Each correct server will eventually receive at lest $2f + 1$ ready messages, and will then store the fingerprinted cross-checksum and complete.

Agreement follows *exactly* as in AVID-FP. If some correct server completes disperse(M), then the server must have received $2f + 1$ ready messages and at least $f + 1$ ready messages much have come from correct servers. This means that all correct servers will eventually receive ready messages from these correct servers. As our protocol implements the amplification step as in all other Bracha's broadcast like broadcast, all correct servers will send ready messages, and all of them will eventually receive at least $2f + 1$ ready messages. Agreement thus follows.

We first prove the first part of availability. In our protocol, if a correct server completes disperse, it must have received $2f+1$ ready messages, and at least one correct server received $m + g_0(L-1) + f$ echo messages. Therefore, at least $m + g_0(L-1)$ correct servers stored consistent fragments. According to the property of pyramid codes, these fragments can be used to reconstruct the original block. Accordingly, if $f + 1$ correct servers complete disperse, any client that initiates retrieve will receive $m + g_0(L-1)$ consistent fragments and $f + 1$ matching fingerprinted cross-checksums. The client can then decode the fragments to generate some block.

We now prove the second part of availability. Following an analogous line of the above argument, if a correct server completes disperse, at least $m + g_0(L-1)$ correct servers stored consistent fragments. If $f + 1$ correct servers complete disperse, any client that initiates read(i) for $i \in [1..m]$ will receive $f + 1$ matching fingerprinted cross-checksums. If the fragment i happens to be available or there is less than g_0 failures in the local group, the fragment will be available for the client. Otherwise, another round of interaction is needed, and the client will obtain $m + g_0(L-1)$ consistent fragments and reconstruct the fragment needed.

We now prove correctness. We first claim that if some correct server delivers fpcc_1 and some correct server delivers fpcc_2, then $\text{fpcc}_1 = \text{fpcc}_2$. The proof is quite "standard" for a quorum based protocol: if fpcc_1 is delivered then $m + g_0(L-1) + f$ servers echoed fpcc_1, of which at least $m + g_0(L-1)$ is correct. The same applied to fpcc_2. As a correct server will only echo once, there are at least $2m + 2g_0(L-1) + f$ servers echoed, which is larger than the total server (note that $L \geq 2$ and $2f \leq (g_0 + g_1)$). This leads to a contradiction. Thus, any block decoded during retrieve or any fragment during read is consistent with the same fpcc. By Theorem 3.4 in [45] and by Lemma 9.1, the probability that clients do not obtain the same block or fragment(s) is negligible. ∎