PGV: A Storage Enforcing Remote Verification Scheme

Mohammad Iftekhar Husain, Steve Uurtamo, Steven Y. Ko, Atri Rudra, Ramalingam Sridhar
Dept. of Computer Science and Engineering
University at Buffalo, The State University of New York, Buffalo, NY
Email: {imhusain, uurtamo, stevko, atri, rsridhar}@buffalo.edu

***Abstract*—This paper presents a storage enforcing remote verification scheme, PGV (Pretty Good Verification). While existing schemes are often developed to handle a malicious adversarial model, we argue that such a model is often too strong of an assumption, resulting in over-engineered, resource-intensive mechanisms. Instead, the storage enforcement property of PGV aims at removing a practical incentive for a storage server to cheat in order to save on storage space in a covert adversarial model.**

At its core, PGV relies on the well-known polynomial hash; we show that the polynomial hash provably possesses the storage enforcement property and is also efficient in terms of performance. In addition to the traditional application of a client verifying the storage content at a remote server, PGV can also be applied to de-duplication scenarios where the server wants to verify whether the client possesses a significant amount of information about a file (and not just a partial knowledge/fingerprint of the file) before granting access to an existing file.

We theoretically prove the power of PGV by combining Kolmogorov complexity and list decoding and experimentally show the simplicity and low overhead of PGV by comparing it with existing schemes. Altogether, PGV provides a good, practical way to perform storage enforcing remote verification.

***Keywords*-storage enforcement, proof of retrievability, cloud storage, proof of data possession, proof of ownership**

I. Introduction

A general structure of the remote storage verification problem is the following: a verifier (client) (i) uploads its data to a remote prover (cloud storage), (ii) deletes the local copy (to save on storage), and (iii) at some later point, tries to verify if the prover is storing the data correctly i.e. assurance of integrity and retrievability. Existing PDP or POR schemes [1], [2], [3], [4], [5], [6], [7] consider a malicious adversarial model [8], which translates to assuming that legitimate cloud storage providers such as Amazon will behave arbitrarily to tamper with client's data. In order to satisfy this strict adversarial model, existing schemes are complex (including modification of original data [2], [6], [3]) and performance inefficient (both in time and space requirements [1], [3], [4], [7]).

The primary motivation for this paper is to show that if we relax this strict adversarial model and focus instead on a more practical adversarial model, we can develop a simpler, far more light-weight verification scheme. Therefore, instead of considering a malicious adversarial model, we consider a covert adversarial model [9]. This model assumes that the adversary is willing to cheat if: (i) it has some incentive, and (ii) it will not be caught. It nicely captures many real world scenarios including remote storage verification where there is a practical incentive for a provider to cheat in order to save on storage as long as it is not caught. This is more practical since storage is the main commodity for storage providers—a storage provider typically charges its clients primarily for the amount of storage that each client uses. Also, the provider incurs a significant amount of cost in handling and managing the storage for its clients such as the costs for hard drives, storage area networking, and power consumption [10], [11]. Thus, there is a practical incentive for a provider to cheat in order to save on storage.

The main contribution of this paper is a *storage enforcing remote verification* scheme, PGV (Pretty Good Verification). This means roughly that, in order to pass our verification, a prover has to commit as much storage space as the information content of the original data. This removes storage saving as an incentive of cheating for the prover. More specifically, if the prover passes our verification of the original data x with probability $\varepsilon > 0$, then the prover has to store $C(x)$ bits of data up to a very small additive factor. $C(x)$ is the plain Kolmogorov complexity of x, which is the size of the smallest algorithmic description of x ([12], [13]). The reason why we enforce $C(x)$ instead of x is because we cannot prevent a prover from compressing x; $C(x)$ is a natural way to represent the amount of information stored in x.

We also show that PGV has application in *proof of ownership* [14]. Consider the case when a client wants to upload its data x to a storage service (such as Dropbox) that uses deduplication. To save on communication, the server asks the client to send it a hash $h(x)$ and if it matches the hash of the stored x on the server, the client is issued an ownership of x. As

identified by [14], this simple deduplication scheme can be exploited for malicious purposes if the server doesn't verify the ownership. Since the prover and the verifier are reversed from the client verifying storage provider scenario, the performance restriction is even more severe. The computation that runs at the client-side has to be light-weight because of the limited capacity of client devices such as smartphones. Further, we cannot expect the client to modify the data to aid in the verification process (as it is needed in some existing PDP/POR schemes). Therefore, PGV can also handle such verification efficiently.

We have implemented our widely applicable *storage enforcing remote verification* scheme, PGV and evaluated through extensive experiments with data files of sizes 1 MB to 1 GB. Our experimental results demonstrate that PGV is significantly performance efficient compared to existing proof of data possession (PDP [1]) and proof of ownership (PoW [14]) schemes.

II. Pretty Good Verification

This section presents the general construction of our storage enforcing remote verification scheme, PGV and outline of the guarantees it provides.

A. Basic Scheme

The scheme has two main phases: *pre-processing and hash generation* and *challenge generation and verification* phase.

Pre-processing and hash generation: To compute a polynomial hash, a verifier needs to pick system-wide parameters once and perform three steps in the pre-processing phase. There are two important system-wide parameters: the finite field size q (e.g., 2^{32} or equivalently, 4B) and the block size (e.g., 64KB). Picking the right values for these parameters involves considerations for performance and security. We discuss these considerations in our evaluation in Section III.

Once the verifier finishes picking the system-wide parameters, the verifier can compute a polynomial hash for each data block: (i) pick a random number (our key) β from the field, (ii) divide the data block into equal-sized *symbols*, $S_0, \ldots, S_{k-1}$, where the size is equal to the field size (e.g., 4B), and (iii) compute the polynomial hash $H_c = \sum_{i=0}^{k-1} S_i \beta^i$. This is equivalent to computing a single entry in a Reed-Solomon code word.This process is per-block. For a file, there is one additional step that divides the file into blocks. Then the verifier needs to repeat the above three steps for each block. Once the polynomial hash is computed for each block, the verifier stores the random keys and hash values locally and uploads the file to remote storage.

Challenge generation and verification: To verify a data block in a storage enforcing fashion, the verifier sends the key β associated with that block to the remote storage. The remote storage compute the polynomial hash $H_c = \sum_{i=0}^{k-1} S_i \beta^i$ and sends back to the verifier. If this hash matches the locally stored hash by the verifier, the verification succeeds. By default, PGV is designed to perform complete verification of the remote storage. However, PGV also supports a probabilistic framework of sampling similar to existing schemes such as PDP which is detailed in Section II-C. The idea of sampling is to partially verify the data and get a probabilistic guarantee instead of complete verification.

B. Summary of Our Guarantees

We can summarize our two main results for remote verification as follows:

- If the prover can pass our verification protocol with a probability greater than ε (where ε can be made to be very small, e.g. 1%), then the prover must provably store almost as much as the Kolmogorov complexity of the user string. Another way to state this is that, if, for example, $\varepsilon = 1\%$ and the prover stores slightly less than Kolmogorov complexity $C(x)$, then the prover will be caught with probability at least 99%.
- If the prover can pass our verification protocol with a probability greater than $1/2+\gamma$ (where γ can be very small, e.g. 1%), then the prover has enough information to recreate the user string. Another way to state this is that, if, for example, $\gamma = 1\%$ and the prover cannot recreate the user data, then it will be caught with probability at least 49%.

The proofs take advantage of the fact that a polynomial hash is just a single entry in an appropriately-chosen Reed-Solomon code word. The proofs and the scheme itself in fact generalize to any error-correcting code and corresponding universal hash function the details of which can be found in [15].

C. Sampling

Although PGV prefers and is able to efficiently handle complete verification of the data (i.e. all the data blocks in a file), PDP uses sampling to scale with the increment in file sizes. To make a fair comparison, we now discuss a probabilistic framework of error detection using PGV. Let's assume a scenario where the server cheats[1] t blocks out of an n-block file. Let, r_{PDP} and

[1]Cheating in the case of PGV means that the server stores less than the Kolmogorov complexity of the block. This e.g. can handle the case when sufficiently bits from a block has been erased. For PDP this means that the server alters the block so that it cannot re-create the original block.

r_{PGV} be the number of sampled blocks during the verification in PDP and PGV, respectively. Let p be the probability that the server can avoid detection of its cheating. For PDP, p is the probability that none of the sampled blocks match the blocks on which the server cheats; i.e., we have: $p = (1 - \frac{t}{n})^{r_{PDP}}$ which leads to $r_{PDP} = (\frac{\log \frac{1}{p}}{\log \frac{n}{n-t}})$. For PGV, p will have two components, one component is due to the sampling error and another component is the verification error ε (we get the latter from Corollary 18 in [15]). So, $p = (1 - \frac{t}{n})^{r_{PGV}} + \varepsilon$. If we use weight $0 \leqslant \alpha \leqslant 1$ as a factor of p for these two components, we have: $(1-\alpha)p = (1 - \frac{t}{n})^{r_{PGV}}$ and $\alpha p = \varepsilon$. The first term gives us: $r_{PGV} = (\frac{\log \frac{1}{(1-\alpha)p}}{\log \frac{n}{n-t}})$ which is equivalent to $r_{PGV} = (r_{PDP} + \frac{\log \frac{1}{(1-\alpha)}}{\log \frac{n}{n-t}})$. When $t = 1\%$ of n, if we want a 99% detection rate (i.e. $p = 1\%$), and $\alpha = \frac{8}{10}$, we get: $r_{PGV} = (r_{PDP} + 23)$ blocks. Now, from Corollary 18 in [15], we have $\varepsilon \leqslant \sqrt{\frac{k}{q}}$, or $k \leqslant \varepsilon^2 q$ where q is the number of field elements and k is the number of symbols in each block. In turn, this bounds the block size for PGV to be $b \leqslant \log_8 q\varepsilon^2 q$ bytes. When we use $GF(2^{32})$, if we want a 99% detection rate ($p = 1\%$), we get $b = 1677\alpha^2 KB$ since $\alpha p = \varepsilon$. For $\alpha = \frac{8}{10}$, the block size can be as large as $1.04MB$.

D. Repeatability

PGV can overcome the bounded repeatability by piggybacking the generation of new hashs with regular block access operations. To make the discussion more concrete, let us compare PDP (which has unbounded repeatability) and PGV (with the piggybacking scheme). In particular let (g_{PDP}, v_{PDP}) and (g_{PGV}, v_{PGV}) be the generation time and verification time for one block for PDP and PGV respectively. Further define $R_g = \frac{g_{PDP}}{g_{PGV}}$. Let B be the maximum number of hashs that the PGV keeps at any point of time per block. Since PDP has unbounded repeatability, it only needs to store one hash per block. Next we try to figure out what B should be in order to keep the overall overhead of PGV still smaller than PDP.

At the beginning both PDP and PGV need to generate the hashs. Thus, PGV spends $B \cdot g_{PGV}$ time while PDP spends g_{PDP} time. Thus, to be ahead of PDP at this stage, we need to make sure

$$B \leqslant \frac{g_{PDP}}{g_{PGV}} = R_g. \tag{1}$$

Now later on, we will have verification for the block interspersed with normal reads for the block. Let us assume that $C > 0$ is the maximum number of verification calls between two normal reads for a block. Now consider any "run" of x consecutive verification calls followed by a normal read (so we have $x \leqslant C$). Note that in this case for PGV we (i) need to make sure that $B \geqslant x$ as we cannot "replenish" the buffer till get to the normal read and (ii) make sure that the overhead incurred by PGV is smaller than PDP. For (ii) we note that the time expended by PDP is $x \cdot v_{PDP}$ while the time expended by PGV is $x(v_{PGV} + g_{PGV})$ (as it needs to perform x verifications and replenish as many hashs in its buffer). Thus, PGV would have a lower overhead if

$$v_{PGV} + g_{PGV} \leqslant v_{PDP} \tag{2}$$

Note that we have ended up with the following constraints on B:

$$C \leqslant B \leqslant R_g.$$

In our experiments (Section III), we have found that (2) is satisfied. (In fact, we observe that $g_{PGV} \geqslant v_{PGV}$. Further, $v_{PDP} \geqslant 10 \cdot g_{PGV}$, which implies that PDP is at least five times as slow as PGV with B hashs during the verification phase.) Finally, we observe in our experiments that $R_g \geqslant 10$. So if we pick $B = C$ and if we have $C < 10$ (the latter is a realistic assumption for the cloud computing applications we envision for PGV), then PGV comes out ahead even in the hash generation part.

III. Client Verifying Storage Provider Experiments

This section presents various performance aspects of PGV as well as comparisons with the performance of a crypto-based scheme, PDP [1]. We emphasize that we have only considered the main schemes in PDP, although there are many variants of the main schemes. Since the variants of each main scheme make design tradeoffs on multiple properties, it is difficult for us to consider those variants altogether.

A. Experimental Platform and Parameters

We use open source C libraries on an Intel Centrino Duo machine at 1.73 GHz with 2.5 GB memory running Linux kernel 2.6.32 to compare PGV with existing schemes. We use a single thread in all implementation and measurements. For cryptographic operations, we use OpenSSL 0.9.8k. For field operations and erasure coding, we use James Plank's Galois field library [16] and Jerasure library [17].

B. Hash Generation Time

For PDP, the hash generation time includes the time required to generate the asymmetric keys, file I/O and per-block hash generation time. Figure 4 shows the hash

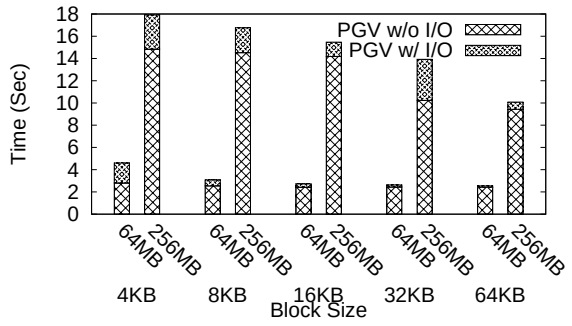

Figure 1. hash generation time for PGV.

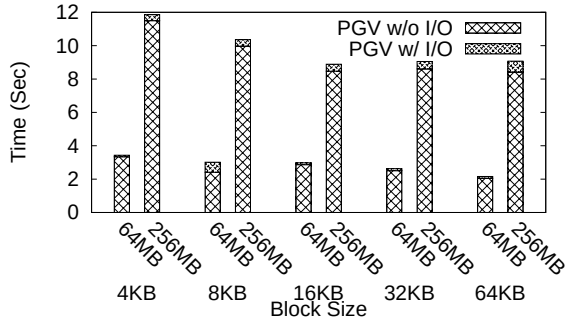

Figure 2. Challenge generation and verification time for PGV.

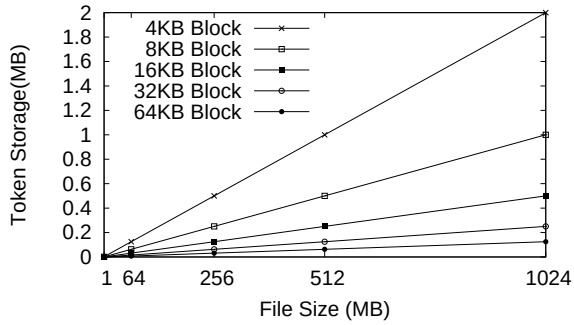

Figure 3. Hash storage for PGV for different file sizes.

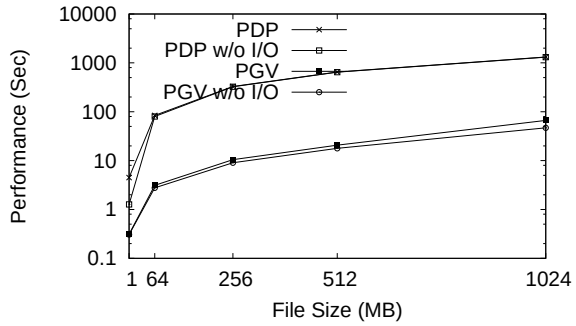

Figure 4. Preprocessing and hash generation time comparison for PDP and PGV

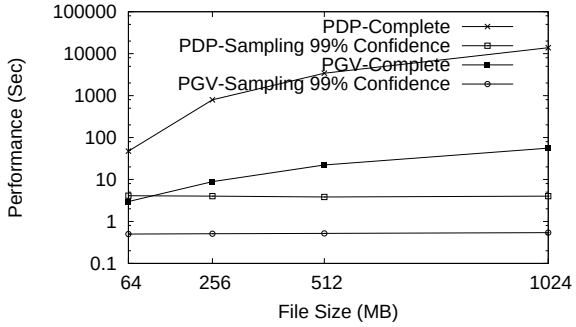

Figure 5. Challenge generation and verification time comparison for PDP and PGV

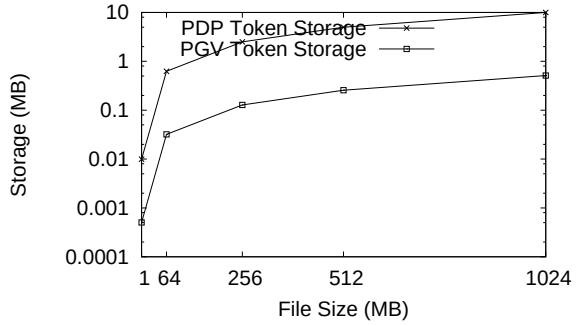

Figure 6. Storage overhead comparison of PDP and PGV

generation time comparison between PDP and PGV. (For both schemes we used a block size of 16K.) As shown in the figure, PGV has little overhead as it just evaluates the polynomial hash. PDP has higher overhead as it relies on computationally intensive cryptographic primitives. It uses asymmetric cryptographic primitive based on modular exponentiation. On average, PGV is approximately 25 times faster than PDP in hash generation for different files sizes.

C. Challenge Generation, Verification, and Sampling Analysis

Figure 5 compares PDP verification time with PGV considering both completeness (i.e. in both cases we verify all the blocks) and sampling (Section II-C with the following setting of parameters: $t/n = 1\%$, $p = 1\%$, $\alpha = .8$ and $q = 2^{32}$). PDP incurs the most overhead due to asymmetric cryptographic operations in all three phases. It also shows the sampling verification time with 99% detection rate. As discussed earlier, in the probabilistic framework, PDP can guarantee 99% detection

rate by sampling 460 blocks. For PGV, it can guarantee the same detection rate by sampling 483 blocks. As in hash generation, PGV outperforms PDP since it involves only a number of simple finite field additions and multiplications. On average, PGV verification is almost 120 times faster than PDP verification.

D. Storage overhead

Figure 3 shows the PGV storage overhead for different file sizes and block sizes. Comparison of storage overhead with PDP is shown in Figure 6. PDP stores its tokens (authenticators) on the server side. These hashs are usually 1024 bits long. Unlike these schemes, PGV does not have any server-side storage overhead. We only need to store hashs at the client side.

IV. Server Verifying Client Experiments

In this section, we compare PGV with PoW [14] in terms of client time (the time required by the prover to compute the proof of ownership of deduplicated data). We also compare both of the schemes' network time to a naive approach where, instead of deduplication, the complete file is transfered over the network.

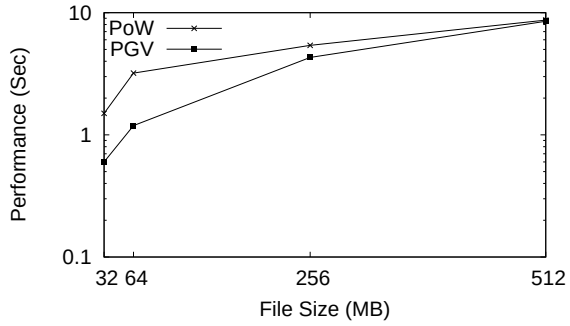

Figure 7. Client time comparison for PoW and PGV.

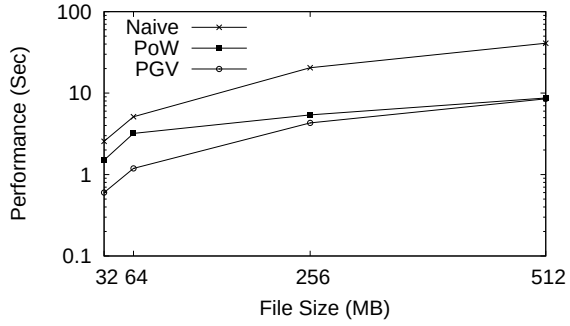

Figure 8. Overall time comparison (client time + network time) for PoW and PGV with a naive approach with no deduplication over a 100 Mbps network.

A. Client time

Figure 7 compares client times for PGV and PoW for different file sizes. As we can see from the figure, PGV performs faster than PoW in client time for the reported file sizes from 32MB to 512MB. However, as the file size increases, the performance follows similar trend. For instance, PoW and PGV takes 15.29 sec and 16.98 sec, respectively for a 1024 MB file. However, note that PoW is optimized only for proof of ownership whereas PGV supports two-way verification with proof of storage enforcement.

B. Network time

Network time in PGV corresponds to sending random numbers β to the client and get the computed hashes back from the client. As we have discussed in Section III-C, PGV can guarantee 99% detection by checking 483 blocks. So, in total PGV transfers around 30K data on the network as a part of the protocol data. For similar guarantee, PoW reports 20K data transfer requirement as it needs to transfer 20 leaves to the provers. Figure 8 shows the overall time (client time and network time) required by PGV and PoW, compared to a naive scheme which always transfers the complete file over the network without deduplication for different file sizes. Our results show that PGV performs better than both PoW and naive approach. Note that, both PGV and PoW will require additional time to verify the hashes which is 0.5 sec for PGV and 0.6 ms for PoW.

V. Related Work

In contrast with our coding based storage enforcement scheme, Golle et al. [3] had proposed a cryptographic primitive called storage enforcing commitment (SEC) which probabilistically guarantees that the server is using storage whose size is equal to the size of the original data to correctly answer the data possession queries. Dodis et al. [18] provided theoretical studies on different variants of existing POR scheme. Note that, one of variants in [18] is stucturally similar to poly hash but, we differ in adversarial modelling, storage enforcement property, proof technique and wide range of application demonstration with experimental results.

VI. Conclusions

This paper presents PGV (*P*retty *G*ood *V*erification), a multi-purpose storage enforcing remote verification scheme that utilizes polynomial hash for storage verification. In particular, its simplicity allows for bi-directional verification (i.e., client can verify that the server is storing its data and the server can verify that a (new) client that has data that it is claiming to have).

We prove using a novel combination of Kolmogorov complexity and list decoding that the polynomial hash provides a strong storage enforcement property as well as proof of retrievability. Our experimental results support our claim of low overhead in pre-processing, token generation, verification, and extra storage space. Our overall results show that it is a good, practical choice for multi-purpose remote storage verification.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, 2007.

[2] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: a High-Availability and Integrity Layer for Cloud Storage," in *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS 2009)*, 2009.

[3] P. Golle, S. Jarecki, and I. Mironov, "Cryptographic Primitives Enforcing Communication and Storage Complexity," in *Proceedings of the 6th international conference on Financial Cryptography (FC'02)*, 2002.

[4] A. Juels and B. S. K. Jr., "PORs: Proofs of Retrievability for Large Files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, 2007.

[5] T. Schwarz and E. L. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.

[6] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," in *Proceedings of the 17th IEEE International Workshop on Quality of Service (IWQoS 2009)*, 2009.

[7] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," in *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS 2009)*, 2009.

[8] R. Canetti, "Security and composition of cryptographic protocols: a tutorial (part i)," *SIGACT News*, vol. 37, no. 3, pp. 67–92, 2006.

[9] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," *J. Cryptol.*, vol. 23, no. 2, pp. 281–343, Apr. 2010.

[10] R. L. Moore, J. D'Aoust, R. Mcdonald, and D. Minor, "Disk and tape storage cost models," in *Archiving 2007*, 2007. [Online]. Available: http://www.imaging.org/conferences/archiving2007/details.cfm?pass=21

[11] M. Allalouf, Y. Arbitman, M. Factor, R. I. Kat, K. Meth, and D. Naor, "Storage modeling for power estimation," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 3:1–3:10.

[12] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 3rd ed., ser. Graduate Texts in Computer Science. New York, NY, USA: Springer, 2008.

[13] A. N. Kolmogorov, "Three Approaches to the Quantitative Definition of Information," *Problems of Information Transmission*, vol. 1, no. 1, pp. 1–7, 1965.

[14] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," Cryptology ePrint Archive, Report 2011/207, 2011, http://eprint.iacr.org/.

[15] M. I. Husain, S. Ko, A. Rudra, and S. Uurtamo, "Almost universal hash families are also storage enforcing," *CoRR*, vol. abs/1205.1462, 2012.

[16] J. S. Plank, "GFLIB C Procedures for Galois Field Arithmetic and Reed Solomon Coding," Website, 2003, http://web.eecs.utk.edu/~plank/plank/gflib/index.html.

[17] ——, "Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications," Website, 2008, http://web.eecs.utk.edu/~plank/plank/papers/CS-08-627.html.

[18] Y. Dodis, S. P. Vadhan, and D. Wichs, "Proofs of Retrievability via Hardness Amplification," in *Proceedings of the 6th Theory of Cryptography Conference (TCC)*, 2009.