GLASS: Towards Secure and Decentralized eGovernance Services using IPFS

Christos Chrysoulas[1][0000−0001−9817−003X], Amanda Thomson[1], Nikolaos Pitropakis[1][0000−0002−3392−9970], Pavlos Papadopoulos[1][0000−0001−5927−6026], Owen Lo[1], William J. Buchanan[1][0000−0003−0809−3523], George Domalis[2], Nikos Karacapilidis[2], Dimitris Tsakalidis[2], and Dimitris Tsolis[2]

[1] School of Computing, Edinburgh Napier University, Edinburgh, United Kingdom
[2] Computer Engineering and Informatics Department, University of Patras, Greece

Abstract. The continuously advancing digitization has provided answers to the bureaucratic problems faced by eGovernance services. This innovation led them to an era of automation, broadened the attack surface and made them a popular target for cyber attacks. eGovernance services utilize the internet, which is a location addressed system in which whoever controls its location controls not only the content itself but also the integrity and the access of that content. We propose GLASS, a decentralized solution that combines the InterPlanetary File System with Distributed Ledger Technology and Smart Contracts to secure eGovernance services. We also created a testbed environment where we measure the system's performance.

Keywords: eGovernance · Security · DLT · IPFS · DHT · Kademlia

1 Introduction

The rapid evolution of digital technologies, including mobile communications, cloud computing infrastructures, and distributed applications, has created an extended impact on society while also enabling the establishment of novel eGovernance models. The need for an inclusive eGovernance model with integrated multi-actor governance services is apparent and a key element towards a European Single Market. Digital Transformation of public services can remove existing digital and physical barriers, reduce administrative burdens, enhance governments' productivity, minimize the extra cost of traditional means to increase capacity, and eventually improve the overall quality of interactions with (and within) public administrations.

eGovernance includes novel and digital by default public services aiming for administrative efficiency and minimization of bureaucratic processes, enabling open government capabilities, behavior and professionalism, improved trust and confidence in governmental transactions. Towards the modernization of public services, public administrations need to transform their manual business flows and upgrade their existing internal processes and services.

However, the digitization of eGovernance services has also expanded the attack surface, thus making them attractive to malicious third parties. In 2017 the National Health Service of the United Kingdom suffered from the WannaCry ransomware, which resulted in missed appointments, deaths, and fiscal

costs [1]. Recently, in May 2021 the American oil pipeline system suffered a ransomware cyberattack that impacted all the computerized equipment managing the pipeline. The company paid a ransom of 75 Bitcoins, approximately \$5 million, to the hackers in exchange for a decryption tool which eventually proved so slow that Colonial's own backups were used to bring the system back to service [2].

As the need for privacy-preserving and secure solutions in eGovernance services is imminent, our decentralized solution, namely GLASS, moves towards that direction by examining the effectiveness and efficiency of distributed cutting edge technologies, demonstrating the capacity of a public, distributed infrastructure, based on the InterPlanetary File System (IPFS). Our contributions can be summarised as follows:

- We analyze the threat landscape in the context of an eGovernance use case.
- We create a distributed testbed environment based on IPFS and detail our methodology.
- We analyze and critically evaluate the runtime performance of our implementation.

The structure of the rest of the paper is organized as follows: Section 2 builds the background on distributed models and presents the related literature, while Section 3 details the GLASS architecture while briefly explaining the threat landscape in the context of an eGovernance services use case scenario. Section 4 consists of our methodology and implementation used to conduct the main experimental activity of our work, while Section 5 presents and evaluates the performance results of our experimental activity. Finally, Section 6 draws the conclusions, giving some pointers for future work.

2   Background and Related Literature

2.1   Kademlia

In 2001 Maymounkov and Mazières published Kademlia, a Distributed Hash Table (DHT) that offered multiple features that were currently not available simultaneously in any other DHT [3]. The paper introduced a novel XOR metric to calculate the distance between nodes in the key space and a node Id routing algorithm that enabled nodes to locate other nodes close to a given target key efficiently. The presented single routing algorithm was more optimal compared to other algorithms such as Pastry[4], Tapestry[5] and Plaxton[6] that all required secondary routing tables. Kademlia was outlined as easily optimised with a base other than 2 with no need for secondary routing tables. The k-bucket table was configured so as to approach the target b (initial implementation was b = 5) bits per hop. With one bucket being used for nodes within distance range of $[j2^{160-(i+1)b}, (j+1)2^{160-(i+1)b}]$ from the initial node for each $0 < j < 2^b$ and $0 \leq i < 160/b$ based on a SHA1 160 bit address space. At any point, it is expected that there would be no more than $(2^b - 1)log_{2^b}$ buckets with entries. The k-buckets were described as being resistant to certain DoS attacks [3] due to the inability to flood the system with new nodes, as Kademlia only inserts new nodes once old ones leave.

In 2008 Baumgart and Mies introduced S/Kademlia [7] which offered several further security enhancements designed to improve on the original specification.

They examined various attacks that peer-to-peer networks were vulnerable to and offered practical solutions to protect against them. The key attacks identified by them were: a) Eclipse Attack, b) Sybil Attack, and c) Adversarial Routing. In 2020 Prünster et al. [8] highlighted the need for further implementation of S/Kademlia mitigations by demonstrating an effective eclipse attack. They were able to generate a large number of ephemeral identities and poison multiple nodes routing tables for very little expense, and *CVE-2020-10937* was assigned to the demonstrated attack.

2.2   IPFS

The InterPlanetary File System (IPFS) is a distributed system based on a peer-to-peer protocol that provides public data storage services to transform the web into a new decentralized and more efficient tool. Its primary purpose is to replace the HTTP protocol for document transactions by solving HTTP's most limiting problems like availability, cost, and centralization of data in data centers.

IPFS is based on a Merkle Directed Acyclic Graph (DAG) [9], the data structure to keep track of the location its data chunks are stored and the correlation between them. Each data block has a unique content identifier (CID) fabricated by hashing its content in this peculiar data structure. In case the content of a node's child changes, the CID of the parent node changes as well. For someone to access a file, knowing its unique Content Identifier, constructed by the hash of the data contained within it, is essential. Each participating node (user) keeps a list of the CIDs it hosts in a Distributed Hash Table (DHT) implemented using the Kademlia protocol [10]. Each user "advertises" the CIDs they store in the DHT, resulting in a distributed "dictionary" used for looking up content. When a user tries to access a specific file, IPFS crawls the DHTs to locate the file by matching the unique content identifier. Using content-based addressing instead of location-based addressing serves in preventing saving duplicate files in the network and tracking down a file by its content rather than by its address.

IPFS enables its users to store and distribute data globally in a secure, resilient and efficient way. Each file uploaded on IPFS is fragmented into chunks of 256Kb and hashed before being scattered in participating nodes around the globe. Following the aforementioned methodology, data integrity is ensured since no one can tamper with a data block without affecting its unique hash. Furthermore, data resilience is ensured by placing the same data block in more than one participating node.

Mukne et al. [11] are using IPFS and Hyperledger Fabric augmented to perform secure documentation of land record management. Andreev and Daskalov [12] are using IPFS to keep students' personal information off-chain in a solution that manages students' data through blockchain. Singh [13] created an architecture for open government data where proof-of-concept uses Ethereum for decentralized processing and BigchainDB and IPFS for storage of large volumes of data and files, respectively.

2.3   Distributed ledger

A Distributed Ledger is a distributed database architecture that enables multiple members to maintain their own identical copy of information without the

need for validation from a central entity while ensuring data integrity. Transaction data are scattered among multiple nodes using the P2P protocol principles and are synchronized simultaneously in all nodes. By providing Identification Management through DLTs, it is ensured that the user has control of their identity records since the information is stored publicly on the ledger instead of the systems of a central authority. Furthermore, since editing information on past transactions on a blockchain system is not supported, protection against unauthorized alteration of the identity records is established. Finally, having a single record of identity information that the user can utilize on multiple occasions minimizes the data duplication on multiple databases [14]. The second generation of blockchain technologies introduced the smart contracts that act as mini-programs used to automate code deployment when some pre-defined terms are met.

Our solution, GLASS, combines the advantages of IPFS with those offered by the Distributed Ledgers and Smart Contracts, thus creating a distributed scalable and secure eGovernance infrastructure. Moving towards the first steps of our implementation, we create an IPFS based testbed environment and empirically evaluate its runtime performance.

3   Architecture

We propose a combination of IPFS with Distributed Ledger and Smart Contracts which are proven to be beneficial for recording massive volumes of transactions. Extracting helpful information efficiently has significant computational challenges, such as analysing, aggregating, visualising, and storing data collected in distributed ledgers. More specifically, the volume and velocity of the data make it difficult for typical algorithms to scale while querying the ledger might come at high computation costs. State-of-the-art efforts seek to introduce new models that deal with such large-scale, distributed data queries to reduce data volume transferred over the network via adaptive sampling that maintains certain accuracy guarantees [15]. As the ledgers (and thus the data) keep getting bigger, a challenge is to make sense of the collected data for the users and perform analytics leveraging big data processing engines (i.e., Spark) that can deliver results quickly and efficiently. In order to adequately protect data resources, it is paramount to encrypt data in such a way that no one other than intended parties should be able to get the original data. The current practice compared to our approach can be seen in Figure 1.

A simple use case presenting a European Union's citizen, Alice, getting a job abroad from Greece to another member state, Portugal, using GLASS ecosystem, is presented in Algorithm 1.

3.1   Threat Landscape

Distributed file systems, such as IPFS, need to solve several challenges related to the security and privacy of the stored data, the infrastructure's scalability, the decentralized applications and big data complexities. However, there is a number of promising solutions that aim to settle some of these hurdles.

Security and privacy challenges The key challenge of distributed file systems, including IPFS, is that when new peers participate in the system, they
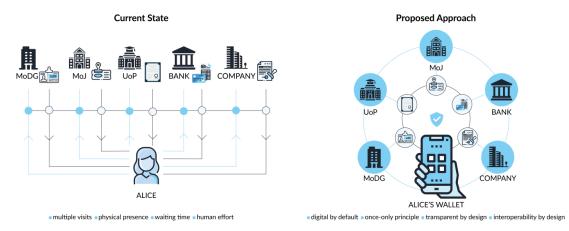
Fig. 1: Current practice compared to our approach [16]

can access any stored file, including sensitive documents. Hence, the security and privacy of the system remain an open question, especially due to General Data Protection Regulation (GDPR) [17] in the European Union. A prominent solution to that is the application of smart contract-based Access Control (AC) policies [18,19,20,21], and further encryption mechanisms [22].

Another security and privacy challenge is related to file erasure. By their nature, distributed file systems distribute all the stored files and documents among their participating peers. Hence, when data owners transmit "erasure commands" to the distributed network, it is not clear if all the peers would obey this command and delete their version of the "deleted" file or document. A solution to this data replication issue can be a common technique commonly present in data centers [23,24].

Scalability Challenges Since GLASS aims to create an eGovernance framework to be followed by all European Union's member states, the infrastructure's scalability poses a real threat. According to [25,26], one of the scalability issues on IPFS is the bandwidth limit in each IPFS instance due to the peer-to-peer nature of the system. Each participant needs to connect to another IPFS node to read or download the data objects. [27] proposed a combination of IPFS and blockchain technology, namely BlockIPFS, to improve the traceability of all the occurred access events on IPFS. The authors measured the latency of each event, such as storing, reading, downloading, by varying the number of IPFS nodes and presented that even incorporating large numbers of IPFS nodes does not significantly improve the latency of all the IPFS actions. However, the authors' experiments were limited to a maximum of 27 nodes; hence, the latency measurement on a vast scale remains an open question.

For the storage optimisation, two prominent solutions can be applied:

– **Storing data off-chain** [28,29,30]. The concept of utilising smart contracts off-chain and use IPFS as a storage database has been presented by some works. This solution is storage efficient since the IPFS nodes need to exchange only hash values of the data.
– **Utilize erasure codes** [31,32,33]. In erasure codes, a file is divided into smaller batches and these batches are encoded. Following that, each batch

---

Algorithm 1 Alice getting a job to Portugal

1: Starting from Greece, Alice finds a vacant job position in Portugal. She applies for the job, and thankfully she gets hired.
2: In Portugal, she has to deal with a series of bureaucratic processes (ID card, social security number, open a bank account).
3: To obtain a Portuguese Residence title, rent an apartment and open a bank account, Alice needs to present at least a validated ID documentation, birth certificate, nationality certification validated by a Greek Authority and proof that she works in Portugal.
4: Adopting the GLASS solution, Alice can request the proof of ID and the validated data from the Ministry of Digital Governance (MoDG).
5: The MoDG can issue the document, and after Alice's permission, the document can be forwarded to the Ministry of Justice (MoJ).
6: After this transaction is completed, Alice can access and securely share her Portuguese social security number through her Wallet.
7: Then Alice's employer in Portugal can directly get the validated social security number from the MoJ, after her approval, to register her credentials to their internal payroll system.
8: Using a decentralized application of the GLASS ecosystem, Alice can use her validated digital identity to request remotely the required documentation from the respective Greek Authority (MoDG), the Portuguese authority (MoJ) and her employer.
9: MoDG can digitally issue and validate the documentation and transmit the encrypted data into the distributed network while the transaction among the users is being recorded.
10: All the transactions, including requests, notifications, and permissions, can be monitored and stored, protecting Alice's (and each participant's) privacy.

---

can be decoded and reconstruct the full file. [34] utilized erasure codes in a scenario combining blockchain and IPFS.

Decentralized applications complexities Multiple novel decentralized applications have already been developed on top of IPFS, with luminous examples, a music streaming platform, and an open-access research publication repository [35,36]. Distributing seemingly centralized applications offer multiple advantages, such as rewarding the creators of music or research publications directly without involving any trusted intermediaries and is feasible with the assistance of blockchain technologies [37].

Within the GLASS ecosystem, it is critical to clearly define where these decentralized applications would be developed and executed to avoid obstacles due to the complexities of the underlying technologies. A potential solution is to carry out the execution of the decentralized applications off-chain [37], similarly

to other popular decentralized applications ecosystems, such as Blockstack [38].

4   Methodology and Implementation

IPFS uses Libp2p[3] as it's base. Originally Libp2p was part of the IPFS project but has since become standalone. It provides all of the transport abstractions and the Kad-DHT functionality. The main release is written in Go, with ports to Rust and JavaScript. To look at the implementation of the DHT, JavaScript was chosen as it natively would not rely on a multi-threading approach but instead asynchronous I/O and an event-driven programming model.

For small scale local testing of the DHT, a simple Libp2p node was created 40 times[4] to listen on the host, and the port will differentiate each node. The DHT configuration [5] is the standard recommended Libp2p Kad-DHT configuration with all standard defaults applied. The exception being the DHT random walk – which is not enabled by default but does allow for random host discovery. The connection encryption used is Noise protocol. [6]

When a new node is initialized, it knows no peers. Typically in IPFS, this issue is solved by bootstrapping the node – providing it with a set of long-serving core nodes that have fully populated routing tables ready to share. In this case, to provide some basic routing entries, the initial node is populated by the address of the next created node, ensuring that each node knows of at least one other but only the next node. Although enabled, the random walk would be an untenable solution to peer discovery in such a small set of nodes given that the Libp2p implementation of the random walk involves dialling a random peerId created from a sha256 multi hash of 16 random bytes.

The last node initialized is then chosen to host the content. To transform the content into a CID, it is first hashed with the standard sha256 algorithm, and then a multi hash is created from this. As we are using CIDv0, the multi hash is then base 58 encoded (CIDv1 is base 32 encoded)and provided to the js-cids library to create the CID.

Once the CID is created, the final node starts providing it to the network. The content routing class of the Kad-DHT will then distribute the pointer to the nodes closest to the key itself. Each node DHT will then begin searching for other nodes and populating its routing table entries. The peer discovery process is best witnessed by examining the debug log for the Kad-DHT by starting the program with the following: `DEBUG="libp2p:dht:*" node index.js`

Each instance of the Kad-DHT is initialized with an instance of the Providers class that manages all known providers – a peer known to have the content for a given CID. The providers class is initialized with an instance of the datastore, which houses the records of providers in the format of a key-value pair, with the key being created from the array of the CID and PeerID and the value being the time the record was entered into the store.

---

[3] Lib2p: https://github.com/libp2p/js-libp2p
[4] Code can be found at: https://github.com/aaoi990/ipfs-kad-dht-evaluation
[5] DHT configuration: https://github.com/libp2p/js-libp2p-kad-dht
[6] Noise Protocol: https://noiseprotocol.org/

When the class is created, it spawns its own cleanup service. The service is a set interval clean up that runs and keeps the list of providers healthy. It is important to note at this point that although a list of providers are stored in the datastore, to ensure access is fast, there is an LRU (least recently used) cache in front of it which speeds up the process of not only cleaning up expired providers but accessing active ones as well. The default constant for the LRU size is 256, and the default cleanup interval is one hour. The cleanup service retrieves all provider entries from the datastore, checks the time of entry against the current time, and batch deletes any which have been in the store for longer than the one-hour window.

The getClosestPeers query is a direct query of the peers taken from the DHT's RoutingTable class, which is responsible for managing the kBuckets. The query looks through all nodes in the kBuckets and returns the closest 20 (as the default bucket size in IPFS is 20). Libp2p uses the javascript implementation k-bucket [7] to handle the management of the buckets. The function does a raw calculation of the XOR distances by comparing each PeerId in the bucket as a unit8aray to the CID as a uint8array and then orders them from nearest to furthest.

With a populated routing table, it is now possible to query the network to find any provider of the created CID. In this instance, the very first initialised node – who only had contact details for the second initialised node – can query the DHT using the built-in *findProviders* function. The result of the promise is an array containing the details of any node providing the requested content. More details on the system's configuration can be found in Appendix A.

5   Evaluation

Fig. 2: All processes - With Kad-DHT processes shown in green

To evaluate the runtime performance of the JavaScript implementation of the Kad-DHT, we can examine the flame graph of the running processes. Figure 2 shows the performance of the entire program from start to finish. Each rectangle represents a stack frame, with the y-axis showing the number of frames on the stack – the stack depth. The bottom of each icicle shows the function on-CPU, with everything above it being the function ancestry. The x-axis spans the entirety of the sample population grouped alphabetically. The total width of

---

[7] K-bucket: https://github.com/tristanls/k-bucket

each rectangle is the total time it was on-CPU or part of the ancestry that was on-CPU; the wider the rectangle, the more CPU consumed per execution. It is worth noting that time is not represented in flame graphs. The Graphs and the logs used to generate them can be found in the corresponding git repo[8].

Table 1: CPU time by package based on Fig. 2.

Package	Function	Percentage
libp2p-noise	performXXhandshake	28.9
libp2p-noise	exchange	18.87
libp2p-noise	finish	10.07
peer-id	createFromPubKey	4.86
libp2p	encryptOutbound	2.43
libp2p	encryptInbound	2.005

Figure 2 illustrates that unsurprisingly the vast majority of CPU usage was spent in the crypto functions, either performing handshakes between nodes or in the functions that support the key generation process.
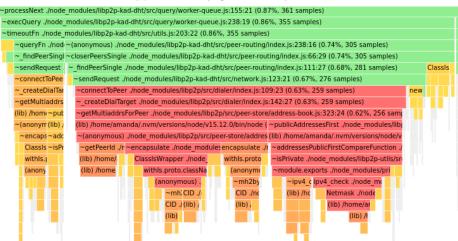

Fig. 3: Some of the Kad-DHT specific processes

The key generation for a basic Libp2p2 node is a base64 encoded string of a protobuf containing a DER-encoded buffer. A node buffer is then used to pass the base64 protobuf to the multi hash function for the final PeerId generation. By default, the public key is 2048 bit RSA. As suggested in the security improvements in [7], peerId generation should be an expensive process in order to mitigate the ease of performing Sybil attacks, and although it was expensive compared to the overall effort of the program, this was primarily because of the default usage of RSA. If EC had been used as per CVE-2020-10937

---

[8] Code   can   be   found   at: https://github.com/aaoi990/ipfs-kad-dht-evaluation/tree/main/perf

[8], the CPU overhead would have been significantly lower. Figure 3 illustrates one of the full stack depths with Kad-DHT ancestry.

Table 2: CPU time by DHT component based on Fig. 3.

Package	Function	Percentage
network	writeReadMessage	1.08
worker-queue	processNext	0.87
peer-routing	closerPeersSingle	0.4
routing	add	0.1
index	nearestPeersToQuery	0.1

Fig. 4: Kad-DHT processes over an one-hour window

Overall the Kad-DHT functions occupied a very low percentage of the CPU time, consistently presenting at less than 3.00%, with the highest usage coming from network functions. The test code being run is a simple start – provide – find – stop sequence, meaning the bulk of the work is being done to configure, connect and route the nodes. It is expected that the longer the program runs, the greater percentage of time the Kad-DHT functions would occupy due to the routing table maintenance functions. During normal operations, the Kad-DHT will force a refresh every 10 minutes by default. During this, each bucket is gone through - from bucket 0 up until the highest bucket that contains a peer (currently capped at 15). A random address from the address space that could fit in the chosen bucket is then selected, and a lookup is done to find the k closest peers to that random address. This constantly ensures that each bucket is filled with as many peers that will fit. Figure 4 results from timing the original code to run for an one-hour window, enabling multiple routing table refreshes. In the timed run, Kad-DHT functions accounted for 11.58% of CPU usage up from the initial program run of 2.55%, which is a 354% increase in the amount of time spent in functions with Kad-DHT ancestry.

6   Conclusions

eGovernance presents unique challenges in terms of privacy-preserving and providing secure solutions in eGovernance services. Precisely when the utilized data

is derived from industrial control systems and sensors. In this paper, we present GLASS, our decentralized solution, that moves towards that direction by examining the effectiveness and efficiency of distributed cutting-edge technologies and demonstrates the capacity of a public, distributed infrastructure based on the InterPlanetary File System (IPFS).

One practical implementation of the GLASS concept is being done within the aims of the GLASS project, highlighting how the GLASS concept can potentially be integrated into a broad field of use cases. Our proposed GLASS-oriented approach is a decentralized solution that combines the InterPlanetary File System (IPFS) with Distributed Ledger Technology and Smart Contracts to secure eGovernance services. We show in this paper how our approach can be used to fulfil the needs of the GLASS concept. Finally, and on top of the above, we created a testbed environment to measure the IPFS performance.

Acknowledgments

References

1. Ghafur, S., Kristensen, S., Honeyford, K., Martin, G., Darzi, A., Aylin, P.: A retrospective impact analysis of the wannacry cyberattack on the nhs. NPJ digital medicine **2**(1) (2019) 1–7
2. Analytica, O.: Efforts to curb ransomware crimes face limits. Emerald Expert Briefings (oxan-db) (2021)
3. Maymounkov, P., Eres, D.: Kademlia: A peer-to-peer information system based on the xor metric. Volume 2429. (04 2002)
4. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Guerraoui, R., ed.: Middleware 2001, Berlin, Heidelberg, Springer Berlin Heidelberg (2001) 329–350
5. Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Computer **74** (05 2001)
6. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing nearby copies of replicated objects in a distributed environment
7. Baumgart, I., Mies, S.: S/kademlia: A practicable approach towards secure key-based routing. Volume 2. (01 2008) 1–8
8. Prünster, B., Marsalek, A., Zefferer, T.: Total eclipse of the heart – disrupting the interplanetary file system (2020)
9. Kothari, R., Jakheliya, B., Sawant, V.: A distributed peer-to-peer storage network. International Conference on Smart Systems and Inventive Technology (ICSSIT) (11 2019) 576–582
10. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In Druschel, P., Kaashoek, F., Rowstron, A., eds.: Peer-to-Peer Systems, Berlin, Heidelberg, Springer Berlin Heidelberg (2002) 53–65
11. Mukne, H., Pai, P., Raut, S., Ambawade, D.: Land record management using hyperledger fabric and ipfs. In: 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT). (2019) 1–8

12. Andreev, O., Daskalov, H.: A framework for managing student data through blockchain, sofia, bulgaria: Academic press. In Proceedings of Xth Anniversary International Scientific Conference (2018) 59–66
13. Singh, S.: A blockchain-based decentralized application for user-driven contribution to Open Government Data. PhD thesis (06 2018)
14. Dunphy, P., Petitcolas, F.: A first look at identity management schemes on the blockchain. IEEE Security & Privacy **16** (2018) 20–29
15. Trihinas, D., Pallis, G., Dikaiakos, M.D.: Admin: Adaptive monitoring dissemination for the internet of things. In: IEEE INFOCOM 2017-IEEE conference on computer communications, IEEE (2017) 1–9
16. Domalis, G., Karacapilidis, N., Tsakalidis, D., Giannaros, A.: A trustable and interoperable decentralized solution for citizen-centric and cross-border egovernance: A conceptual approach. arXiv preprint arXiv:2103.15458 (2021)
17. Voigt, P., Von dem Bussche, A.: The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing **10** (2017) 3152676
18. Barati, M., Rana, O.: Design and verification of privacy patterns for business process models. In: Blockchain Technology and Innovations in Business Processes. Springer (2021) 125–139
19. Huang, H., Zhou, S., Lin, J., Zhang, K., Guo, S.: Bridge the trustworthiness gap amongst multiple domains: a practical blockchain-based approach. In: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE (2020) 1–6
20. Papadopoulos, P., Pitropakis, N., Buchanan, W.J., Lo, O., Katsikas, S.: Privacy-preserving passive dns. Computers **9**(3) (2020) 64
21. Stamatellis, C., Papadopoulos, P., Pitropakis, N., Katsikas, S., Buchanan, W.J.: A privacy-preserving healthcare framework using hyperledger fabric. Sensors **20**(22) (2020) 6587
22. Wang, S., Zhang, Y., Zhang, Y.: A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. Ieee Access **6** (2018) 38437–38450
23. Plank, J.S.: A tutorial on reed–solomon coding for fault-tolerance in raid-like systems. Software: Practice and Experience **27**(9) (1997) 995–1012
24. Huang, H., Lin, J., Zheng, B., Zheng, Z., Bian, J.: When blockchain meets distributed file systems: An overview, challenges, and open issues. IEEE Access **8** (2020) 50574–50586
25. Wennergren, O., Vidhall, M., Sörensen, J.: Transparency analysis of distributed file systems: With a focus on interplanetary file system (2018)
26. Shen, J., Li, Y., Zhou, Y., Wang, X.: Understanding i/o performance of ipfs storage: a client's perspective. In: 2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS), IEEE (2019) 1–10
27. Nyaletey, E., Parizi, R.M., Zhang, Q., Choo, K.K.R.: Blockipfs-blockchain-enabled interplanetary file system for forensic and trusted data traceability. In: 2019 IEEE International Conference on Blockchain (Blockchain), IEEE (2019) 18–25
28. Norvill, R., Pontiveros, B.B.F., State, R., Cullen, A.: Ipfs for reduction of chain size in ethereum. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE (2018) 1121–1128
29. Poon, J., Buterin, V.: Plasma: Scalable autonomous smart contracts. White paper (2017) 1–47

30. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
31. Rizzo, L.: Effective erasure codes for reliable computer communication protocols. ACM SIGCOMM computer communication review **27**(2) (1997) 24–36
32. Wilkinson, S., Boshevski, T., Brandoff, J., Buterin, V.: Storj a peer-to-peer cloud storage network (2014)
33. Vorick, D., Champine, L.: Sia: Simple decentralized storage. Retrieved May **8** (2014) 2018
34. Chen, Y., Li, H., Li, K., Zhang, J.: An improved p2p file system scheme based on ipfs and blockchain. In: 2017 IEEE International Conference on Big Data (Big Data), IEEE (2017) 2652–2657
35. Jia, B., Xu, C., Gotla, R., Peeters, S., Abouelnasr, R., Mach, M.: Opus-decentralized music distribution using interplanetary file systems (ipfs) on the ethereum blockchain v0. 8.3. Opus Foundation **2017** (2016)
36. Tenorio-Fornés, A., Jacynycz, V., Llop-Vila, D., Sánchez-Ruiz, A., Hassan, S.: Towards a decentralized process for scientific publication and peer review using blockchain and ipfs. In: Proceedings of the 52nd Hawaii International Conference on System Sciences. (2019)
37. Truong, N., Lee, G.M., Sun, K., Guitton, F., Guo, Y.: A blockchain-based trust system for decentralised applications: When trustless needs trust. Future Generation Computer Systems (2021)
38. Ali, M.: Stacks 2.0 apps and smart contracts for bitcoin (2020)

A   Appendices

A.1   Libp2p node initialisation

```
const node = await Libp2p.create({
addresses: {
 listen: ['/ip4/0.0.0.0/tcp/0']
},
modules: {
 transport: [TCP],
 streamMuxer: [Mplex],
 connEncryption: [NOISE],
 dht: KadDHT,
},
config: {
 dht: {
   kBucketSize: 20,
   enabled: true,
   randomWalk: {
     enabled: true,
     interval: 300e3,
     timeout: 10e3
   }
  }
 }
})
```

Listing 1.1: Libp2p node initialisation.

A.2   Random walk PeerId creation

```
const digest = await multihashing(
    crypto.randomBytes(16), 'sha2−256')
const id = new PeerId(digest)
```

Listing 1.2: Random walk PeerId creation.

A.3   Transforming content to a CID

```
const hash = crypto.createHash('sha256')
    .update('hello world!').digest()
const encoded = multihash.encode(hash, 'sha2−256')
const cid = new CID(multihash.toB58String(encoded))
```

Listing 1.3: Transforming content to a CID.

A.4   A node providing content

```
await node.contentRouting.provide(cid)
```

Listing 1.4: A node providing content.

A.5   Distributing content to the closest peers

```
async provide (key) {
dht._log('provide: ${key}')

  /** @type {Error[]} */
  const errors = []

  // Add peer as provider
  console.log('starting to provide')
  await dht.providers.addProvider(key, dht.peerId)

  const multiaddrs = dht.libp2p ? dht.libp2p.multiaddrs : []
  const msg = new Message(Message.TYPES.ADD_PROVIDER, key.bytes, 0)
  msg.providerPeers = [{
    id: dht.peerId,
    multiaddrs
  }]

  async function mapPeer (peer) {
    dht._log('putProvider ${key} to ${peer.toB58String()}')
    try {
      await dht.network.sendMessage(peer, msg)
    } catch (err) {
      errors.push(err)
```

```
  }
 }

 // Notify closest peers
 await utils.mapParallel(dht.getClosestPeers(key.bytes), mapPeer)

 if (errors.length) {
  throw errcode(new Error('Failed to provide to ${errors.length} of ${dht.↵
 kBucketSize} peers'), 'ERR_SOME_PROVIDES_FAILED', { errors })
 }
},
```

Listing 1.5: Distributing content to the closest peers.

A.6 Creation of the datastore

```
const dsKey = [
   makeProviderKey(cid),'/',
   utils.encodeBase32(peer.id)].join('')
const key = new Key(dsKey)
const buffer = Uint8Array.from(
   varint.encode(time.getTime()))
store.put(key, buffer)
```

Listing 1.6: Creation of the datastore.

A.7 Calculating the closest Peers using the XOR metric

```
closest (id, n = Infinity) {
ensureInt8('id', id)

if ((!Number.isInteger(n) && n !== Infinity) || n <= 0) {
 throw new TypeError('n is not positive number')
}
let contacts = []

for (let nodes = [this.root],
   bitIndex = 0; nodes.length > 0 && contacts.length < n;) {

 const node = nodes.pop()
 if (node.contacts === null) {
  const detNode = this._determineNode(
     node, id, bitIndex++)
  nodes.push(
     node.left === detNode ? node.right : node.left)
  nodes.push(detNode)
 } else {
  contacts = contacts.concat(node.contacts)
```

```
    }
  }

  return contacts
    .map(a => [this.distance(a.id, id), a])
    .sort((a, b) => a[0] − b[0])
    .slice(0, n)
    .map(a => a[1])
}
```

Listing 1.7: Calculating the closest Peers using the XOR metric.

A.8   Finding providers

```
await all(nodes[0].contentRouting
    .findProviders(cid))
```

Listing 1.8: Finding providers.

A.9   Result of the "Finding Providers" query

```
{
 id: PeerId {
   _id: <Buffer 12 20 83 42 f7 0e 33 90 d1 c4 41 d0 80 d7 16 63 be 43 95 20 3c ↪
   b1 79 5e 23 d7 28 12 3e 4a 0f aa d9 d3>,
   _idB58String: 'QmXB3LoMkXQh3HzQo1fy−
    9UEJZZQw2MmJKWRhG4nfbTR7Qe',
   _privKey: undefined,
   _pubKey: undefined
 },
 multiaddrs: []
}
```

Listing 1.9: Result of the findProviders query.