Enhancing IPFS privacy through triple hashing

Thomas Katsantas, Yannis Thomas, Christos Karapapas and George Xylomenos
Mobile Multimedia Laboratory, Department of Informatics
Athens University of Economics and Business, Greece
e-mail: tomkatsantas@gmail.com, {thomasi, karapapas, xgeorge}@aueb.gr

***Abstract*—The InterPlanetary File System (IPFS) is a popular distributed storage system, with worldwide presence. Due to the open nature and distributed routing of IPFS, malicious users can monitor the content other users are retrieving, violating their privacy. Unlike previous privacy enhancement schemes that rely on user anonymity, hiding the identity of the requesting user behind a proxy, we hide instead the identity of the content, using only hash functions. This prevents intermediaries from detecting what users are retrieving, without relying on any trusted third parties. Our solution also supports optional content encryption, which is especially useful for caching. We present the design and implementation of our solution for IPFS and evaluate its privacy and security properties, showing that our scheme enhances privacy and prevents a range of DDoS attacks.**

I. Introduction

The evolution of Web 3.0 has renewed interest on distributed file systems, such as the *InterPlanetary File System* (IPFS) [1], a fully decentralized *Peer to Peer* (P2P) system. In P2P systems like IPFS, many peers need to collaborate to locate a piece of content, creating an obvious privacy issue: a malicious peer in the lookup path can monitor the content that other peers are looking up. The malicious peer can even create multiple virtual nodes to amplify its ability to peek into lookup requests. So far, IPFS has no defenses against such privacy attacks.

Most current P2P systems address the need for privacy via user anonymity: they hide the identity of the peer asking for content by inserting a trusted proxy in front of it. This technique relies on a trusted third party, which is a single point of failure, and it increases lookup latency, as lookups need to take a detour through the proxy.

We present a simpler solution: instead of hiding the user's identity, we hide the content's identity by using triple hashing when storing and looking up content, switching to single and double hashing for the actual content exchange. As a result, intermediate peers cannot detect what a peer is looking for. Our solution also supports optional content encryption, which is useful for caching. Our privacy and security analysis shows that, in addition to enhancing privacy, our scheme deals with a range of DDoS attacks against IPFS. Furthermore, it does not add routing delays, it does not rely on trusted third parties and it has a negligible processing overhead.

The remainder of this paper is structured as follows. In Section II we outline IPFS operations and explain the motivation for our work. We describe our triple hashing solution in Section III. We discuss its implementation and evaluate its privacy and security properties in Section IV. In Section V we review related work and we conclude in Section VI.

II. Background and Motivation

A. The InterPlanetary File System

The IPFS is a modular suite of protocols, designed from the ground up for content addressing and peer-to-peer networking [1]. Each stored object in IPFS has a unique *Content Identifier* (CID). CIDs support different encodings and hash functions by indicating (a) the base-encoding for the CID ("b" for base32), (b) the CID version (v0 or v1), (c) the multicodec, showing how the addressed data are encoded (protobuf, json, cbor, etc.) and (d) the multihash, showing the hash function used, the hash length and the hash value of the content.

Each peer in IPFS has a unique *Peer Identifier* (peerID), in the same space as the CID; it is produced by hashing a public key of the peer (not its IP address). IPFS organizes the peers into a *Distributed Hash Table* (DHT), using a customized version of the Kademlia DHT [2]. The basic function provided by Kademlia is locating the k nodes whose peerIDs are the closest to a given CID or PeerID based on the XOR metric.

When an object is made available in IPFS, the peer storing it creates its CID and then looks up via Kademlia the k closest peers (by default, $k = 20$) to that CID. These peers are then asked to store a (CID, peerID) pair, called the Provider Record; the peerID indicates the provider of the CID, that is, the node that stores the corresponding object. To contact actual nodes, each peer also maintains a number of *Peer Records*, which are triplets of the form (peerID, IP address, UDP port). Based on these records, an IPFS peer can store and retrieve content.

Lookup in IPFS proceeds iteratively, that is, the requesting peer asks its neighbors who are closest to a CID to return either the Provider Record (if they have it), or their own closest neighbors to the CID (if any). The process is repeated, until no peers closer to the CID can be found. Kademlia ensures that this process takes $O(\log N)$ steps for a network of N peers.

B. The Bitswap protocol

Bitswap is a protocol used to find, search and exchange content blocks between IPFS peers [3]. Bitswap uses three types of request messages (WANT_HAVE, WANT_BLOCK and CANCEL) and three types of response messages (HAVE, BLOCK, and DONT_HAVE). When a user wants to retrieve one or more content blocks, it sends a WANT_HAVE message with a WANTLIST to one or more peers. The WANTLIST contains the CID(s) of the content block(s) being requested. Each peer who receives the WANT_HAVE, adds the WANTLIST to a ledger of pending requests. If the peer has the block, it responds with a HAVE message. When the requesting client

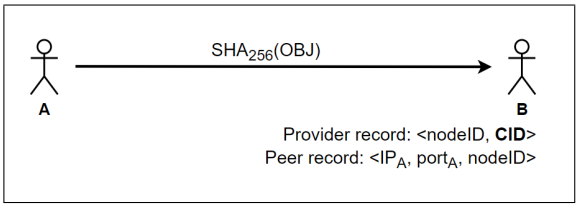

Fig. 1. Plain IPFS store: A chooses B based on the CID of OBJ.

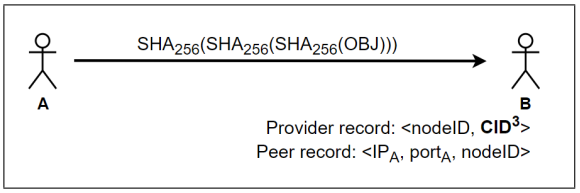

Fig. 2. Triple Hash store: A chooses B based on the CID^3 of OBJ.

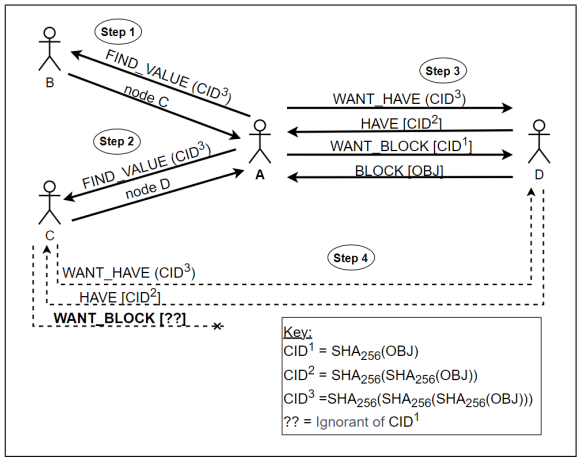

Fig. 3. Triple Hash lookup.

receives a HAVE message, it responds with a WANT_BLOCK request to retrieve the block. The block is returned in a BLOCK response, and its CID is removed from the ledger. If a root CID, which is a block pointing at a set of second level CIDs, is used instead of a single CID, the procedure of WANT_HAVE requests, and transfer of blocks using WANT_BLOCK is repeated, until all blocks are retrieved.

In IPFS, a peer first tries to retrieve an object by using Bitswap to broadcast WANT_HAVE messages to all it neighbors. If there are no responses, after 1 sec the peer looks up the CID using Kademlia to retrieve the Provider Record for the CID and the Peer Record for its provider. Then, the peer uses Bitswap with the provider of the object to retrieve it.

C. Threat Model

Since in IPFS a peer always starts by broadcasting its requests via Bitswap, all its neighbors can monitor which CIDs it is requesting. If the peer resorts to a DHT lookup, all peers on the path of the request also learn the CID of the requested object. To amplify this, a malicious node can create a large number of nodes who behave lawfully, but also spy on their peers. A recent study shows that an alarming number of IPs that have been reported to be malicious, is using IPFS [4].

If a specific peer is targeted, the malicious node just needs to become its neighbor, to monitor all its requests via Bitswap. The knowledge of the CID allows malicious nodes to also retrieve the actual content of the object. Balduf et al. [5] performed this attack and observed the requested CIDs, the client and the request time, and even retrieved the objects corresponding to the CIDs. If the object was unencrypted, the attacker could create a full profile of the victim [6].

In this paper, our goal is to mitigate such problems, by not revealing which CIDs a peer is looking up, both to its neighbors and to peers found through the DHT. The only peer that can learn what the requesting peer is asking for, is the peer that actually stores the corresponding content, that is, the actual CID is only revealed to the peer storing that content.

III. The Triple Hash Solution

A. Ensuring Privacy

To make an object OBJ available in IPFS, a server A first creates the object's CID by hashing its content and adding some metadata. Then, it adds Provider Records and Peer Records pointing at itself in the k nodes whose PeerIDs are the closest to that CID, using the Kademlia DHT; Fig. 1 shows the information inserted in a chosen node B. In our solution, in addition to CID, server A also creates CID^3 by performing a *triple hash* of the object's content. Then, it inserts a Provider Record for CID^3 and a Peer Record for itself at the k closest nodes to CID^3, as shown in Fig. 2. The only change then, is that we use CID^3 instead of CID for the insertion. In the following, we will use CID^1 as a synonym for CID, and CID^2 for the double hash of the object.

To retrieve an object from IPFS, a client A starts with the CID of the object; the CID may have come from a search engine, a web site, or a previously retrieved object. As shown in Fig. 3, in our solution A calculates CID^3 and sends a FINDVALUE RPC to the $a = 3$ closest nodes to CID^3 in its routing table. When node B receives the RPC (Step 1), it first checks to see if it already has the object locally stored; if it does, it will be marked with both CID^1 and CID^3. In our example it does not, so it returns the details of node C who is closer to CID^3. Then, node A sends the same RPC to node C (Step 2). Node C happens to be one of the k closest nodes to CID^3, so it can respond to A with a Provider Record for it, which shows that node D has the desired object.

Node A, knowing that node D has the object corresponding to CID^3, switches to Bitswap for the actual retrieval, sending to D a WANT_HAVE message with CID^3 (Step 3). Node D, responds with HAVE, but using CID^2 rather than CID^1. When node A receives CID^2 from D, it verifies that D knows the requested object, as CID^2 can be easily calculated from CID^1, but not from CID^3. A finally sends a WANT_BLOCK with the true hash, CID^1. Node D thus verifies that A already knew CID^1, so it returns the object with a BLOCK message.

Consider now what would happen if node C, who found out that node A was looking for CID^3 during routing, was malicious. C cannot calculate CID^1 from CID^3, but it could

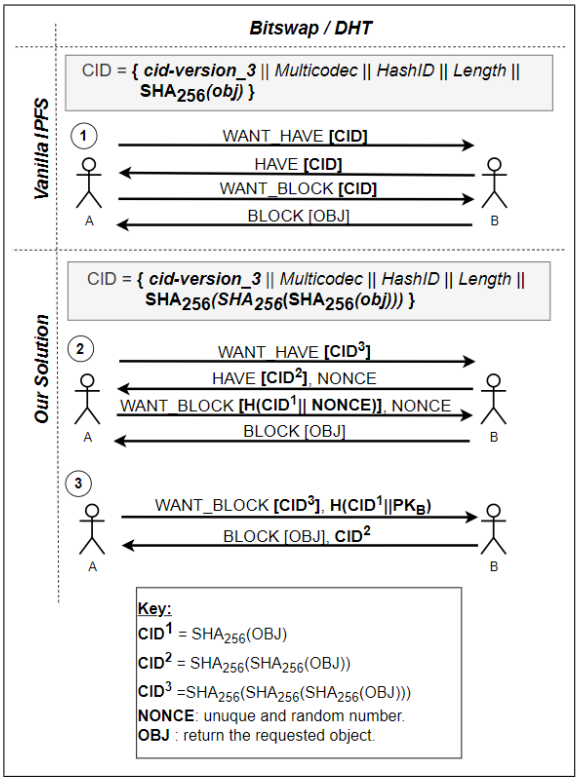

Fig. 4. Retrieval with plain IPFS (top) and with triple hash (bottom).

try to ask node D for the object (Step 4). However, D replies with CID^2, from which C still cannot calculate CID^1, so node C cannot find either the true hash or the content of the object.

To allow our solution to co-exist with plain IPFS, we use a new version number for the CID, to indicate that the requesting node wants to use the triple hash method (see Section IV). Therefore, the main differences of our solution with plain IPFS are: (a) the version and object hash fields of the multihash are modified to indicate and include the triple hash, (b) the store and retrieve processes use the triple hash and, (c) in the Bitswap stage, both the client and the server confirm that the other end knows the true, single hash, before the exchange.

Fig. 4 shows the details of the Bitswap exchange with plain IPFS and triple hash. Case 1 shows plain IPFS, assuming that the requesting peer A either has found through the DHT that peer B holds the desired object, or just happens to have B in its neighbor list to begin with. The process relies on four messages using the CID (or CID^1) of the object [1].

Case 2 show Bitswap with triple hash. We observe that the number of exchanged messages remains the same, but the content of the messages is differentiated. Specifically, in the first message the client requests CID^3, in the second message the server responds with CID^2, in the third step we request the block with CID^1 and in the final step the object is returned. To prevent eavesdropping and replay attacks, we also add in the HAVE message a *nonce* value, which is randomly generated by node B and associated with the IP address of node A. When node A sends the WANT_BLOCK message, it includes the nonce and uses as the content identifier the hash of CID^1 and the nonce. Node B, by verifying the hash, confirms that node A knows CID^1, even though CID^1 is never sent unprotected, and that the message is fresh, due to the nonce.

Case 3 shows an alternative two message scheme, which is applicable when A already knows that B has the desired object; this is the case when B if found by a DHT lookup, but not when Bitswap brodacasts the request to all neighbors. In this case, A does not have a nonce to hide CID^1, but still needs to prove to B that it knows it, so the WANT_BLOCK message includes CID^3, to indicate the desired content, and a hash of CID^1 and PK_B, the public key of B. Node B, by verifying the hash, confirms that node A knows CID^1. The object is returned with CID^2, which allows A to recognize it and confirm that B knows CID^1, without revealing CID^1.

As we discuss in Section IV, this *two message triple hash* solution cannot prevent some of the DDoS attacks prevented by the *four message triple hash* solution. A simple heuristic would be to normally use the two message solution, but if a node receives more messages from the same IP address than a threshold, switch to the four message solution.

B. Encrypted Caching

Since looking up content via the DHT is a multistep process, IPFS tries to locate as much content as possible via Bitswap. Recall that when a peer sends its WANTLIST to its neighbors, the neighbors that do not have the corresponding objects, add their CIDs to a ledger, along with the requesting peer. Each time a peer receives a new object, it checks if any of its peers have asked for it, and sends it to them if they do. This improves the chances of finding content via Bitswap. [3]. To further speed up content retrieval, more powerful nodes can proactively cache popular content. For instance, if a node receives many requests for a CID, it can request the corresponding object and cache it. Similarly, if it receives a STORE RPC for the Provider Record of a CID, it can request the corresponding object and cache it.

Our triple hash solution, though, prevents this, since a node needs to know CID^1 to retrieve an object, which we have hidden behind CID^3. To allow caching without compromising privacy, we can allow caches to receive an *encrypted* version of the object matching CID^3, without revealing its CID^1, while still allowing the nodes who know CID^1 to decrypt the content. In Fig. 5 we show how our schemes can be adapted to allow caching encrypted objects; on the left we show cases where a node B holds the unencrypted object, while on the right we show cases where a node C holds the encrypted object.

Case 1 shows a normal retrieval procedure, where server$B$ has the unencrypted object and CID^1, and the requesting client A knows CID^1 and wants the corresponding object. Server B verifies that node A knows CID^1 from the WANT_BLOCK message, so it returns the unencrypted object in the BLOCK message. In case 2, the difference is that node C only knows CID^3. After learning CID^2 and the nonce value from node B, it can only request the encrypted version of the object, by including CID^2 hashed with the nonce in the WANT_BLOCK message. Node B notices that CID^2 was used, so it returns in the BLOCK message the object encrypted with CID^1 as the key. Node C cannot decrypt the object, as it does not know CID^1, but it can cache it for other nodes.

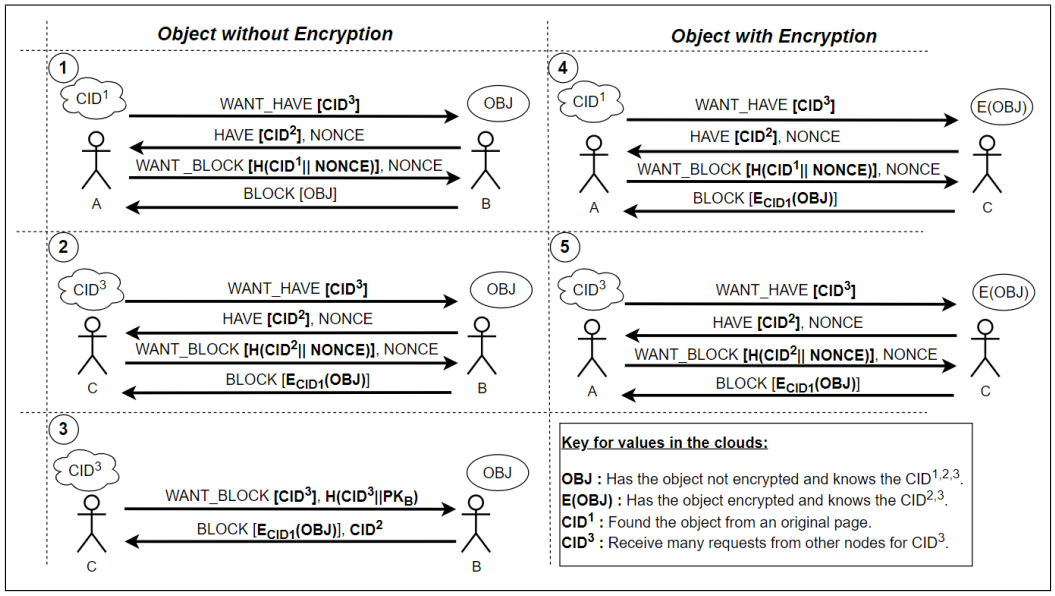

Fig. 5. Combining triple hash with encrypted objects for caching purposes.

In case 3 we have the same setup, but node C uses the two message triple hash solution. In this case, node C hashes CID^3 with the public key of node B, rather than CID^1. Node B again notices this, and responds with the object encrypted with CID^1 as the key, plus CID^2. CID^2 proves that B has the original object (the hash of CID^2 should produce CID^3). In both case 2 and case 3, client C learns CID^2 and retrieves the encrypted object, but does not learn CID^1. CID^2 is needed to return the encrypted object to other nodes.

Now we can see what happens when node C has cached the encrypted object, as a result of case 2 or 3. In case 4, node A knows CID^1, so the only difference from case 1, is that node C only has the encrypted content and does not know CID^1. As a result, it cannot verify whether node A actually knows CID^1 from the WANT_BLOCK message, but this is unimportant: if node A does not know CID^1, it cannot decrypt the object anyway. So, node C returns the encrypted object in the BLOCK message, which node A can decrypt with CID^1. In case 5, node A does not know CID^1, so it can only cache the encrypted object. In its WANT_BLOCK it uses CID^2 and the nonce for the hash; again, C returns the encrypted object, which A can now cache. This is the same as case 2, the only difference is that the server only had the encrypted object.

IV. Implementation and Evaluation

A. Implementation

To store an object in plain IPFS, we need to create a CID, whose last field is the hash of the object's content:

$$CID : \{\mathbf{cid-version}||Multicodec||HashID|| Length||\mathbf{SHA_{256}(obj)}\} \quad (1)$$

To maintain compatibility with plain IPFS, when using the triple hash method a new version number is used for the CID:

$$CID^3 : \{\mathbf{cid-version\_3}||Multicodec||HashID|| Length||\mathbf{SHA_{256}(SHA_{256}(SHA_{256}(obj)))}\} \quad (2)$$

By looking at the version number, a node can determine how to interpret the CID during Bitswap; the DHT does not differentiate them. However, a client who has found a CID from a web site, a search engine or via another file, cannot know whether the storing node used plain or triple hash IPFS. For this reason, when an object is stored using triple hash, its CID should also be advertised using the new version:

$$CID^1 : \{\mathbf{cid-version\_3}||Multicodec||HashID|| Length||\mathbf{SHA_{256}(obj)}\} \quad (3)$$

All these procedures only require calculating hashes of other hashes, so their computational overheads are negligible.

B. Privacy evaluation

In our triple hash solution, storing and looking up Provider Records, as well as requesting the corresponding objects, always relies on the triple hash of a CID, the CID^3. Assuming a cryptographic hashing function, like SHA-256, it is computationally infeasible for a malicious node to determine the CID from CID^3. Therefore, a malicious node that learns a requested CID^3 via Bitswap or the DHT, cannot find the real CID or retrieve the corresponding object. The only node that learns the CID that a requesting client has asked for, is the server storing the requested object, as it must eventually return the object to that client via Bitswap.

Our triple hash solution offers partial protection against eavesdroppers and replay attacks on Bitswap. In the four message solution, the server first proves to the client that it knows the true CID by sending CID^2, and prevents replay attacks by also sending a nonce. The client then proves to the server that it knows the true CID, without revealing it, by hashing it with the nonce. It is impossible by looking at these messages to determine the true CID, and it is impossible to replay the WANT_BLOCK message due to the nonce.

In the two message solution, the CID is again never revealed, as it is hashed with the public key of the server, the

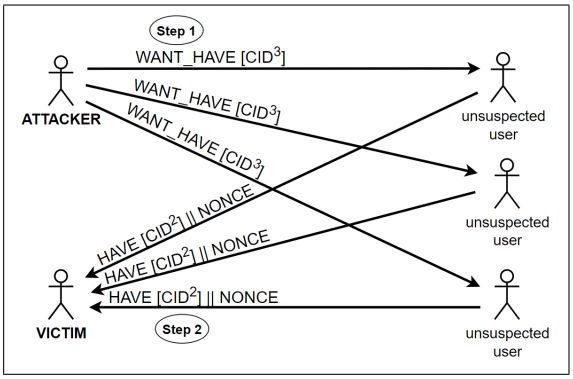

Fig. 6. Preventing DoS attacks with four message triple hash.

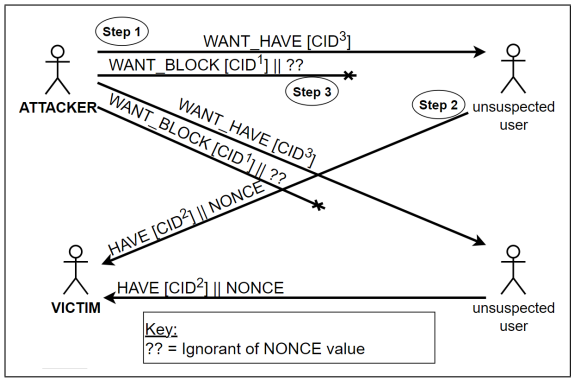

Fig. 7. Preventing SR_DRDoS attacks with four message triple hash.

server verifies that the client knows the true CID from that hash, and the client verifies that the server knows the true CID, as the last message also contains CID^2. However, since there is no nonce, this solution is vulnerable to replay attacks.

To prevent an eavesdropper on Bitswap from learning the contents of an object, we can encrypt it, if desired, using the CID as a key, as described in the solution used for caching. The encrypted solution has the same properties as the unencrypted one, whether the four message or the two message solution is used: the true CID is never sent in the clear, and a client that cannot prove knowledge of the true CID never gets the unencrypted content or the true CID.

C. Protection against DDoS attacks

Due to the open nature of IPFS, many DDoS attackers can be mounted against it; our triple hash solution can prevent some of them. A simple DDoS attack is for a malicious node to locate copies of a large object in legitimate servers, as part of its normal Bitswap and DHT operation, and then send a large number of WANT_BLOCK requests for that object by spoofing the IP address of a victim. The result is an avalanche of large BLOCK messages to the victim, from legitimate servers. This attack is easy to set up, since IPFS relies on UDP, so no connection needs to be setup; the responses can even be directed to a non-IPFS node. Furthermore, as peerIDs are derived from public keys and not IP addresses, a malicious node can more easily spoof IP addresses.

In our triple hash solution, it is much harder to launch such an attack. First, it is hard to determine the size of an object requested via CID^3, as without the true CID a malicious node cannot retrieve the object. Also, as shown in Fig. 6, even if the attacker spoofs a large number of WANT_HAVE messages to unsuspecting servers in the four message solution, they will respond only with a HAVE message with CID^2. Without proving knowledge of CID^1, the server will not return the actual block, so the messages returned are comparable to the size of the requests; there is no amplification. If the two message solution is used, the attacker will not be able to construct the right WANT_BLOCK message, as it does not known CID^1, therefore the server will not return a response. However, if an attacker captures a legitimate WANT_BLOCK message, since the two message variant does not have a nonce to guard against replay attacks, the attacker can mount a DDoS attack by spoofing the IP address of the captured message.

A more complex attack is the SR_DRDoS attack, which leverages the services of third-party servers to amplify and reflect attacks towards a target, overwhelming it with a massive amount of traffic. In this scenario, the attacker first plants is own object, for which it knows both CID^1 and CID^3, in unsuspecting servers [7]. In plain IPFS, the attacker can again spoof the WANT_BLOCK messages using the victim's IP address, to direct a large amount of traffic to the victim. When triple hash is used, this attack can be slightly modified, since now the attacker knows the true CID of the large object. First, the malicious node spoofs the victim's IP address and sends a WANT_HAVE with the CID^3 of its own object. The servers will respond with a HAVE with CID^2 to the victim. The attacker waits for some time, and then sends a WANT_BLOCK message with CID^1, spoofing again the victim's IP address to direct the BLOCK message to it.

Our four message solution is resistant to this attack, as shown in Fig. 7, since for every WANT_HAVE request, the server issues a nonce and ties it with the (spoofed) IP address of the victim. Since the attacker does not know the nonce, it cannot create a correct WANT_BLOCK message, which also requires the nonce, so it cannot make the servers send the large object to the victim. On the other hand, if the attacker does not spoof the IP addresses of the WANT_HAVE messages to get the nonces itself, the WANT_BLOCK messages will not match the IP address tied to these nonces, preventing again the servers from responding. Finally, if a replay attack is mounted with a captured WANT_BLOCK message, it will fail, as the nonce will not be current. Unfortunately, the two message solution is susceptible to this attack, due to the lack of the nonce. Since the attacker knows the true CID, it can trick the servers in returning the large object to the victim.

V. Related Work

Many privacy enhancing schemes have been proposed for P2P systems. In [8], the authors add delay to messages in order to confuse attackers, at the cost of also increasing communication latency. In [9], the authors introduce dummy messages along with the legal messages, which introduces traffic overhead. In [10], the authors use multiple paths to prevent eavesdropping, which complicates deployment. Finally, a very

popular tool in this context is the Tor Browser, which uses an onion routing technique to conceal the IP address IP of the requester, at the cost of significant latency [11]. Some privacy enhancing schemes rely on a *Public Key Infrastructure* (PKI) to establish trust [12]. While the exploitation of a PKI, or trusted third parties in general, can address several privacy attacks on P2P networks, it makes a distributed system rely on centralized elements.

A privacy enhancing solution customized to IPFS is to use Bloom filters rather than the CIDs themselves to obfuscate the request and response messages in Bitswap [13]. Unlike our triple hash solution, this solution requires many changes to Bitswap, it has considerable computational and storage overhead, since each node must maintain Bloom filters of all its content, and it does not ensure privacy during DHT lookups.

The use of hashing for enhancing privacy is also met in the GNUnet P2P network [14]. GNUnet uses a triple hashing technique to encrypt the name of the content item and to verify that the serving node has indeed the requested item. However, it only works with encrypted data, which is not always necessary. Our triple hash solution can work with encrypted or unencrypted data, and it also provides resistance to DDoS attacks. While our solution does not offer full privacy, as the server knows the client, it allows the client to verify that the server has the desired object.

A new privacy approach to IPFS is the *InterPlanetary Network Indexer* (IPNI), which is a separate network with its own protocols. Every provider can store its content to IPFS and IPNI at the same time; IPNI nodes do not hold content, just pointers to IPFS nodes. The IPNI uses a two hash technique: a provider does not store an object's CID, but a hash of the CID; all data stored in IPNI are also encrypted with the CID. Furthermore, the messages exchanged are all encrypted, while in our solution the protocol can work with and without encryption. The way that messages are encrypted in IPNI, prevents the use of two hash privacy in IPFS; as a result, if a CID is not found in IPNI, the request falls back to plain IPFS with no privacy. Our three hash solution can be used directly with IPFS, without requiring a separate network and different protocols.

VI. Conclusion

In IPFS malicious nodes can easily monitor the requests that other nodes make, learn the CIDs of the objects requested and even retrieve the corresponding content. To counter these privacy attacks, we presented a triple hash solution which hides the CIDs requested from nodes who do not store the actual object. Our solution also allows encrypting objects, to allow caching them at any node. Our solution does not hide the node requesting an object via proxies, to avoid introducing delays and trusted third parties. Instead, it hides the identity of the object that is being searched. It can be added to IPFS without affecting its current functionality, and it can be combined with node anonymization mechanisms. Our four message triple hash solution is resistant to many common DDoS attacks, eavesdropping and message replaying, without adding any latency. Our two message triple hash solution is not as resistant to attacks, but it reduces Bitswap latency by half.

References

[1] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of IPFS: a storage layer for the decentralized web," in *Proceedings of the ACM SIGCOMM Conference*, 2022, pp. 739–752.

[2] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proceedings of the International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.

[3] A. De la Rocha, D. Dias, and Y. Psaras, "Accelerating content routing with bitswap: A multi-path file transfer protocol in IPFS and filecoin," Protocol Labs, Tech. Rep., 2021.

[4] C. Karapapas, G. C. Polyzos, and C. Patsakis, "What's inside a node? malicious IPFS nodes under the magnifying glass," *arXiv preprint: 2307.12212*, 2023.

[5] L. Balduf, S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Monitoring data requests in decentralized data storage systems: A case study of IPFS," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 658–668.

[6] S. Sridhar, O. Ascigil, N. Keizer, F. Genon, S. Pierre, Y. Psaras, E. Riviere, and M. Król, "Content censorship in the interplanetary file system," *arXiv preprint: 2307.12212*, 2023.

[7] B. Liu, S. Berg, J. Li, T. Wei, C. Zhang, and X. Han, "The store-and-flood distributed reflective denial of service attack," in *Proceedings of the International Conference on Computer Communication and Networks (ICCCN)*, 2014, pp. 1–8.

[8] K. Bauer, D. McCoy, D. Grunwald, and D. Sicker, "Bitblender: Light-weight anonymity for bittorrent," in *Proceedings of the workshop on Applications of private and anonymous communications*, 2008, pp. 1–8.

[9] J.-F. Raymond, "Traffic analysis: Protocols, attacks, design issues, and open problems," in *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, 2001, pp. 10–29.

[10] D. Mhapasekar, "Accomplishing anonymity in peer to peer network," in *Proceedings of the International Conference on Communication, Computing & Security*, 2011, pp. 555–558.

[11] R. Dingledine, N. Mathewson, P. F. Syverson *et al.*, "Tor: The second-generation onion router." in *Proceedings of the USENIX Security Symposium*, vol. 4, 2004, pp. 303–320.

[12] G. Gheorghe, R. Lo Cigno, and A. Montresor, "Security and privacy issues in p2p streaming systems: A survey," *Peer-to-Peer Networking and Applications*, vol. 4, pp. 75–91, 2011.

[13] E. Daniel and F. Tschorsch, "Privacy-enhanced content discovery for bitswap," in *Proceedings of the IFIP Networking Conference*, 2023, pp. 1–9.

[14] D. Kügler, "An analysis of gnunet and the implications for anonymous, censorship-resistant networks," in *Proceedings of the International Workshop on Privacy Enhancing Technologies*, 2003, pp. 161–176.