DecenArch: a decentralized system for privacy-conscious Web archiving against censorship

Simone Colombo

School of Computer and Communication Sciences
Decentralized and Distributed Systems lab
Master Thesis

July 2018

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Eleftherios Kokoris Kogias
EPFL / DEDIS

Supervisor
Kirill Nikitin
EPFL / DEDIS

Abstract

The importance of Internet is today immeasurable and its impact on our lives continues to grow at an impressive pace. However, there are several important issues to be addressed. We observe a centralization of the service providers and continuous attacks on net neutrality, which increases the risk of censorship of the data stored on the Internet, either by deletion, modification or access denial. In addition to this, surveillance by public and private actors is a constant menace. It is therefore crucial to develop systems to resist against this censorship threat, while being attentive to privacy.

We present DecenArch, a decentralized system for privacy-conscious webpages archiving. Independent servers retrieve the webpage pointed by a URL submitted by the user and agree a consensus on its content. The consensus protocol provides *privacy* for the parties, which reach a *correct result* even in presence of up to $\lfloor \frac{n}{3} \rfloor$ malicious adversaries, where n is the number of nodes in the system. The result of the consensus phase is then *cryptographically verified* by the servers and only if the entire process has been correctly executed, the webpage and its external resources are archived on a *distributed immutable ledger*. Anyone can then access the webpage stored on the ledger, thereby achieving *censorship resistance*.

Empirical evaluation of our prototype implementation shows that DecenArch running on 16 hosts archives a real-world webpage in less than 10 minutes, while the retrieval of a saved page from the immutable ledger is almost instantaneous. The time complexity of DecenArch increases polynomially with the number of hosts and the size of the webpage being stored. We argue that the system is usable for real-world purposes and achieves a good level of security and privacy guarantees.

Keywords

web, archive, decentralization, consensus, privacy, censorship-resistance

Contents

1 Introduction **5**

2 Background **7**

2.1 Plancherel's decentralized internet archive (DIA) 7

2.1.1 Overview 8

2.1.2 Discussion 11

2.2 ElGamal cryptosystem 13

2.3 Threshold ElGamal cryptosystem 14

2.4 Proofs of knowledge for equality of discrete logarithms 16

2.5 Bloom filter 17

2.6 Collective signature 18

2.7 Skipchain 20

3 System and threat models **20**

3.1 System model 20

3.2 Threat model 23

4 Overview of DecenArch **24**

4.1 Setup protocol 24

4.2 Consensus protocol 24

4.2.1 Bottom-up phase 24

4.2.2 Top-down phase, or decryption protocol 27

4.2.3 Consensus for additional data 28

4.3 Signing protocol 28

4.4 Retrieval protocol 30

5 Security and privacy analysis **30**

5.1 Privacy for conodes 30

5.2 Consensus correctness 31

5.2.1 Censorship impossibility 31

5.2.2 Implantation impossibility 33

5.3 Integrity of archived data 35

6 Prototype implementation **35**

7 Performance evaluation **36**

7.1 Number of conodes 36

7.2 Size of the webpage 38

7.3 Real-life webpages 41

8 Discussion and possible improvements **42**
8.1 Comparison with DIA 42
8.2 Overall time complexity 43
8.3 Topology 44
8.4 The choice of ElGamal 45
8.5 The consensus algorithm for structured data 46
8.6 The verification function for structured data 47
8.7 Management of additional data 49
8.8 The storage efficiency 50
8.9 The requirements about conodes 51
8.10 The right to be forgotten 52

9 Bonus: provide trusted data to smart contracts **52**

10 Conclusions **54**

1 Introduction

Internet is today part of our lives. We use it to store our data, to stay in touch with friends, to share cat pictures, to stay informed about the world and to learn about a wide variety of subjects. By looking at the current Internet architecture, we see a heavy centralization of service providers, which implies that most data are controlled by a few powerful entities. At the same time, a frontal attack on net neutrality is currently underway and fake news are part of the media landscape. This situation allows the expansion of two important and serious risks to our freedom: *censorship* and *surveillance*. Before focusing on the goals and results of this project, we present in a more detailed way these two fundamental threats.

Although censorship is today considered by too many people a prerogative of dictatorial states and dystopian books, it is more despread than ever on the Internet [15, 13]. The fact that most data and information are controlled by a small and powerful number of actors worsens the situation even more. Governments can easily force these large organization to comply with censorial or people's control laws [12, 1, 60], or deny access to non-cooperating websites [4]. Since services and data are centralized, the impact of these actions is considerably big. However, censorship is not caused only by governments. With the appearance of the so-called fake news, several big players of the information technology industry started to filter the content on their webpages with algorithms and human control [39]. While these actions may help to stop the spread of false information, the danger of entrusting profit-oriented companies with the task of dividing between legitimate and fake content is clear. Another source of problem is represented by external attackers, mostly represented by state-level adversaries [1], whose goal is to modify or delete sensitive content on targeted websites. Their job is simplified by the centralization of these data sources, which represent a perfect example of single point of failure. The continuous attacks on net neutrality [56] are also a threat to our freedom of access all the resources on the Internet. By allowing internet service providers to treat content differently, we risk blocking access to valuable sources of knowledge for the least economically able citizens. And again, with the predominant Internet design currently in place the impact of a multi-speed Internet is even bigger.

Surveillance on the Internet is another problem. Everyone is a target [3], but the most exposed people are activists against censorship or other threats to freedom, such as whistleblowers and people willing to share a document protected by a paywall [11]. Therefore, this pervasive surveillance risks making a system unusable by those who need it the most, because of the serious troubles they may encounter.

To address these challenges, we propose DecenArch, a decentralized system for privacy-conscious Internet archiving against censorship. DecenArch is based on a consensus algorithm ensuring *correctness* even in pres-

ence of up to $\lfloor \frac{n}{3} \rfloor$ malicious adversaries, while providing *privacy* for the participating parties. It guarantees *authenticity* and *integrity* of the archived content, which is then accessible to anyone on an immutable and distributed ledger, thereby achieving *censorship resistance*.

First, DecenArch introduces a consensus algorithm resistant to Byzantine adversaries [43]. This solves the most important weakness of prior work, proposed by Plancherel [52], namely the assumption of a fully honest and trusted leader for the consensus protocol. The webpage, whose URL is submitted by a user, is hashed into a Bloom filter [10], a space-efficient probabilistic data structure, by every party. The content of the filter is then encrypted using the collective key resulting from a prior execution of a distributed key generation protocol [33] for threshold ElGamal cryptosystem [32] and the ciphertext is sent to the leader of the protocol. Once the contributions of all the parties have been collected, the leader combines them using the additive homomorphic properties of the aforementioned cryptosystem. Finally, the aggregation is jointly decrypted by the parties and the content of the resulting consensus webpage is reconstructed.

Second, the result of the consensus protocol is verified and signed by all the servers, thereby ensuring the correctness of computations and the correctness of the resulting consensus view. Censoring and implanting content in the webpage to be archived is impossible thanks to zero knowledge proofs [35], produced by each node during the consensus phase, and thanks to a verification of the work performed by peer servers. The reconstruction of the webpage, done by the leader of the protocol, is also verified the other parties.

Third, the webpage, i.e. the main HTML document and the external resources, is archived on an immutable distributed ledger. DecenArch employs a skipchain [47], a data structured introduced by Nikitin et al. based on skip lists and blockchains. This data structure ensures the integrity and availability of the data. The client can submit the URL of an archived webpage to DecenArch and the corresponding content is retrieved from the ledger, without ever querying the original host of the website. Since a webpage can be archived several times to keep track of its changes, the system accepts a URL even if the corresponding page is already archived. In order to access these different versions, the user specifies an appropriate timestamp. To add an anonymity layer, the client can use any anonymous communication system, such as Tor [27], to interact with DecenArch.

Our empirical evaluations show that DecenArch can store a real-world webpage in less than 10 minutes when the distributed consensus protocol is executed by 16 servers. We argue that this time is acceptable to store a webpage, also based on the fact that in this project we do not compromise on security and privacy, even if this detracts from efficiency. Moreover, the complexity of the system increase polynomially with the number of servers and the size of the webpage. Even if the time needed to store a webpage is

not relevant per se, the throughput of the system is crucial for a production deployment. We discuss some practical solutions to improve the efficiency in the last part of this document.

This report is structured as follows. Section 2 analyzes the first version of a decentralized internet archive proposed by Plancherel [52] and presents the principal mathematical and technological tools used within DecenArch. In Section 3 introduces the system and threat models, while the proposed system is explained in Section 4. The security and privacy guarantees of DecenArch are evaluated in Section 5. Section 6 focuses on the prototype implementation. Sections 7 and 8 are devoted to the experimental results and to a discussion over the limitations of DecenArch and possible improvements. Section 9 presents an alternative utilization of the decentralized DecenArch consensus agreement algorithm as a trusted source of data for smart contracts. Section 10 presents the conclusions of this project.

2 Background

This section is divided in two parts. To start with, we introduce the first version of the decentralized internet archive (DIA), developed by Nicolas Plancherel [52] during his master's thesis, by presenting its functionalities and discussing its main limitations and weaknesses. In the second part, we present the tools that DecenArch is built upon.

For the rest of this paper we denote with $\mathbb{F}_p$ a finite field for a prime $p > 3$, $E(\mathbb{F}_p)$, sometimes abbreviated with E, an elliptic curve, $bG \in E(\mathbb{F}_p)$ a point of $E(\mathbb{F}_p)$, where $G \in E(\mathbb{F}_p)$ is a generator of a subgroup of $E(\mathbb{F}_p)$ of large prime order q and $b \in \mathbb{Z}_q^*$.

Some specific vocabulary is used throughout this report. A conode, or cothority server, is a server that takes part in a decentralized and distributed system and uses the communication protocols provided by the ONet [24] library. A roster is a set of conodes participating in the same protocol or in the same service. In the cothority framework, a protocol has only a state that lasts the time of one execution, allows conodes to communicate has an entry and an exit point and a predefined number of steps between them. A service is run by the conodes in the roster and can keep a persistent state, meaning that it can be restarted if a node crashes. The service initiates and joins protocols. It also communicates with the clients through an API.

2.1 Plancherel's decentralized internet archive (DIA)

In this section we introduce the work of Nicolas Plancherel on the first version of the decentralized internet archive [52]. The first part is devoted to an overview of DIA, while in the second part we analyze the strengths and weaknesses of Plancherel's proposal.

2.1.1 Overview

The goals of DIA are a subset ours, namely

- prevent the user from having to trust a single entity;
- create a censorship resistant Internet archive where the content to store for a given webpage is chosen and verified by all the servers participating in the protocol;
- remove individual-based content from the website before storing it;
- let the user retrieve a stored webpage without revealing any informations to the real host of the website, i.e. ensure unlinkability [51] for the user with respect to the host of the website.

To achieve these goals, DIA uses a decentralized system over which different protocols are executed. To store the data DIA relies on an immutable distributed ledger. These main building blocks are used also by DecenArch. The most important difference of DIA with respect to DecenArch is the consensus protocol used to choose the content to store for a given webpage, while other parts of the system, such as the implementation and usage of immutable distributed ledger (see Section 2.7), are identical. In the following we therefore focus on the consensus protocol, by presenting its details and discussing its strengths and weaknesses.

The consensus protocol of DIA is executed by a group of servers organized in a tree of height 1, whose root is the leader of the protocol. The DIA consensus protocol takes place as follows:

1. the leader initiates the consensus protocol (CP);
2. the leader requests the webpage from the web server;
3. the leader creates a `MasterTree` and one or more `MasterHash`es, depending on the content of the webpage. The `MasterTree` is a copy of the original HTML tree of the retrieved webpage in which all nodes are hashed, as shown in Figure 1. A `MasterHash` is created for every additional data, i.e. for every CSS file and every image present in the webpage. The `MasterHash` is a hash of the data associated to a map of signatures of the hash itself, as shown in Figure 2.
4. The leader creates a signature associated with the `MasterTree`, by signing the hash of the concatenation of the HTML nodes' hashes traversed using breath-first search. The creation of the signature is also shown in Figure 1.
5. At this point the leader has prepared all the material necessary for the CP and sends an announcement message containing the URL provided

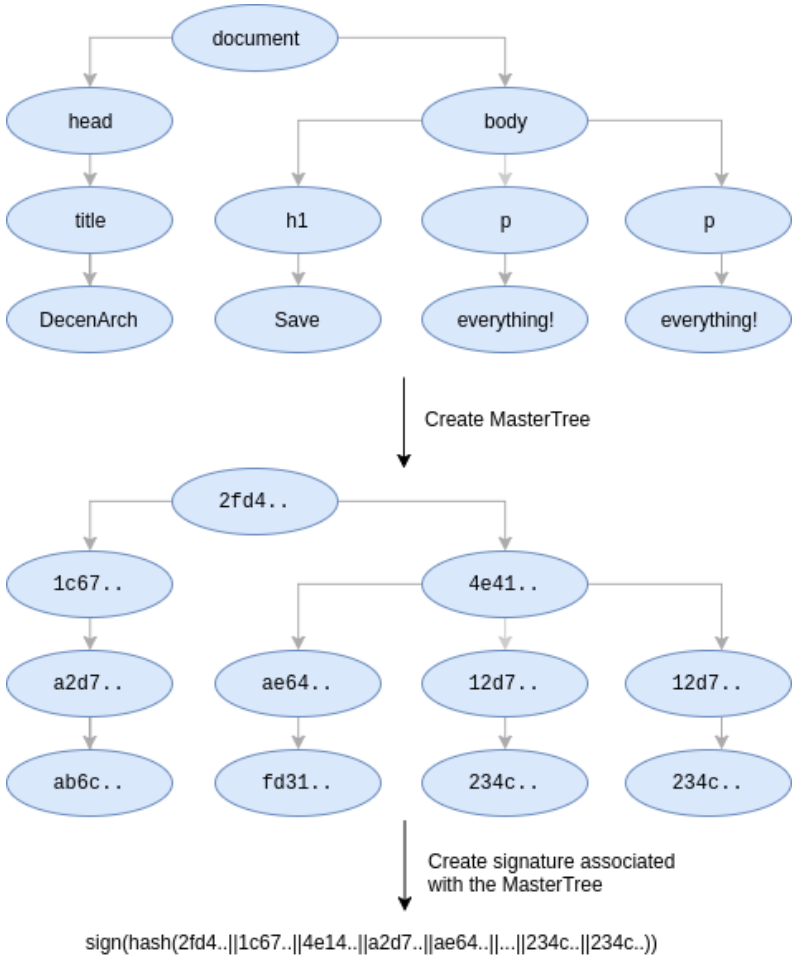

Figure 1: Algorithm to create the `MasterTree`.

by the user, the `MasterTree` and the `MasterHash`es to all the conodes, including himself.

6. Upon receiving the announcement message, every conode execute the following steps

   - the conode gets the webpage from the web server and computes the hashed HTML tree, called `LocalTree`, and all the needed `LocalHash`es. These objects are the conode's equivalent of the `MasterTree` and `MasterHash`, respectively.
   - The conode stores all the plaintext data, i.e. HTML page, CSS files and images, locally;
   - the conode compares the `LocalTree` with the `MasterTree` and the different `LocalHash`es with the corresponding `MasterHash`es.

Figure 2: Procedure to create the `MasterHash`.

After comparison, the conode creates a signature associated to the `MasterTree` (see Figure 1) and adds its signature to the appropriate `MasterHash`es, according the `LocalHash`es. Moreover, the conodes fill the `SeenArray`, indicating which HTML nodes the conode has compared to the `MasterTree`. In other words, the conode compares the content of its view of the webpage with the content of leader's view of the same webpage. The algorithm to create both the signature and the `SeenArray` is shown in Figure 3.

- The last step depends on the role of the conode:
  - if it is a leaf, it simply sends `LocalTree`, `MasterHash`es, `SeenArray`, signature associated with the `MasterTree` to the leader.
  - If it is the leader, it takes the nodes of the `MasterTree` and the additional data that have been seen by at least a threshold number of children, by checking the `SeenArray`s for the former and the signatures contained in the `MasterHash`es for the latter. During this process the signatures of the consensus material received from the children are verified and a child contribution is take into account only if the signature is valid.

In the real implementation, all this process is executed sequentially for the HTML page and for every additional data, but without loss of generality we can assume here that the HTML document, the CSS file and all the images are processed in parallel.

Once the leader has the result of the consensus protocol, it reconstructs the HTML page from the tree and sends it, together with the additional data, to all the conodes for the signing step. Note that in this case the conodes do not have any possibility to check that the leader does not temper with the consensus result during the reconstruction process.

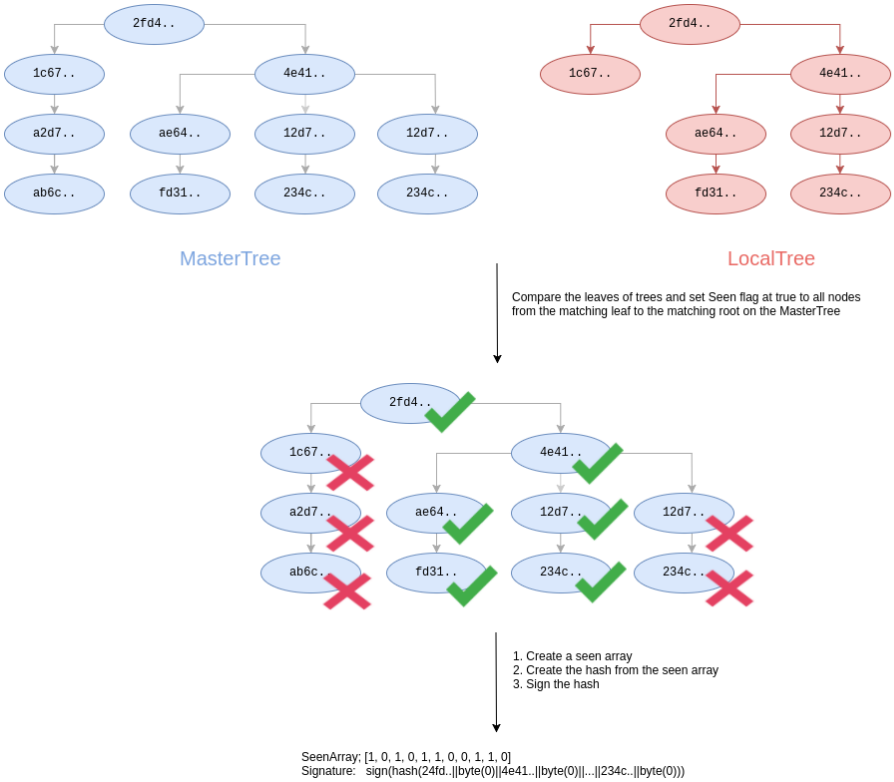

Figure 3: Procedure to create the `SeenArray` and the signature associated with the `MasterTree` from the `LocalTree`.

2.1.2 Discussion

We discuss now the principal limitations of DIA. Note that here we present only the negative aspects of DIA, but the work of Nicolas Plancherel is interesting and gave us an important and solid starting point from which to develop DecenArch.

The trusted leader assumption The leader has absolute control over the result of the consensus protocol and therefore it must be considered trusted. We shortly explain why the leader has complete control over the consensus protocol's outcome, and we then discuss the implications of this absolute power.

As highlighted in Section 2.1.1, the consensus algorithm for the webpage is based on a reference tree, called `MasterTree`, which is compared to the locally computed tree by all the conodes. This clearly imposes the leader to produce an honest reference. If the leader is dishonest, he can easily censor any part of the website without being detected: removing some nodes from the `MasterTree` is equivalent to remove them also from the final consensus

result and therefore from the stored webpage.

The power given to the leader, and therefore the obliged trusted leader assumption, is an important limitation of DIA, because it greatly weakens the decentralization of the whole system. In fact, we have a centralized system, where a trusted third party can decide to take into account the proposals of a bunch of other servers, but it is also free to completely ignore them. Not only the decentralization objective is impacted, but also the goal of building a censorship resistant archive. As explained, the leader can censor any content from the final consensus result, and he can also modify the consensus tree before the collective signing phase.

The lack of privacy for the conodes To improve efficiency and to provide privacy for the cothority servers, Plancherel proposes to hash the nodes of the HTML tree before publishing them. The problem is that the HTML nodes are directly hashed a published, without any kind of protection, e.g. a salt. It is therefore easy for an attacker to test whether a conode has seen a given HTML node, just by computing the hash of the node and test if it is present in the supposedly anonymized tree proposed by the conode. The fact that everything is sent in clear (see next paragraph), makes the impact of this issue even greater.

Other minor implementation issues The transition from the final consensus tree reconstructed by the leader to the HTML code collectively signed by all the conodes is a critical point. Indeed, in Plancherel's implementation, the conodes sign whatever the leader proposes, without checking if the proposal is the real result of the consensus protocol. To correct this the nodes can check the signed `SeenArrays` before signing the leader's proposal. Note that this verification only solves the implanting problem, while a malicious leader can still censor the webpage content.

If a link to an external resource in the retrieved HTML page is given in absolute path (`http://...`), the resource is rightly considered as additional data and therefore stored on the immutable distributed ledger, but since there is no edition of the HTML page, the user willing to open the saved page in his browser will request the image from the web server, instead of using the stored one. This makes saving the images useless and above all it violates the DIA's unlinkability requirement for the user with respect to the web server.

The communications between user, cothority servers and entities are not encrypted nor authenticated. This was actually a limitation of the first version of ONet [24], which only allowed communications over TCP, without adding the TLS layer. To conclude, data are stored uncompressed on the distributed ledger. Since the consensus HTML page and its additional data are all stored in a single block of the ledger, the absence of compression can

lead to potentially big blocks and therefore greatly affects DIA's efficiency.

2.2 ElGamal cryptosystem

For public-key cryptography, DecenArch relies on the probabilistic and additively homomorphic[1] ElGamal cryptosystem [32]. The ElGamal cryptosystem works for any family of groups for which the discrete logarithm (DLOG) problem is considered to be intractable. Part of the security of the scheme actually relies on the Diffie-Hellman assumption, which implies the intractability of the DLOG computation [62]. For DecenArch we use elliptic curve ElGamal, using the curve E introduced at the beginning of the section.

We will now briefly describe the cryptosystem using elliptic curves; because of this we stick for the rest of the report to the additive notation, instead of the most common multiplicative notation. The public parameters are the elliptic curve E and a point $G \in E$ of large prime order q, which generates the cyclic subgroup of order q of E. The private key is an integer $k \in \mathbb{Z}_q^*$ and the public key is $K = kG$. A point $P_m \in E$ is encrypted under public key K by computing

$$\texttt{Enc}_K(P_m) = (R, C) = (rG, P_m + rK),$$

where $r \in_R \mathbb{Z}_q^*$. To decrypt the ciphertext (R, C), it is sufficient to use the private key k to compute

$$\texttt{Dec}_k((R, C)) = C - kR = P_m + rK - k(rG) = P_m + rK - rK = P_m.$$

The plaintext point P_m is obtained as a result.

As said before, the elliptic curve ElGamal cryptosystem is additively-homomorphic. Let (R_1, C_1) and (R_2, C_2) be two ciphertext encrypting P_{m0} and P_{m1}, respectively. Then the additively homomorphic property result from

$$\begin{aligned}(R_0, C_0) + (R_1, C_1) &= (R_0 + R_1, C_0 + C_1) \\ &= ((r_0 + r_1)G, P_{m0} + P_{m1} + (r_0 + r_1)K)\,,\end{aligned}$$

whose decryption returns $P_{m_0} + P_{m_1}$.

There is however an important challenge to solve: how to encrypt an integer $i \in \mathbb{Z}_q$? This integer should be mapped to a point $P_i \in E$ using an appropriate function $\texttt{map} : \mathbb{Z}_q \to E$ defined as follows

$$i \mapsto i \cdot G,$$

[1] Note that the ElGamal cryptosystem is not always additively homomorphic, this depends on the group used in the system.

where in this case $\cdot$ represents the multiplication of the point G by the scalar i. Note that encrypting 0 using the `map` function is perfectly secure, since the random value r, called sometimes blinding factor, ensures the semantic security of the ciphertext. We use this mapping function to exploit the additive homomorphic property presented above:

$$P_{m_0} + P_{m_1} = m_0G + m_1G = (m_0 + m_1)G.$$

Finally, by computing the DLOG for $(m_0 + m_1)G$ we can retrieve $m_0 + m_1$. This may sound strange, since the security of the cryptosystem is based exactly on the DLOG problem intractability. However, since our plaintexts are only small numbers, we can solve it in a reasonable time. Some implementation's efficiency improvements, such as the usage of a map to store the already computed value, make the computation even faster.

The used cryptosystem should fulfill another requirement. We want to be secure that an attacker does not gain any information about the plaintext encrypted by a given ciphertext even if he knows the list of possible plaintexts encrypted by the given ciphertext. This concept is called indistinguishability under chosen-plaintext attack (IND-CPA) [36]. The following theorem, presented here without proof, states that, under some assumptions, ElGamal is IND-CPA:

Theorem 1 ([62, Theorem 5.4]) *If the decisional Diffie-Hellman problem (DDH) is hard in the group generated by the ElGamal cryptosystem, and if the plaintext space is included in the group, then the cryptosystem is IND-CPA secure.*

A detailed discussion about elliptic curve cryptography is out of the scope of this report, therefore we again state without proof that the DDH problem is hard for the group generated by the ElGamal cryptosystem, i.e. the subgroup generate by $G \in E$, and the plaintext space is included in the aforementioned group. We conclude that it is impossible to determine the plaintext of a ciphertext by looking at it. This property is particularly important when encrypting Bloom filters, presented in Section 2.5. Indeed, thanks to the IND-CPA property, we are sure that it is impossible to tell if an entry in the filter contains 0 or 1.

2.3 Threshold ElGamal cryptosystem

The goal of a threshold scheme for public-key encryption [26] is to allow a group of parties to encrypt a message, whose decryption requires the cooperation of at least a threshold number of participants, so that no minority is able to perform this operation by themselves. To achieve this goal, a shared public key K is generated, but the corresponding private key k is never computed. Instead, every party has a share s_i, where i indicates the

participant, and only the combination of a threshold number of these shares allows the decryption of a ciphertext encrypted with K. Security properties of the cryptosystem must be ensured even in the presence of malicious contributions to the key generation. A threshold cryptosystem is therefore composed of two main protocols:

- a *key generation* protocol to generate the shared public key and the shares of the distributed private key for each participant.
- a *decryption* protocol to jointly decrypt a ciphertext without reconstructing the private key. This protocol should successful only if a threshold number of parties collaborate.

For the ElGamal cryptosystem described in the previous section, a solution for the first protocol has been proposed by Gennaro et al. [33], while for the second we use the approach described in [18].

Distributed key generation The key generation protocol proposed by Gennaro et al. in [33] is an improvement[2] over the protocol proposed by Pedersen [50]. The protocol starts by running a commitment phase where each party P_i commits to a t-degree polynomial (t being the threshold of the scheme) $f_i(s)$, for $s \in \mathbb{Z}_q^*$, whose constant term $f_i(0) = s_i$ is the random contribution of P_i to the jointly generated private key k. The realization of this commitment phase is based on the verifiable secret sharing (VSS) protocol proposed by Pedersen [49], which ensures that once the commitment stage is terminated the secret key k is fixed and cannot be modified by any malicious party. Moreover, at the end of this VSS protocol, each honest party produces an equal set of so-called qualified servers $QUAL$, thereby excluding malicious party from further participating in the key generation protocol. After the value k is determined, the public key $K = kG$ is computed in an efficient and secure way, without actually computing the secret value k. This is done by running another VSS protocol between the qualified parties. Finally, the shared public key is:

$$K = \sum_{i \in QUAL} z_i G.$$

We stress that the private key k such that $K = kG$ is never reconstructed by the parties.

[2]Gennaro and coauthors show that the protocol suggested by Pedersen does not guarantee a uniform random distribution for the generated key. They however prove that the Pedersen's protocol is secure for schemes whose security is proven to be equivalent to the hardness of computing DLOG, such as ElGamal. However, we decide to use the protocol proposed in [33] to have a more robust DKG protocol, e.g. in case we want to extend DecenArch, and for historical reasons. For an in-depth explanation about the limitations of Pedersen's proposal see [33].

Decryption To decrypt a ciphertext $(R, C) = (rG, P_m + rK)$, where K is the shared public key, without reconstructing the shared private key k, each party decrypts the ciphertext with its own share and sends the partial decryption to the leader. If the leader receives at least a threshold number of partial decryptions, it can reconstruct the original plaintext using Lagrange interpolation, as highlighted in [18]. Note that we have to be sure that the party indeed sends a correct partial decryption and not some garbage value. This would result in a wrong Lagrange interpolation and therefore a wrong reconstruction result. A solution is to use a DLEQ proof, presented in the next section, to prove that the party knows the private share z_i [18] and performed the partial decryption using the correct ciphertext. We do not go into the details of this DLEQ proof since we introduce this concept later and the proof construction is similar to the content proof construction, also introduced in the next section. The interested leader should refer to [18, 58].

2.4 Proofs of knowledge for equality of discrete logarithms

We present here one of the most important building blocks of DecenArch, namely the zero-knowledge [35] proof for the equality of discrete logarithm (DLEQ). The goal of this proof is that, given two elliptic curve points xG and xH, one can check that $\log_G xG = \log_H xH$, without revealing the secret value x. The efficient protocol to build this proof is a Σ-protocol [20] due to Chaum and Pedersen [14], which works as follows:

1. Alice chooses a commitment v and sends $a = vG$ and $b = vH$ to Bob.
2. Bob chooses a random challenge $c \in \mathbb{Z}_q^*$ and sends it to Alice.
3. Alice sends $r = v + cx \mod q$ to Bob.
4. Bob checks that $a = vG = rG + c(xG)$ and $b = vH = rH + c(xH)$

For implementation purposes, the protocol is turned into a non-interactive proof using the Fiat-Shamir heuristic[3] [29].

Thanks to the DLEQ proof we can prove that a ciphertext is the encryption of either a message m_0 or a message m_1. This is important to prove the content of encrypted Bloom filters (see Section 2.5). Suppose we have two elliptic curve points P_{m_0} and P_{m_1}, where P_{m_0} and P_{m_1} are appropriate mappings from m_0 and m_1 to E, respectively. We want a proof that a ciphertext is the encryption of one of the two values and nothing else. Clearly, decrypting the ciphertext is not an option and the proof should not

[3]Note that the security of the Fiat-Shamir heuristic is proved in the random oracle model and here a hash function combined with an extensible output function is used to instantiate the random oracle. A discussion about this instantiation step is out of the scope of this report, for further information about the subject see [8, 62] and the `dleq` package in Kyber [23].

reveal anything about which value as been encrypted. Consider an ElGamal ciphertext of the following form

$$(R, C) = (rG, P_m + rK), \quad \text{with } P_m \in \{P_{m0}, P_{m1}\}$$

To show that the pair (K, C) is the encryption of either P_0 or P_1 without revealing which one, it is sufficient to produce a DLEQ proof indicating that

$$\log_G R = \log_K rK.$$

Note that the secret value of this proof is the blinding factor r, known only to the encrypting party. The verifier then can check proof with two different inputs, namely

$$\log_G R = \log_K C - P_0 \quad \vee \quad \log_G R = \log_K C - P_1.$$

If the prover is honest, exactly one of the verifications return true by construction. The verifier can check the statement because he knows (R, C), which is the ciphertext, and the points representing the two messages supposed to be encrypted, i.e. P_0 and P_1. Note that this proof must be generated for every ciphertext and this results in a vector of proofs when dealing with encrypted Bloom filters, presented in the next section. We refer to this proof as *content proof* in the rest of the report.

2.5 Bloom filter

Bloom filters are probabilistic space efficient data structures allowing for membership queries over a given set [10]. A Bloom filter uses k hash functions $h_1, h_2, \ldots, h_k$ to hash elements from a set S with cardinality n into a m-bit vector V, where $h_i : S \to [0, m[$, for $i = 1, \ldots, k$. To insert an element $s \in S$ in the filter, the bits at positions $h_1(s), \ldots, h_k(s)$ in the vector are set. To test for membership, the element is again hashed into k indices with the same k hash functions. If the bits of all k indices are set, then the element is in the set; if at least one of the k bits are unset, then the test return false. Note that thanks to the optimization technique presented by Kirsch and Mitzenmacher in [40], two hash functions are sufficient to compute the k indices. For the Bloom filter of DecenArch, two cryptographic secure hash functions are used. This is important to fulfill the security and privacy requirements of the system, as we will explain later. It is important to note that this data structure allows a probability of a false positive error, i.e. a membership test can return a positive result for an element $z \notin S$, but no false negative error. The impact of a non-zero false positive probability on the design of DecenArch is discussed later in the report. The probability of error is defined by the filter's false positive rate ϵ. Observe that the

probability that a specific bit is still 0 after inserting n elements in a vector of size m is exactly

$$\left(1 - \frac{1}{m}\right)^{kn}.$$

Therefore the probability that k specific bits are set to 1 is

$$\epsilon = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{\frac{-kn}{m}}\right)^k.$$

This equation is minimized by $k = \frac{m}{n} \ln 2$. Therefore, after some computations, we have the following closed formulae for the optimal m and k given the cardinality n of set and the false positive rate ϵ:

$$m = -\frac{n \ln \epsilon}{(\ln 2)^2}, \quad k = \frac{m}{n} \ln 2.$$

The limitation of classical Bloom filters is that they can represent only sets and not multisets. Since we want to work with a threshold approach, we need to find a way to count how many times an element has been added to a filter. The solution to this constraint are spectral Bloom filters [16], an extension of the classical Bloom filter to multisets, which allows to estimate the multiplicity of an element with a small error probability. A spectral Bloom filter replaces the m-bit vector V with a vector of m counters, C. All the counters in C are initially unset and when inserting an item s we increase the counters $C_{h_1(s)}, \ldots, C_{h_k(s)}$ by 1. Therefore, the spectral Bloom filter stores the multiplicity of each item. To estimate the multiplicity of an element s in the spectral Bloom filter, we use the *minimum selection* estimator [16], i.e. the multiplicity of s is given by $\min_{i=1}^{k} C_{h_i(s)}$. Note that also in this case we can have a false positive error, whose probability is the same as classical Bloom filters [16]. Since a classical Bloom filter is basically a spectral Bloom filter with a maximal counter value of one, we sometimes omit the word spectral.

The encryption of a Bloom filter, both classical and spectral, is defined as the separate encryption of each counter composing the filter. Let B be a spectral filter, then

$$\texttt{Enc}_K(B) = [\texttt{Enc}_K(C_1), \ldots, \texttt{Enc}_K(C_m)].$$

2.6 Collective signature

For reasons which will be clear later, for DecenArch we need a scalable and efficient cryptographic primitive implementing multisignatures [46]. We use CoSi [61], which enables a leader to request the collective signature of a message to a group of servers, or witnesses. We give here a high-level

presentation of CoSi, and we let the interested reader refer to [61] for more details.

CoSi is build on one of the simplest and oldest signature schemes: the Schnorr signature algorithm [57], which supports multisignatures. Schnorr signing can be viewed as a Σ-protocol [20], i.e. a three-move protocol, which is made non-interactive using the Fiat-Shamir heuristic [29]. We explain shortly how Schnorr multisignatures on elliptic curve works, while we let the interested reader refer to the appropriate paper for Schnorr signatures. Consider a group of N signers with individual private keys $k_1, \ldots, k_n$ and the corresponding public keys $K_1 = k_1 G, \ldots, K_N = k_N G$. The aggregate public key is computed from the individual keys as $K = \sum_{i=1}^{N} G k_i = G \sum_{i=1}^{N} k_i$. To sign a message M, the N signers adopt the following procedure. Each signer i chooses a random and uniformly distributed secret r_i and shares the commit $R_i = r_i G$ with the other parties. The leader collects all the N commits, computes a collective commit $R = \sum_{i=1}^{N} R_i$ and use a cryptographic secure hash function to compute the challenge $c = H(R||M)$, where $||$ indicates concatenation. The leader broadcasts the challenge c to all the signers, each of whom computes its share $s_i = r_i - c k_i$ and sends it back to the leader. Finally, the leader computes the aggregation of the shares $s = \sum_{i=1}^{N} s_i$ and outputse the collective signature (s, c). The verification of the signature can be done using the classical Schnorr signature verification algorithm.

To make multisignatures scale to a large group of participants, CoSi split the communication and computation costs of generating multisignatures across a tree, which has height three in the implementation [22] used by DecenArch. To collectively sign a message M, the following protocol is executed on the tree:

1. Announcement: The leader sends the announcement containing the message M to be signed down to the tree.

2. Commitment: Starting from the laves, each node computes its individual commit and sends it to the parent. Each parent computes aggregate the commits of the children with its own commit and passes the aggregation up to its parent, unless it is the leader. Therefore, the root receives an aggregate commitment from all nodes of the three.

3. Challenge: The leader computes a collective challenge and sends it down to the three.

4. Response: In a final bottom-up phase, each leaf send its individual response to its parent. Each parent aggregate the individual responses from the children with its own individual response and passes the aggregation up to its parent, unless it is the leader. The final signature is finally outputted by the leader.

Note that, for signature verification the public key of the witnesses should be known. Moreover, if a witness is offline during the signing process or refuses to sign a message, the signature resulting from the above protocol will containts metadata that documents the circumstance.

2.7 Skipchain

To securely store the retrieved webpages, DecenArch relies on an immutable distributed ledger. In particular, DecenArch uses a skipchain, an authenticated data structured introduced by Kirill et al. [47] which combines ideas from skip lists and blockchains. The main difference between a normal blockchain and a skipchain is the presence of forward links, together with the classical backward links. As in regular blockchains, backward links are cryptographic hashes of previous blocks, while forward links are cryptographic signatures of next blocks, added retroactively when the target block is appended to the chain. Skipchains are designed to allow an efficient traversal of short or long distances thanks to the presence of multi-hops links, both in forward and backward direction. The consensus algorithm used by the ledger is ByzCoinX [42] developed by Kokoris Kogias et al., which is an improved version of BFTCoSi introduced in ByzCoin [41]. ByzCoinX uses Byzantine consensus to allow immediate transaction validation, thereby efficiently ensuring a single consistent timeline. An in-depth explanation of the ByzCoinX algorithm is out of the scope of this report and the interested reader should refer to the appropriate papers.

Note that for DecenArch we actually use a skipchain as a regular blockchain, since we only need the regular backward and single-hop links. However, since this project is developed within the DEDIS lab, we stick to the previous work produced by the lab. Because of our basic usage the powerful skipchain, in the rest of the report we use skipchain and blockchain interchangeably to refer to the immutable ledger used by DecenArch.

3 System and threat models

3.1 System model

Our system, as depicted in Figure 4, consists of four main entities:

- the user asking to save or retrieve a webpage;
- the web server on which the webpage is hosted;
- the conodes responsible for retrieving and archiving the webpages (the blue square in Figure 4). We refer to this roster as the DecenArch roster, and we denote the number of servers composing it with n_d. The servers of this roster are sometimes referred to as archiving conodes.

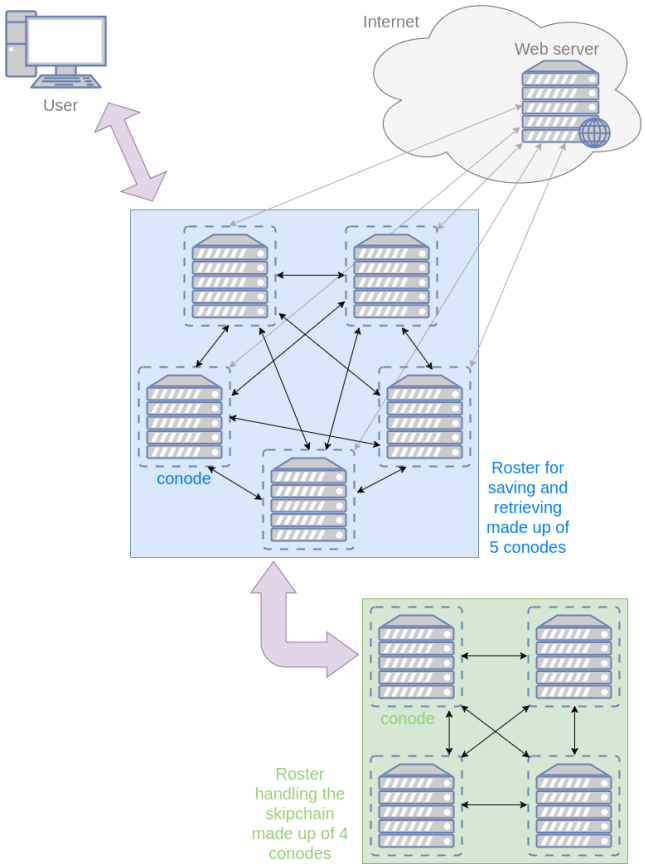

Figure 4: DecenArch model, with the four entities highlighted. The bold coloured arrow between the two entities indicates a communication between entity, while the thin arrows represent a communication between two machines. In this example we have $n_d = 5$ and $n_s = 4$. Image adapted from [52].

- The conodes responsible for handling the skipchain (the green square in Figure 4). We refer to this roster as the skipchain roster, and we denote the number of servers composing it with n_s.

The flow between the different entities can be described as follows. The user provides a URL to the DecenArch roster, which request the webpage pointed by the URL from the web server and reach a consensus on a unified view. This view includes the CSS files and the images attached to the webpage. Once the consensus is reached, its result is verified and signed by the archiving servers. Finally, the data are stored on the immutable distributed ledger, handled by the appropriate roster. Anyone can then securely retrieve the view from the ledger, by submitting the URL and an optional timestamp to the skipchain roster. Once the data are retrieved

from the ledger, the signature attached to them is verified and if it is valid, the reconstructed webpage is sent to the user.

It is important to note that the two different rosters can be composed of the same machines, or one roster can be a subset of another roster. Moreover, when we indicate a communication between two entities, it may happen that the communication takes effectively place between only two machines, but in this case the machine is seen as being *part of an entity*, i.e. the machine involved in the communication could potentially be replaced by any other machine that is part of the entity.

We now discuss the functionalities that DecenArch must provide along with its security and privacy requirements.

Functionality DecenArch must permit to *store any webpage accessible on the World Wide Web* by providing its URL. The data to be stored are determined by a *consensus protocol* executed by the conodes of the DecenArch roster. The consensus is computed for the main HTML file, for the CSS files attached to it and for all the images of the page. We do not require DecenArch to reach a consensus, and therefore to store, any other file present in the webage, e.g. a video. The consensus for the webpage and its additional data is defined as a *threshold set-union* over the content retrieved by every conode. Content viewed only by a number of conodes smaller that the threshold should not appear in the consensus result. Prior to storage, the view of the consensus should be *verified* and *signed* by all the servers in the DecenArch roster. Moreover, DecenArch should allow anyone to *retrieve* a previously saved webpage, at any time and from any location. If a page is saved multiple times, a user should be able to provide a timestamp to identify the correct version of the page. The webpage stored closer to the timestamp should be returned.

Security and privacy The consensus algorithm used by DecenArch should guarantee the *correctness* of the result in the Byzantine model [43], where less than $\frac{1}{3}$ of the participating servers are malicious and collude to alter the output of the algorithm. DecenArch should protect the *privacy of the conodes* involved in the archiving process[4]. This means that an attacker, who can be an external entity, a single conode or a group of colluding conodes, is able to see only the data included in the final consensus view, while the data seen only by a single conode are never disclosed. The data must be *collectively signed* prior storage to allow later inspection. DecenArch should be *publicly verifiable*, i.e. it should allow any entity to check the correctness of the system's computations even after these have been performed.

[4]Due to the difficulty to agree on a common definition, we refer in this work to the definition of privacy given by prof. Hubaux in the course Advanced Topics on Privacy Protection at EPFL: "Privacy is the ability of individuals to determine when, how and to what extent information and data about themselves is revealed to others."

Finally, DecenArch should protect the *integrity of the data* resulting from the consensus protocol, which cannot be modified.

3.2 Threat model

We present here DecenArch's thread model by discussing the role of and the assumptions about each entity in the system.

DecenArch roster We assume that a threshold $t_d > \frac{2}{3}n_d$ of the n_d conodes in the roster responsible for executing the retrieve and save protocols are honest, meaning that less than t_d, i.e. less than $\frac{1}{3}n_d$ are compromised and can collude to tamper with the saving or retrieving process. We indicate the number of malicious servers in the DecenArch roster with m_d. In particular, the malicious conodes can interfere with the consensus process to implant and/or remove information with respect to the consensus reached in the ideal world, where all the servers are honest and functioning. Moreover, since the same conodes responsible for the consensus protocol participate in the collective signing of the consensus view, the malicious cothority servers can interfere with the signature process, e.g. by trying to convince honest conodes to sign something different from an ideal world consensus result.

Skipchain roster The roster responsible for handling the immutable distributed ledger, i.e. the skipchain, is viewed and use as a black box by DecenArch. Therefore, we use the same threat model as the one presented in the Chainiac paper [47], in which skipchains are introduced. Given the number n_s of servers in the skipchain roster, among which m_s are malicious, we require that $n_s \geq 3 \cdot m_s + 1$, otherwise the consistency of the timeline is not guaranteed. Note that this bound on malicious servers is the same as the one required for the DecenArch roster. For an in-depth discussion about the adversarial assumptions for skipchains see [47, 41].

Data providers Data providers can be malicious. In particular, they may censor the content on the basis of the origin of the requests, e.g. by removing some content from the webpage is the request comes from a specified range of IP addresses. Moreover, the data providers can collude with the malicious servers of the DecenArch roster to break the security and privacy guarantees of the system.

An attack on DecenArch is successful if an attacker succeeds in accomplishing at least one of the following:

- Provide evidence that a conode has seen a given HTML leaf by looking at the consensus material produced by the conode and its peers.

- Implant one or more leaves in the consensus HTML page that would not have been present in the consensus reached by a roster composed of only fully-honest and functioning nodes.
- Remove one or more leaves in the consensus HTML page that would have been present in the consensus reached by a roster compose of only fully-honest and functioning nodes.
- Similarly, implant and/or remove one or more additional data that would, respectively would not, have been present in the consensus reached by a roster composed of only fully-honest and functioning conodes.
- Once a consensus is reached and signed, temper with the consensus data, i.e. HTML page and additional data.

4 Overview of DecenArch

In order to achieve the security and privacy requirements introduced in Section 3.1 DecenArch is developed as a modular, decentralized and distributed system composted of different protocols. In this section we describe the different modules on which DecenArch is built upon.

4.1 Setup protocol

Two things are needed for DecenArch in order to run properly: a collective public key K generated using the DKG protocol presented in Section 2.3 and a skipchain. During the setup phase a randomly selected conode acting as a leader starts the DKG protocol and initializes a new skipchain, i.e. it initializes the skipchain roster and creates a genesis block, which does not contain any data. We stress that the skipchain roster is independent of the DecenArch roster. In particular, the skipchain is initialized only once, while the DKG protocol generating the collective key for the servers of the DecenArch roster can be executed several times. Imagine for example that a conode joins the DecenArch roster later. In this case, a new DKG protocol must be executed, because the new conode need its private share of the never computed collective private key k in order to take part to the consensus protocol. DecenArch handles this case and support the execution of several DKG protocols to cope with the joining of new servers.

4.2 Consensus protocol

4.2.1 Bottom-up phase

For the consensus protocol over structured data, i.e. over the HTML document, the conodes are structured to communicate in a 1-level tree (flat)

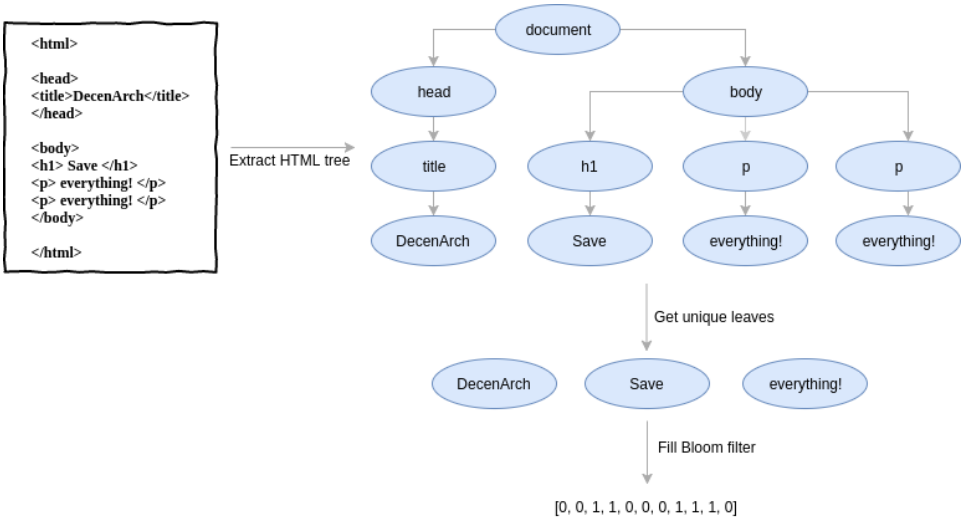

Figure 5: Procedure to store the HTML document within a Bloom filter.

topology. We discuss the possibility to use different topologies in Section 8.3. The *archiving conodes* are randomly inserted into the tree by the randomly selected root, which act as leader of the protocol. The protocol is started by the leader when the user initiates a save request by submitting to DecenArch the URL of the webpage he wants to be archived. Upon receiving the URL, the leader download the webpage and computes the optimal parameters for the Bloom filter (see Section 2.5). Table 1 shows the parameters for some unique leaves values. We will return on these numbers in Section 7, where we analyze the performance of DecenArch. Only the unique leaves of the HTML tree are used to compute the optimal parameters of the Bloom filter, since only these are stored in the filter by each conode. The leader then broadcast the announcement, containing the URL and the filter's parameters to all the children.

Leaves	m	k
64	614	7
128	1227	7
256	2454	7
512	4908	7
1024	9816	7
2048	19631	7

Table 1: Bloom filters' parameters for some unique leaves values. Recall that m is the length of the filter and k represents the number of hash functions.

When a node receives the announcement, it downloads the HTML content of the webpage and extracts the unique leaves form the correspond-

ing HTML tree; the leaves are then inserted into a classical Bloom filter, constructed using the parameters specified by the leader. This procedure, depicted in Figure 5, is performed also by the root. All the nodes encrypt then the local filter with the collective public key K as explained in Section 2.5. Moreover, they produce a cryptographic proof stating that every bucket of the encrypted filter contains either a 0 or a 1, i.e. the content proof for the filter. Recall that the encryption of a filter is defined as the separate encryption of each value of the filter. Therefore, to generate the content proof the node produces a non-interactive zero-knowledge proof for the equality of discrete logarithms (DLEQ), introduced in Section 2.4, for every ciphertext stating that $\log_G R = \log_K rK$, where the ciphertext is $(R, C) = (rG, P_M + rK)$. Finally, the filter is hashed and signed by the conode using its own private key k_c. Then, depending on its position in the tree, every node behaves as follows.

Leaf node The node sends the encrypted Bloom filter, the content proof and the signature of the hashed ciphertext to the leader of the protocol.

Root node The node waits for the consensus material of all its children. If a child does not send the material in a reasonable amount of time, a timeout is triggered and the child is ignored. Upon receiving all the inputs, the node iterates over them and performs the following. The content proof and the signature of the encrypted filter are verified. Recall that the content proof is composed of m DLEQ proofs, i.e. a proof for every ciphertext of the filter, stating that: $\log_G R = \log_K rK$, where the ciphertext is $(R, C) = (rG, P_M + rK)$. To verify the proof the leader uses the mapping function `map` to compute P_0 and P_1, which are the mappings of 0 and 1, respectively. Then it verifies the DLEQ proof produced by the child with two different inputs, namely

$$\log_G R = \log_K C - P_0 \quad \vee \quad \log_G R = \log_K C - P_1.$$

If the child is honest, one of these checks returns true. Indeed, suppose that the ciphertext encrypts P_0, then

$$\begin{aligned} \log_G R = \log_K C - P_0 &\implies \log_G R = \log_K P_0 + rK - P_0 \\ &\implies \log_G R = \log_K rK, \end{aligned}$$

which is true by construction.

If the verifications are successful, the encrypted filter is added to the leader's filter, by leveraging the additive homomorphic property of the elliptic curve ElGamal cryptosystem (see Section 2.2). On the other hand, if the verification fails, the contribution is ignored. The leader sets the ciphertext for the verification of the child's content proof to the received encrypted filter. Once all the children Bloom filters have been added, the node produce

the zero-knowledge proof for the aggregation. This proof consists in publishing all the received encrypted Bloom filters, the local filter of the leader and the result of their sum. Table 2 shows the aggregation proof, where the contributions are the encrypted filters of all the children, i.e. $\texttt{Enc}_K(B_{c_1})$, ..., $\texttt{Enc}_K(B_{c_n})$, together with the contribution of the root, i.e. $\texttt{Enc}_K(B_r)$, and the aggregation is the sum of all the contributions, computed using the additive homomorphic properties of the cryptosystem. Due to the semantic

Contributions	Aggregation
$\texttt{Enc}_K(B_{c_1})$	
$\texttt{Enc}_K(B_{c_2})$	
...	$\sum_{B \in C} \texttt{Enc}_K(B)$
$\texttt{Enc}_K(B_{c_n})$	
$\texttt{Enc}_K(B_r)$	

Table 2: Aggregation proof, where $C = \{B_{c_1}, \ldots, B_{c_n}, B_r\}$.

security of the ElGamal cryptosystem, publishing the ciphertexts does not leak any information about the corresponding plaintexts. In order to verify an aggregation proof, a verifier can simply compute the sum of all the ciphertexts, as shown for the aggregation in Table 2, and check that the result corresponds to the published result. Finally, the root collects the proof material, i.e. content proof and ciphertext signature, of all the children, add its own material, i.e. content proof, signature and aggregation proof, and broadcast the complete proofs of the consensus protocol to all the leaves.

4.2.2 Top-down phase, or decryption protocol

Once the consensus has reached the root of the tree, we have an encrypted spectral Bloom filter containing the contributions of all nodes. To decrypt the filter the decryption protocol for threshold ElGamal cryptosystem (see Section 2.3) is used. The root sends the ciphertext to each archiving server, which in turn decrypts the ciphertext with its own share of the private collective key k. The server sends the partial decryption and the DLEQ proofs for the correctness of the decryption to the leader, which accepts the partial decryption only if the proof is valid, otherwise the contribution is ignored. Finally, the root reconstructs the consensus Bloom filter using Lagrange interpolation.

Once the consensus Bloom filter has been reconstructed, the leader reconstructs the HTML code of the webpage. To accomplish this, the leader takes its local HTML tree and iterates over all the leaves. If the multiplicity of a leave in the consensus filter is equal or greater the threshold t_d, the leave is added to the consensus HTML tree. We use t_d as threshold because it represents the number of honest servers in the roster. Once the leaves of the leader's local tree have been verified, the resulting tree is parsed into a

proper HTML document. As the reader can imagine, this procedure inserts in the consensus HTML tree only the laves present in the leader's local tree. We discuss this crucial point further in Section 8.5.

4.2.3 Consensus for additional data

After the consensus for the HTML document, the consensus protocol for unstructured data, i.e. CSS files and images, is executed. The protocol used by DecenArch is exactly the same as the one used by DIA, presented in Section 2.1. Essentially the algorithm consists in sharing the hash of the image or CSS file and including in the consensus result only the data whose hash appears at least a threshold number of times. We stress that there is no verification on the additional data inserted in the consensus result by the leader. We discuss this issue in Section 8.7. Refer to Section 2.1 and to Plancherel's report [52] for details about the consensus algorithm for CSS external resources.

4.3 Signing protocol

Once a consensus has been reached and the corresponding webpage reconstructed by the leader, the HTML code and the external resources must be *verified* and *cosigned* by the archiving servers using CoSi [61], presented in Section 2.6.

Every conode agrees to sign only if the data to be signed satisfies a so-called verification function. This function allows the signer to check that the page reconstructed by the leader is the result of a correct consensus protocol. The verification for the HTML code is composed of several steps. If at least one of these step fails, the conode refuses to sign.

- The position of the conode in the tree is verified. The root node must provide an aggregation proof, while leaf nodes must not. Therefore, the verifier must be sure about the position of the conode in the three.
- The cryptographic proofs of all the peer conodes are verified. If at least one of these verification fails, the overall verification of the cryptographic proofs fails.
  - The signature of the encrypted Bloom filter is verified, thereby proving the identity of the conode producing the proofs, because it is the only one holding the private conode key k_c, and verifying that the ciphertext has not been tempered with (recall that ElGamal is malleable).
  - The content of the encrypted classical Bloom filter is verified. For every encrypted value of the filter, the node verifies that the plaintext value is either 0 or 1 using the content proof, as explained in Section 2.4.

  - Only for the root, the aggregation proof is verified. To be sure that all the inputs from the children are taken into account, the verifier first checks that its filter is correclty in the list of the contributions. Then the sum of all the children inputs together with the root's contribution is compared to the final consensus filter provided by the leader, i.e. the aggregation.

- The unique leaves of the consensus HTML tree are compared against the unique leaves of the local tree, resulting from the download of the webpage by the conode. If the local set of leaves is a subset of the consensus set of leaves, the node passes to the next step, otherwise he refuses to sign.

- The unique leaves of the consensus HTML tree are compared against the final consensus spectral Bloom filter, by checking that the multiplicity of every leave in filter is at least equal to the threshold t_d.

- The work of the leader in reconstructing the final consensus filter is audited. The following verifications are performed.

  - Check that the aggregation of the leader corresponds to the encrypted filter sent by the root for the top-down phase of the consensus protocol, i.e. the decryption phase. Note that this verification is needed even if the aggregation proof has already been checked. Indeed, the root could provide a correct verification proof, but it could tamper with the consensus result before sending it to the children for the top-down phase.
  - Take all the partial decryptions of the final encrypted filter and check that the reconstruction has been performed correctly.

For DecenArch we use a *threshold policy* for the collective signing, namely if at least the given threshold number of conodes t_d in the roster correctly sign the data, the signature will be considered valid during verification.

The current implementation does not include a verification function for unstructured data, which are therefore signed without any proof of correctness. This lack of verification has a number of reasons explained in Section 8.7.

Once the consensus page and its additional data are passed through the signing protocol, we can add them to the skipchain. The webpage and its additional resources are sent to the skipchain roster, which compresses everything using with Gzip [53] and adds the data on a new block of the ledger. We stress that HTML code, CSS files and images are *stored on a single block*. Here terminates the process for saving a webpage inside DecenArch.

4.4 Retrieval protocol

The retrieval protocol aims at providing a tool to retrieve a webpage stored on the immutable distributed ledger at a given time. The protocol is initiated by the user, which submits a URL and a timestamp to DecenArch using the provided API. If the timestamp is left blank, the current time is used. Upon receiving the URL and the timestamp, the DecenArch roster forwards them to the skipchain roster. The servers handling the skipchain look for the right block, by iterating from the latest block down to the genesis, in the worst case. For each block the data are first decompressed, then the URL and the timestamp contained in the block are compared with the URL and timestamp submitted by the user. If the URL matches and if the provided timestamp is after the stored timestamp, the page and all the additional data, which are stored on the same block, are returned to the DecenArch roster. Note that by default the most recent stored version of the page is returned, since the current time is used if the timestamp is not provided.

Once the DecenArch roster receives the data and the signatures, the latter are verified and if they are valid, the retrieved webpage is forwarded to the client. Recall that we use a threshold policy for the collective signature of the stored data, meaning that at least a threshold number of conodes are required to correctly cosign.

The DecenArch client receives the data and stores them in the user file system. Finally, the client outputs the local path of the requested page to the user.

5 Security and privacy analysis

In this section, we informally analyze the security and privacy requirements of DecenArch against the threat model introduced in Section 3.2. In particular, we show that an attacker cannot accomplish any of the attacks introduced in the same section. We thereby assume that the adversary is computationally bounded, all the cryptographic primitives used are secure, the private keys are correctly handled and the Byzantine assumption about honest and malicious servers introduced in Section 3.2 are fulfilled.

5.1 Privacy for conodes

As said before, one successful attack is to provide evidence that a conode has seen a given HTML leaf by looking at the consensus material produced by the victim conode or other conodes. If a successful implementation of this attack is possible, DecenArch does not meet one of the security requirements, i.e. privacy for the conodes.

As mentioned in Section 4, every bit of the classical Bloom filter locally produced by a conode is encrypted using ElGamal. Since, as explained in Section 2.2, ElGamal is IND-CPA secure it is impossible for an attacker to tell if a given entry in the Bloom filter contains a 0 or a 1. Therefore, it is impossible to tell whether a conode has inserted a specific leaf in his encrypted filter. One may argue that the Bloom filter can also de decrypted, but since we use a threshold cryptosystem the collaboration of at least a threshold number of nodes is necessary to successfully decrypt a ciphertext. By assumption a threshold number of conodes are, thus we can conclude that the only decrypted ciphertext is the spectral Bloom filter resulting from the consensus algorithm.

Another critical point is the content proof, which prove the content of the filter produced by each conode. As explained in Section 2.4, this proof is zero-knowledge [35], meaning that nothing is revealed about the value being proved. This ensures that also the proof does not leak anything about which content the conode included in the filter.

An additional discussion is needed for the spectral Bloom filter computed by the root, which is the result of the consensus algorithm. Indeed, this filter will be jointly decrypted by the conodes in order to reconstruct the HTML consensus page. In this case, the homomorphic addition of all the contributions decouples any relation between the participants and the content of their local filters. So it is impossible gain informations about the root's view by looking at the output of the consensus protocol. Note that the only way to recover a conode's contribution from the result of the consensus protocol is to subtract all except one contribution. But, since the honest nodes do not disclose their unencrypted filters, this attack is infeasible.

We conclude that DecenArch protect the *privacy* of the cothority servers participating in the consensus protocol.

5.2 Consensus correctness

The most critical part of DecenArch is the consensus algorithm, which should ensure correctness even in presence of a limited number of malicious and colluding conodes. We show in this section that is impossible for an attacker, or a group of attackers, to implant and/or remove one or more leaves in the consensus HTML page that would have been present, respectively not present, in the consensus reached by an ideal roster.

5.2.1 Censorship impossibility

We show in this section that censoring one or more HTML nodes is impossible for a single conode or a group of conodes. We first analyze the case of a single malicious server and then the case of a colluding group of malicious conodes.

Imagine a single malicious node, the goal of which is to censor the content of the webpage. We show that it is impossible for the node to achieve its goal by analyzing two different cases, namely the conode is a leaf or the root of the three.

Leaf The only thing a leaf can do to censor an HTML node is to avoid inserting the node into its filter. The leaf can also send a completely empty Bloom filter to try to censor everything. Since the leaf should provide a proof about the content of the Bloom filter, no values different from 1 and 0 can be inserted into the filter. In particular, no negative values can be inserted into the filter. During the reconstruction phase, HTML nodes appearing at least a threshold number of times in the spectral Bloom filter are considered, therefore we conclude that a single leaf cannot remove any content from the final consensus result.

Root We analyze here to role of a malicious root node. As before, the goal of the root is to censor some content in the final consensus view. The most trivial attack is to simply add content to the consensus webpage proposed to the children for the signing phase. This attack however fails, because the list of unique leaves of the proposed HTML document is not a superset of the list of leaves locally stored by each conode. Thanks to the verification function in the signing procedure, the honest conodes detect this error and refuse to sign. Since the signature uses a threshold policy, the roster does not sign the webpage proposed by the malicious leader and nothing is stored on the skipchain. The malicious leader can also produce an empty or partial filter, but as shown for the malicious leaf case this is not effective. Another plan of attack for the root is to modify the contributions of the children. However, modification, e.g. by exploiting the malleability of ElGamal, is detected during verification since every encrypted filter is signed using the Schnorr signature algorithm [57]. Another possible attack is to ignore the children contributions. To ignore a contribution, the root do not add it to the consensus filter and remove it from the aggregation proof. However, also this attack is detected during verification. Indeed, as presented in Section 4, every leaf node checks that the aggregation proof contains its correct encrypted filter. Therefore, it is impossible for the root to ignore the contributions of the children without being caught. If the root trivially produce a tailored filter for the decryption part while having a correct aggregation proof, the attack is detected because the ciphertext proposed for the decryption phase is compared against the aggregate result contained in the aggregation proof. We have shown that it is impossible for the root to ignore or tamper with the children's contribution. Moreover, the attack techniques presented for a leaf node are ineffective also for the root conode. We conclude that the root alone cannot censor the consensus result.

We move now to the more interesting case of several malicious conodes colluding to censor the result of the consensus protocol. Imagine a number $m_d < \frac{1}{3}n_d$, where n_d is the size of the DecenArch roster, of malicious conode collude. If these conodes are not connected in the three, the multiple malicious nodes case boils down to the single malicious node. We therefore introduce the concept of *malicious path*, indicating that the corrupted nodes are connected in the tree network topology used for the consensus protocol. Suppose that the root and at least one child are malicious, i.e. there is a malicious path of colluding nodes, the aim of which is to censor the output of the consensus protocol. The attacks presented above are ineffective also in the multiple malicious servers case. In order to censor the content the colluding servers must tamper with the proofs of the resulting algorithm. The malicious leaf produces a correct filter and all the corresponding proofs, but sends also a corrupted Bloom filter, e.g. containing negative integers, to the malicious leader. Since TLS can be used for the communication inside the DecenArch roster, we cannot control what is exchanged between the compromised conodes. Once the consensus material reach the root, the root itself uses the corrupted filter to produce the final spectral Bloom filter together, while the correct one is used for all the necessary proofs. Suppose that the malicious nodes sign the result, because they know it has been tempered with. The honest conodes will check all the cryptographic proofs, which are valid by construction. However, cryptographic proofs are not the only verification means used by honest conodes. As presented in Section 4.3, the unique leaves of the consensus HTML tree are compared against the unique leaves of the local HTML tree. If the local set of leaves is a subset of the consensus set of leaves, the conode accept to sign, otherwise it refuses. Suppose now that some HTML node is present in the local tree of all the honest cothority servers, but it has been censored from the consensus view by the malicious conodes. All honest codes refuse in this case to sign, because the HTML node is missing from the consensus proposal. Since we use a threshold policy for the signing protocol, the signature will be refused by the DecenArch roster.

We have shown that even for a group of colluding conodes organized in a malicious path is impossible to censor content during the consensus protocol. The above analysis allows us to conclude that DecenArch achieve *censorship resistance*. We move now to the specular problem, namely implanting content in the output of the consensus phase.

5.2.2 Implantation impossibility

We show in this section that implanting one or more HTML nodes is impossible for a single conode or a group of conodes. As before, we first analyze the single compromised conode case and then the case of a colluding group of conodes.

Imagine a single malicious node, whose goal is to implant content in the consensus result. We analyze again two cases, namely the conode is a leaf in the network topology or the conode is the root of the topology.

Leaf The only strategy for a leaf to implant some HTML content in the consensus result is to produce a spectral Bloom filter containing the threshold value in the buckets to which the HTML content is hashed. This attack strategy is however immediately detected by the leader, because the content proof is verified with the received encrypted filter.

Root We analyze now the role of a malicious leader. The most trivial attacks is to simply add some content to the HTML code proposed to the children for the signing phase. This trivial attack fails, because each child checks if every leaf of HTML tree is indeed in the consensus Bloom filter. If it is not the case, the conode refuses to sign. As explained before, modifying the children contributions, e.g. by multiplying every encrypted counter by a scalar, is not an option, since all the encrypted vectors are signed. Another possible attack strategy for the root is to produce two filters, one containing only 0s and 1s and the other containing corrupted values. The correct filter is used for the proof material, while the malicious one is injected into the consensus spectral filter. This behaviour is not detected by the verification of the cryptographic proof, since the classical Bloom filter for the verification of the content proof is set directly by the malicious leader, because he does not have a parent. Nevertheless, the malicious behaviour is detected when the (honest) conodes audit the work of the leader. As presented in Section 4, one of the step of the audit consists in verifying that the sum of the contributions in the verification proof corresponds with the encrypted filter sent to the nodes for the decryption phase. If the malicious root adopt the two filters attack, the check of the children fails and therefore they refuse to sign the consensus result. We conclude that the leader alone cannot implant content in the consensus result.

We move now to the more interesting case of several malicious conodes colluding to implant content in the result of the consensus protocol. Imagine a number $m_d < \frac{1}{3}n_d$ of malicious conode collude. If these conodes are not connected in the three, the multiple malicious nodes boils down to the single malicious node. Imagine that the root and at least on child are malicious, i.e. there is a malicious path of colluding nodes, the aim of which is to implant content in the output of the consensus algorithm. The only solution is again to try to tamper with the proofs of the consensus protocol. The malicious leaf produces two filters, a correct and a malicious one, e.g. by setting every counter to the threshold t_d. As regards the leader, he creates a correct filter for itself and uses the leaf's correct one for the proof, while

the incorrect filter is used for the aggregation of the contributions. This attack is not detected by the cryptographic proofs, since they are all valid by construction. However, the audit on the work of the leader detects that the proposed final consensus filter is not the result of the sum of all the aggregations and therefore the (honest) nodes refuses to sign. We have shown that even a group of colluding conodes organized in a malicious path is unable to implant content in the result of the consensus algorithm.

To conclude, the analysis of this section shows that *correctness* is guaranteed by the consensus algorithm of DecenArch even in presence of a bounded number of malicious participants.

5.3 Integrity of archived data

One of the goal of DecenArch is to fight censorship, by allowing everyone to permanently store and access any webpage. It is however essential to ensure that a stored webpage, together with its additional data, could not by modified. Since DecenArch uses a skipchain [47] to store the webpage, the additional data and the signatures associated to them, we can conclude that the integrity of the stored consensus result is guaranteed thanks to the immutable distributed ledger. Note that in order to improve the efficiency of DecenArch, we could store all the data but the signatures on a peer-to-peer system such as BitTorrent. We further discuss this possibility in Section 8.8.

6 Prototype implementation

We implemented DecenArch in Go and made it publicly available, along with the instructions on how to install and run it, on GitHub:

`https://github.com/dedis/student_18_decenar`.

For the implementation we used existing open-source code, mainly developed by the DEDIS lab at EPFL. DecenArch is built upon the collective authority (cothority) project [21], which provides a framework for development, analysis and deployment of decentralized, distributed and cryptographic protocols [21]. This framework offers a huge list of building blocks, some of which are used by DecenArch: the cothority network library ONet [24], the skipchain [47] implementation [25] and the ftCoSi [61] implementation [22]. Moreover, DecenArch heavily relies on Kyber [23], the advanced crypto library for the Go language always developed and maintained by the team of DEDIS. For cryptography operations, DecenArch uses also the library based on Kyber developed by the authors of UnLynx [31], a decentralized system for privacy-conscious data sharing proposed by the LCA1 lab at EPFL. This library offers a high parallelization for costly

operations, e.g. the homomorphic addition of ElGamal ciphertexts, which greatly increase the efficiency of DecenArch. The Bloom filter library is partially based on the work of Will Fitzgerald [30]. The most important difference between our implementation and Fitzgerald's one is the usage of secure cryptographic hash functions instead of non-cryptographic ones.

Bloom filters' false positive rate is set to $\epsilon = 0.1$, which gives a good trade-off between probability of false positive and computational overhead. DecenArch uses the elliptic curve Ed25519 [9] with 128-bit security. The library implementing Bloom filters uses SHA-256 [59] and BLAKE2b [6], both implemented in the Go's native crypto library. Communication between the different servers and the client relies on TCP with TLS, which is also used to download the webpage if the host supports it.

Even though the implementation is not yet in production quality and especially it has not been verified and audited yet, it is usable for experiments, evaluation and future work.

7 Performance evaluation

This section evaluates the performance of DecenArch. The simulations are all realized on two servers 24-core Intel Xeons at 2.5 Ghz with 256 GB of RAM, where using Mininet [2] we run up to 64 nodes with the delay between any two nodes set to 100ms. We evaluate here only the performance of the DecenArch roster, starting from the URL submission until the end of the signature process, since the skipchain handling is done by independent servers and is not the central part of this project. We do not include the time needed for the setup, i.e. DKG and skipchain initialization, because these operations are done only once nd do not impact the goal of our simulations, i.e. an in-depth analysis of the DecenArch's efficiency in archiving a webpage.

All the figures, except the the second one, have a logarithmic scale and the results of these simulations are available on the GitHub repository of the project for further analysis.

7.1 Number of conodes

An important factor for DecenArch's time complexity is the number of conodes in the roster. The simulation is realized using a fixed standardized webpage with 512 unique leaves and without additional data. Note that, as shown by Figure 7, more than 80% of the Alexa top 30000 websites have less than 512 unique leaves. Moreover, we do not include any additional data because the goal of this simulation is to test the efficiency of the performances of DecenArch on HTML documents. The results of the simulation are shown on Figure 6. The tests are performed 5 times for the same number of

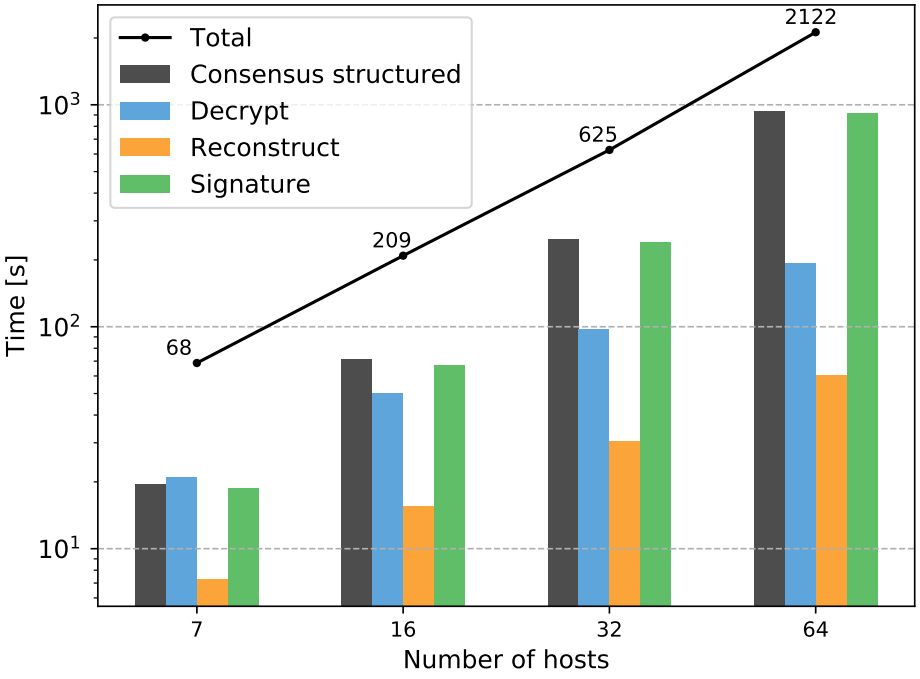

Figure 6: Simulation on a standardized webpage with 512 unique leaves with different number of conodes.

conodes and the avrage value is shown in the graph. The standard deviation is not shown because it is insignificant. From the graph we see that the complexity increases polynomially with the number of hosts. These results confirm also that most of the time is needed for the consensus agreement and the verification, while the decryption of the encrypted consensus filter and the reconstruction of the webpage take a time almost negligible with respect to the overall complexity.

We identify the best trade-off between efficiency and sufficient conodes diversity at 16 servers. We argue indeed that about 4 minutes are an acceptable time to store a webpage, while 16 hosts already guarantee a good security level. To break the security and privacy properties of DecenArch, it is necessary to compromise at least 5 servers. If we ensure enough diversity between the different nodes, as discussed in Section 8.9, successfully attacking at least 5 of them can be considered difficult. For comparison, it is interesting to note that Tor [27] relies on 10 so-called directory servers[5], which are in charge of handling a directory of all the Tor relays. One of

[5]The updated list of directory servers is available on Atlas: `https://metrics.torproject.org/rs.html#search/flag:Authority`. Moreover, the complete list of directory servers is hardcoded in Tor's source code in the file `src/or/auth_dirs.inc`.

these servers (Bifroest) is used for bridge access, so for classical access there are only 9 servers. Since onion routers are included or excluded from the director by majority voting, it is sufficient to control 5 directory servers to include as many compromised onion routers in the final directory as the attacker wishes [27]. And 5 is exactly the number of hosts that needs to be compromised in order to alter the behaviour of DecenArch if 16 cothority servers are used. This comparison with one of the most widely used anonymity solutions confirms that 16 conodes already ensures a good level of security for our system. The discussion on the requirements about conodes and some possible solutions as future work is presented in Section 8. Finally, note that even with 32 hosts, DecenArch takes a bit more than 10 minutes to reach and verify a consensus over the webpage content. This is quite slow, but it almost meets the *10-minute target* set at the beginning of the project.

For the next simulations, where we analyze the impact of the webpage size on DecenArch and we run the system for 7 real-world webpages, 16 conodes are used.

7.2 Size of the webpage

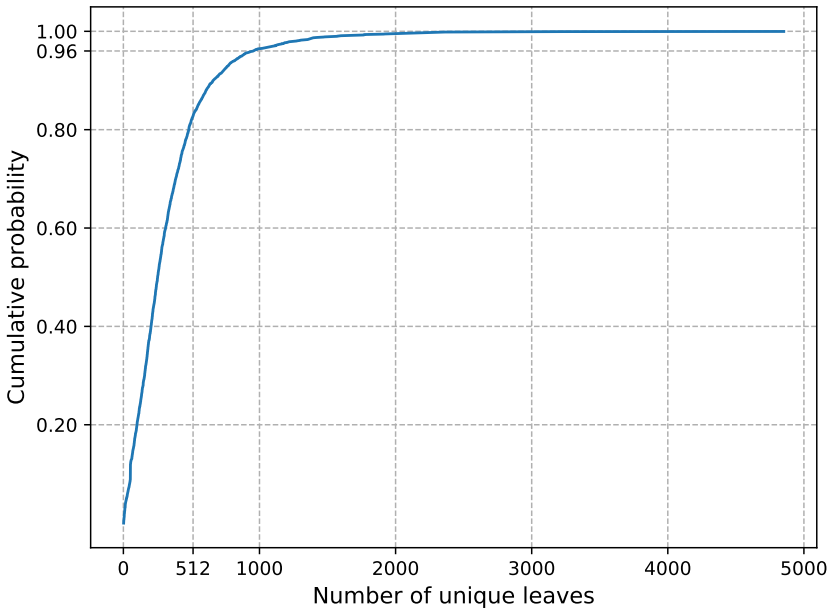

Figure 7: Empirical distribution function of the number of unique leaves in the Alexa top 30000 websites.

This simulation aims at investigating the impact of the webpage size, i.e. the number of unique leaves in the HTML tree, on DecenArch. We first

analyze the Alexa top 30'000 webpages to understand how many unique leaves they have. Three outliers, two of which are Russian pornography webpages, with more than 10'000 unique leaves are removed from results. The empirical distribution function is shown in Figure 7. From the chart we can clearly see that 96% of the websites have at most 1'000 unique leaves and that almost 100% have less than 2'000 unique leaves. Note that 512 leaves have a percentile of 80% and this value is used in the previous section to measure the impact of the number of conodes on the system.

We use standardized webpages created for the purpose of these simulations[6], whose source code is always of the form showed in Figure 8. The

```
<!DOCTYPE html>
<html>
<body>

<p>A leaf 1</p>
<p>A leaf 2</p>

</body>
</html>
```

Figure 8: Standardized HTML code to test webpage's size impact on DecenArch.

different webpages have an increasing number of unique leaves, from 64 to 2048, to be consistent with the results of the analysis on the Alexa top 30'000 webpages. Moreover, the standardized pages do not contain any additional data. Note that here we focus exclusively on the consensus over structured data and over the decrypt, reconstruct and signature protocols. However, it must be said that DecenArch always parses the webpage to look for external links and a longer page will therefore result in a longer parsing time. Nonetheless, the parsing time is negligible compared to the time taken by the different phases presented in the graph. The results of the simulation is shown on Figure 9. The simulation is performed 5 times for the same webpage and the average value is shown in the graph; we do not show the standard deviation since it is insignificant.

Before going into the details of this simulation it is important to understand the relation between the number of unique leaves and the size of the Bloom filter. Indeed, once the HTML tree has been downloaded by a node and the unique leaves are added to the filter, the system only works with the latter. The HTML tree is used again only during the verification function in the signing phase. Therefore, the complexity of DecenArch

[6]The HTML code for the webpages can be found here: `https://github.com/dedis/student_18_decenar/tree/master/simulation/test_input/leaves`.

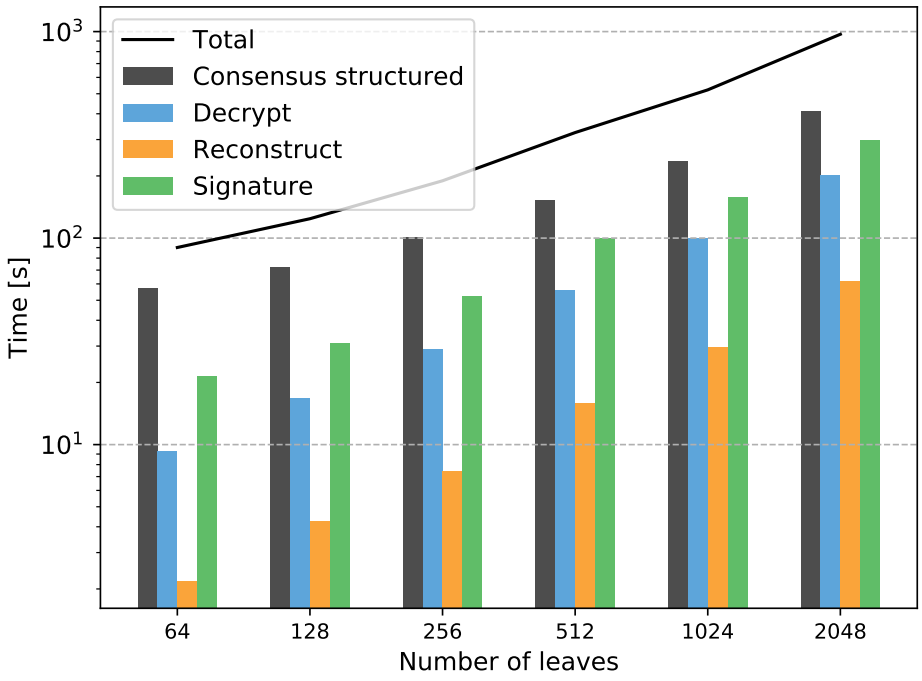

Figure 9: Simulation on 16 conodes with different standardized websites without any additional data.

greatly depends on the size of the filter. Recall that given l leaves and a false positive rate $\epsilon = 0.01$, the size of the filter is given by:

$$m = -\frac{l \ln \epsilon}{(\ln 2)^2} \approx 2.08 \cdot l \cdot \ln \epsilon \approx 9.59 \cdot l$$

Therefore the size of the filter is about 10 times the number of leaves. This is relevant in particular if we consider that all the cryptographic operations, like homomorphic addition or zero knowledge proof creation, are performed on every single bucket of the filter.

From Figure 9 we see that the main components of DecenArch are the consensus and signing protocols. The costly part of the signing process is the verification function, which allows every node to verify the consensus result. The predominance of these two phases on the total time taken by the system is an expected result. During both protocols the nodes should execute a lot of expensive cryptographic operations, such as encrypt the local Bloom filter with ElGamal, create and verify the content proof for the encrypted filter and the root adds all the children contributions with its own local filter, using the additive homomorphic property of the used cryptosystem. Due to this complexity, it is very difficult to compute a precise bound for the different phases of DecenArch. As often during practical implementations, the

constants normally ignored in a theoretical analysis play a crucial role in the overall complexity of the system. Nevertheless, the empirical analysis clearly shows that the time complexity increases polynomially with the number of leaves. Even if a polynomial increase is not very good, we see that more than 80% of the webpages, i.e. up to 512 leaves, are stored by the system composed of 16 conodes in about 10 minutes. This is acceptable for an archive like DecenArch. However, this polynomial increase limits the *throughput* of the system. We discuss this important issue further in Section 8.2.

7.3 Real-life webpages

This simulation aims at evaluating the time taken by DecenArch to reach a consensus over 7 selected real-life webpages. The setup is the same as the second simulation, namely 16 servers compose the DecenArch roster, with a delay between any two of them set to 100ms. Figure 10 presents the

ID	Webpage	Unique leaves
1	`http://www.bbc.com/news/world-europe-44313905`	449
2	`http://www.20min.ch/ro/`	863
3	`http://www.20min.ch/ro/news/monde/story/L-UE-repliquera-de-fa-on--unie--aux-sanctions-US-27701771`	432
4	`https://www.internazionale.it/bloc-notes/2018/05/31/governo-italiano-fantasma-stampa-straniera`	332
5	`https://www.theguardian.com/cities/2018/may/09/fuck-off-google-the-berlin-neighbourhood-fighting-off-a-tech-giant-kreuzberg`	261
6[7]	`https://twitter.com/ignaziocassis`	552
7	`http://tass.com/politics/1007602`	387

Table 3: List of webpages used for the real life simulation. The numbers of unique leaves are also shown for every webpage.

outcome of this simulation, whose values are the result of an average over 5 executions. One important aspect is the presence of additional data, which have a substantial impact for some webpages. For example, the homepage of the 20minutes newspaper (ID 3) has a lot of images and the time needed

[7]Remember when newly elected Federal Councillor Ignazio Cassis cleaned up his Twitter profile from part of the tweets written before the elections? With DecenArch we would have had the possibility to understand what contents the politician preferred to remove. See the NZZ newspaper article for more information: `https://www.nzz.ch/schweiz/bundesrat-cassis-loescht-twitter-nachrichten-aus-zeit-vor-wahl-in-bundesrat-ld.1343424`.

to handle them greatly increase the total time spent by DecenArch on this webpage. The handling of external resources is therefore of fundamental importance for the efficiency of the system. We discuss this further in Section 8.7. Nevertheless, the real-life analysis confirms the critical impact of the consensus for structured data and its verification on the efficiency of the system. This confirms the results of the first two simulations and the difficulty of dealing with malicious adversaries. On a positive note, we can say that the mean of the time needed to reach and verify a consensus over the analyzed webpages is below our 10-minute target, even with 16 conodes, which ensure an acceptable security level.

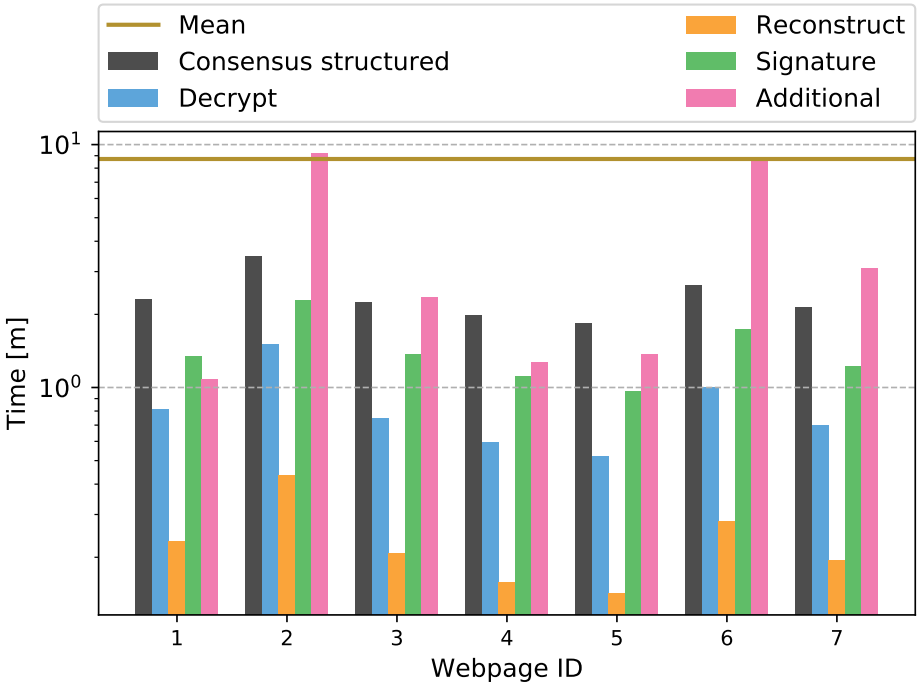

Figure 10: Simulation on 16 conodes using different real-life websites. We stress that in this bar graph the time is expressed in minutes, contrary to the previous one where the second is used as a unit of time measurement. Refer to Table 3 for the mapping between IDs and URLs.

8 Discussion and possible improvements

8.1 Comparison with DIA

We discuss in this section the trade-offs between DecenArch and DIA, the first version of the decentralized internet archive proposed by Plancherel [52]. Recall that DIA is presented and examined in Section 2.1.1.

DIA relies on a trusted leader assumption, without which the system does not offer censorship resistance. The malicious leader can in fact remove some nodes from the reference HTML tree before sending it to the children, which is equivalent to remove them also from the final consensus result. Note that for a malicious leader it is still impossible to implant content in the consensus view and modify the archieved webpages. On the other hand, DecenArch ensures the security and privacy requirements also if the leader is part of the malicious servers, whose number can be at most $\lfloor n_d/3 \rfloor$, where n_d is the total size of the DecenArch roster. Moreover, DIA does not protect the privacy of the conodes. Indeed, every conode simply hashes its view of the webpage before sharing it with the peers. This protection is inefficient, because the content of the webpage has low entropy and it is very easy for an attacker to check whether a conode has some specific content in its view or not. This can be a problem, e.g. for conodes located within an oppressive regime. DecenArch solves this problem using the IND-CPA secure ElGamal cryptosystem.

However, the increased security and privacy guarantees have a cost. In fact, DecenArch has a bigger time complexity compared to DIA. The empirical simulations presented in [52] show that DIA takes less than 40 seconds to reach a consensus on the website, which is about 2 minutes if 16 conodes are used. This means that DIA is 5 times faster than DecenArch on real-life websites. Moreover, the time complexity of Plancherel's system increases linearly with the number of servers, while the increase of DecenArch is a polynomial function of the conodes number. For both systems, the complexity increases polynomially with the size of the webpage. Recall the DIA's consensus algorithm is more complete that the DecenArch one, because it takes into account also the formatting of the HTML document, while our consensus algorithm completely ignores the structure of the webpage. See Section 8.5 for a discussion about the DecenArch consensus algorithm.

To summarize, DIA is much more efficient than DecenArch, but the latter offers higher security and privacy guarantees than the former. The choice between the two systems should be therefore based on the threat model and on the performance requirements.

8.2 Overall time complexity

The big time complexity is without a doubt the greatest limitation of DecenArch. The results of the performance evaluation shows that the complexity increases polynomially both with the size of the webpage and the number of hosts. As mentioned at the beginning of the report, the time taken to store a webpage is not very relevant per se, but it becomes important for the *throughput* of the system. In fact, the goal of DecenArch is to securely store the content for the future, therefore the question is not how

fast we can store a given webpage, but how many webpages we can store in a given time.

From the empirical analysis we see that the consensus and signing protocol have the biggest impact on the complexity of the system. This is expected and the great complexity is the result of the adopted threat model, where we assume the presence of malicious adversaries. The possible improvements on these two phases of the archiving process are however very limited. In order to ensure privacy for the conodes, we absolutely need the costly cryptographic operations of the consensus algorithm. Moreover, to resist against malicious adversaries, all the cryptographic proofs and verification mechanisms are crucial. Dividing the consensus agreement and the verification of its result would be an interesting idea, but it is not applicable. Indeed, in order to execute the verification function, a node must have participated in the consensus and the subsequent decryption protocol. Therefore, the set of conodes taking part in the consensus agreement must be the same one that verifies and signs the webpage.

A possible solution to mitigate the effects of the high time complexity on the throughput of the system, is to create a federation of several DecenArch rosters. The size of one roster is set to 16 conodes, because of the discussion of Section7. The user always interact with a single DecenArch roster, which in turn is responsible for outsourcing the archiving procedure to one of the other rosters. We can imagine to randomly create the group of 16 servers for every request of archival, in order mitigate the impact of an attacker targeting a specific list of servers. The skipchain roster remains the same and the different DecenArch rosters independently communicate with it. The main problem of this approach is the verification of the webpage's signature during the retrieval process. In fact, once the data are retrieved from the ledger, the signature is verified by the archiving servers and only if it is valid, the retrieved webpage is sent to the user. In order to verify the signature, the public key of the singers is needed and therefore the system must know which servers were in charge of archiving the webpage. One solution is to store the list of the servers that participated in saving the webpage on the skipblock, together with the data and the signatures. This solves the problem of knowing the list of signers and make the federation approach applicable to DecenArch.

The federation approach is a general solution to improve the throughput and the usability of DecenArch. In the rest of this section we discuss more specific problems, some of which have an impact also on the time complexity of the system.

8.3 Topology

As presented in Section 4.2, for the consensus protocol the servers are structured to communicate in a 1-level tree topology. This simplifies the security

and privacy analysis and facilitates the implementation, but the efficiency of system is affected. Indeed, the root perform almost all the computations in the system, while it would be better that these were more evenly distributed among the different archiving servers. In fact, the consensus protocol of DecenArch can be adapted to a different topology, e.g. a binary tree. This adaptation requires however an appropriate theoretical analysis, e.g. what happens if an intermediate parent does not forward the contributions of the children, and some implementation efforts. Due to lack of time we stick on the simplest topology. In a future work an analysis of the best servers' communication structure for the consensus algorithm should be in the to-do list.

8.4 The choice of ElGamal

For public-key cryptography DecenArch uses the elliptic curve ElGamal cryptosystem. There are several reasons for choosing it and we discuss them in this paragraph. We also introduce another possible cryptosystem that can be used within our system, by discussing its advantages and disadvantages.

ElGamal was an early choice in the project. For all its products and experiments the DEDIS lab uses cryptosystems based on the elliptic curve discrete logarithm problem and none of the libraries developed by the laboratory uses cryptosystems based on different hard problems, such as integer factorization. Since this master thesis is realized at DEDIS, we choose to work with elliptic curve cryptography. Moreover, working with elliptic curves is more *powerful and efficient*, e.g. keys are smaller, than alternatives like RSA or other systems based on different intractability hypothesis [7]. The choice of ElGamal brings however also some drawbacks. Before encrypting an integer $n \geq 0$, the integer should be mapped to a point $P_n \in E$ using an appropriate mapping function, which increases the time complexity of the encryption process. Moreover, in order to retrieve the plaintext from a ciphertext, we have to solve the discrete logarithm problem. Since we encrypt only small numbers the computations are still doable, but again we increase the complexity of the decryption process. Note that to minimize the impact of finding the discrete logarithm for a given point $P \in E$ we use an efficient data structure to store already computed results.

An alternative to the ElGamal cryptosystem is the Paillier cryptosystem [48], based on the decisional composite residuosity assumption. Note that this assumption is based again on the factoring problem, but an in-depth discussion is outside the scope of this report. We refer the reader to [48] for an exhaustive explanation. The cryptosystem proposed by Paillier is by construction additively and multiplicatively homomorphic and is therefore suitable for DecenArch. Moreover, an integer $0 \leq i < N$ is directly encrypted without need of a mapping function, where $N = pq$ is the public key and p, q two equal length large primes selected uniformly at random by

```
<!DOCTYPE html>
<html>
<body>

<p>Alice</p>
<p>insulted</p>
<p>Bob</p>

</body>
</html>
```

(a) Original webpage.

```
<!DOCTYPE html>
<html>
<body>

<p>Bob</p>
<p>insulted</p>
<p>Alice</p>

</body>
</html>
```

(b) Modified webpage.

Figure 11: Two simple HTML pages for which the formatting matters.

the key generation algorithm. Finally, threshold Paillier cryptosystems exist [19, 38]. The disadvantage of this cryptosystem is its inefficiency, both in terms of space and time. Nevertheless, it would be interesting to implement and evaluate a version of DecenArch using the Paillier cryptosystem.

8.5 The consensus algorithm for structured data

The consensus algorithm is the most important part of DecenArch. Its design and implementation have taken up most of the time dedicated to this project. The difficulties in designing the consensus algorithm arise from its constraints. The algorithm should implement a *threshold set-union* over the content of the webpage and the result must be *correct* even in presence of a limited number of *malicious adversaries*. Moreover, the *privacy* for the participating parties is a requisite, meaning that no information except the content of the union is disclosed. Finally, the consensus protocol and the entire saving process should not take too much time. The goal at the beginning of the project was below 10 minutes for a real-life webpage and the empirical results presented in the previous session show that DecenArch achieves this objective. In order to meet all the requirements presented above, we have to take some decisions. The first, and perhaps most questionable, decision is to completely ignore the order of the content of the webpage. As explained, the consensus algorithm works only on the leaves of the HTML tree, therefore the formatting of the webpage is not taken into account. This decision must be taken in order to make the consensus algorithm usable in practice. Indeed, if the consensus algorithm is fed with the entire HTML code, the Bloom filters used to store the nodes become intractable. In support of our choice, we argue that the content of a webpage is far more important than its formatting, since if the latter changes the former is still visible and present in the stored webpage. However, one can produce an HTML document whose formatting is essential to make sense of

the content, such as the one shown in Figure 11. In this case, if Bob controls the server acting as the leader of the consensus protocol, he can modify the original code shown in Figure 11a into the code shown in Figure 11b without being detected. We believe that such webpages are very rare on the Internet, but nonetheless the problem of an efficient consensus algorithm taking into account also the formatting is an open problem that should be explored in future work.

Related to the choice of working only with the HTML leaves, is the choice to keep only the unique leaves. A leave is considered equal to another one if the data, i.e. the string, contained in both of them are equal. This choice is also due to efficiency constraints and the observation that often when multiple identical leaves appear in an HTML tree, they are also used for formatting. It is common for example to find multiple times a leave containing `\n\n\n`. Again, this choice is a trade-off between efficiency and completeness of the consensus agreement and, even if this choice as a lower impact than ignoring the parent nodes of the HTML tree, some work is needed to avoid possible attacks.

The usage of Bloom filters is another critical point of the consensus algorithm. The motivation behind the choice of using this data structure is twofold. On one hand we need to represent the set of unique leaves in a way that is suitable for additively homomorphic computations, and (spectral) Bloom filters meet this need, on the other hand efficiency constraints force us to represent the potentially very big set of unique leaves in an efficient and practical way, and again Bloom filters meet this requirement. The problem of using a probabilistic data structure is however the error probability. Bloom filters have a constant probability of false positive, which is set to 1% for DecenArch. The presence of false positives could allow a malicious node to insert an HTML leaf in the consensus result, if the hashes of the leaf point to counters with values higher than the threshold. This risk is partially mitigated by our filters implementation, which uses cryptographic secure hash functions. A malicious node therefore cannot artificially build an HTML leaf that will be hashed to specified locations of the filter, because the used hash functions are resistant to first pre-image attack [63]. The only way to exploit a non-zero false positive rate is therefore to test all the possible leafs whose hashes give the same locations as the legitimate leaves.

8.6 The verification function for structured data

The verification function for the HTML document used in the signing protocol (see Section 4.3) is quite involved and ensures, using different proofs, that the archiving servers do not sign an incorrect consensus result. However, the actual implementation of the verification function does not give enough room for manoeuvre to the conodes. Recall that the consensus HTML tree is re-

constructed from the consensus filter by the leader of the protocol, which performs a membership test for all the unique leaves of its own local HTML tree. This implies that the leader can remove any content from the consensus HTML content sent to the children for the signing process. The children detect that something is missing from the consensus result, thanks to the subset check, and refuses therefore to sign the proposal. The dispute resolution is however very rigid, since the only options for a node are to sign or to refuse to sign. We discuss here two possible improvements, namely the *change of leader* and the *proposal of insertion* of verified HTML content.

The change of leader is a very simple concept: if the current leader is behaving incorrectly or it is censored, its role is randomly assigned to another node of the roster. To detect if the leader is malicious, we should modify the signing protocol as follows. Right now the choice of signing or not a message is communicated by the node only to the root, while other servers in the tree are not aware of the choice. If a node broadcast it's choice to everyone, anyone in the roster can keep track of the number of refusals and if a given threshold is exceeded, requests a change of leader. This solution gives the nodes more degree of freedom about the signature of the consensus material, but there is still the *censored leader* issue. Suppose that the sever hosting the leader of the protocol is located in a country blocking some specific content of a webpage. Even if the blocked content should be present in the final consensus view, because it has been downloaded by a threshold number of conodes and the leader is honest, it is not included in the leader's proposal for the signing protocol. This happens because, as said before, the consensus tree is reconstructed by the leader using its own local HTML tree, which does not contain the censored content. With the proposal of insertion we try to mitigate this censorship risk and solve the censored leader problem.

When a conode detects that the list of the unique leaves resulting from the consensus protocol is not a subset of the local list of leaves, it proposes a content insertion. To do so, the conode sends the content to be inserted together with a proof that the content is present in the conode's view of the page. One solution to provide the proof is to use TLS-N [55], a TLS extension that generates non-interactive cryptography proofs about the content of a TLS session, which can be efficiently verified by third parties. TLS-N supports different proof types and the sensitive content of the TLS session can be partially protected using privacy protection solutions. The limitation of TLS-N is that the server hosting the webpage should support it and, since the extension proposal is recent, for the moment it has been adopted only by a few servers devoted to research. Another possible source of trusted content is Town Crier (TC) (see Section 9). In this case a conode to propose the insertion of some HTML content broadcast to all other nodes the result given by TC and the query used receive the result. Any other node can then question TC with the same query and verify the insertion proposal.

The implementation of the two improvements presented above can be

quite involved and adds a lot of complexity to the signing protocol, both in term of time and communication overhead. We however believe that future work should explore these and other possible ways to relax the rigidity of the verification function and solve the censored leader problem.

8.7 Management of additional data

The consensus for additional data used by DecenArch is the same as the one in DIA and suffers therefore from the problem highlighted in Section 2.1. In particular, with the current consensus algorithm it is very easy for a malicious conode to detect if another conode proposes a specific external resource for the consensus agreement. It is sufficient to verify if the `MasterHash` of the given external resource contains the signature of the victim conode. This very critical point should be addressed in a future work, to develop a consensus algorithm for non structured data providing privacy for the involved parties.

A possible approach is to use Bloom filters also for images and CSS files. Every node would then hashes all the external resources into a Bloom filter and the consensus algorithm would work exactly as the one for the main HTML code of the page. In fact, every type of file can be inserted into a Bloom filter thanks to the hashing procedure. This prototype version of DecenArch aims at being a proof of concept of the usage of Bloom filters for secure and privacy preserving consensus agreement over webpages. Due to lack of time we do not implemented the consensus algorithm based on filter for additional data, but if the algorithm based on encrypted Bloom filters is considered interesting, thanks to all the libraries developed during the project it should not be difficult to improve the handling of additional data in this sense.

The usage of Bloom filters also for the additional data's consensus algorithm can solve also another problem related to them, namely the verification function. For the prototype version of DecenArch, the verification function for additional data is not implemented, since we mainly focused on the HTML document of the webpage in this work. Nevertheless, it is important to verify also the consensus agreement on external resources, because these are part of the webpage's content we want to correctly and securely store. If we stick with the consensus protocol proposed by Plancherel, a possible way to verify the correctness of the consensus result is to compare the hash of each external resource in the consensus webpage with the hash of the same resources locally stored by each conode. This is very inefficient from a storage point of view, because each server should keep a local copy of all the additional data, but it is the only way to verify correctness if we use Plancherel's consensus algorithm for additional data. On the other hand, if we decide to use the algorithm for the HTML code also for the external resources, we can use the verification function presented in Section 4,

thereby increasing the security of the whole system and reducing the code complexity.

Another critical point related to additional data is the choice to consider only CSS files and images, while ignoring everything else. We argue that this choice is a good trade-off between content preservation and storage efficiency. It is however true that other files, e.g. videos, could be crucial to correctly store the content of a webpage. Moreover, these different file types could be victim of censorship and it would be useful to securely store them. When discussing what do store and what to ignore in a webpage there is no correct or incorrect choice, but only subjective definition of the importance of a given content and trade-offs evaluation. A better storage efficiency could allow DecenArch to store also other kind of files. We discuss this in the next section.

8.8 The storage efficiency

Storage efficiency is not considered a critical point of this project, but it deserves nevertheless some discussion. In DecenArch, the webpage resulting from the consensus protocol and its additional data are first compressed with the lossless compression algorithm Gzip[8] [53] and then stored on a single block of the skipchain. Also, the lookup of a stored page is done directly on the skipchain, by looking for the right block from the most recent one to the genesis. This is the simplest solution to implement and it is adapted for a prototype implementation, but if we want to use DecenArch on a real-world scale we definitely need to improve this aspect. In the following we propose some solutions to improve the storage efficiency of DecenArch without compromising its security and privacy properties.

The first and most obvious solution is to separate the actual data from their signatures. The former should be stored on a peer-to-peer system (e.g. BitTorrent) and the former on the immutable ledger. Indeed, the immutability property is needed only for the signatures, whose goal is to verify the integrity of the data. On the other hand, the data can be stored everywhere, since thanks to the signatures we can verify their validity. With this configuration the retrieval protocol is modified. The client first query a distributed hash table (DHT) [5] for the content of a webpage at a given time and then perform a lookup of the signatures in the skipchain. The retrieved content is then verified using the signatures stored in the skipblock. Data storage should also rely on a decentralized system and it should not be possible to censure it. If we do not trust any peer-to-peer system we can keep a copy of the data on the skipchain, but at the same time distributing the data on the untrusted system to improve the efficiency of webpages retrieval.

[8] We choose Gzip because it is the algorithm used in HTTP compression. In the scope of a future work, it could be useful to perform a comparative analysis of different compression algorithms to find the most suitable for DecenArch.

The next solution, proposed also by Plancherel [52] is to use an incremental storage approach. With the actual implementation if a URL is submitted multiple times, the corresponding webpage together with its external resources is entirely stored in the skipchain, without exploiting the similarity between the different versions of the page. Instead of always storing all the data, we could save the entire webpage only the first time and for the subsequent requests we store only the differences between the most recent stored version and the current version.

8.9 The requirements about conodes

The requirements about the cothority servers of the DecenArch roster have already been analyzed in [52], but we report and extend the discussion here since the issue is very critical. Note that this discussion is related to the idea of having a federation of DecenArch rosters presented in Section 8.2.

In order to achieve the security and privacy goals it is essential to have a sufficient number of conodes. It is very difficult to estimate the minimal number of conodes needed by DecenArch, but we can nevertheless list some important properties all the server, the ones used for all the protocols and the ones handling the skipchain, should fulfill in order to strength DecenArch. First, the servers should be geographically distributed and they must be subjected to different, and possibly correct, laws. If possible, an important part of the servers should be hosted in countries where freedom of press is (almost) guaranteed, such as Switzerland or Canada [54]. This is the only way to avoid censorship imposed by state-level adversaries, which may block sensitive content directly on the server hosting the target webpage or may impose some restrictions to the cothority servers used for DecenArch. Moreover, having servers all over the world makes a coordinated attack against them more difficult to carry out. We could imagine imposing these geographical and legislative requirements directly in the protocol, by ensuring that the conodes are located in different countries before archiving a webpage. Cothority servers for DecenArch should moreover be hosted by different and independent entities, like universities, research centers, private actors and individuals. This ensures an important diversity in the rosters, increases the protection against state-level adversaries, e.g. it is more complicated to censor a university, and helps to avoid the risk of intentional or even unintentional collusion among the cothority servers. Finally, it is crucial that the DecenArch software is always up-to-dated and verified. To ensure the authenticity and integrity of the DecenArch binary and its updates we could use Chainiac, a software-update framework proposed by Nikitin et al. [47]. To summarize, the conodes of the DecenArch roster and of the roster handling the skipchain should be geographically dispersed, subjected to different laws, operated by different and independent entities and regularly and securely updated.

8.10 The right to be forgotten

We all know it: the Internet never forget, but forgetting *could be* technically possible. On the other hand, it is by definition impossible to modify and/or remove a block from a skipchain. This is actually the main point of using an immutable distributed ledger to store the outcome of our consensus protocol, i.e. the skipchain ensures the integrity of our stored webpages. While this is a desirable property, it clashes with the *right to be forgotten*. The concept of right to be forgotten arises from the fundamental need of a person to control the development of his personal life independently and self-determinately, without being stigmatized and excluded as a consequence of a specified and limited action taken in the past, especially if these events occurred several years ago and do not have any consequence over the present-day situation [44]. The EU General Data Protection Regulation (GDPR) [28], which will be enforceable from 25 May 2018, contains an article about the right to be forgotten, which states that if one of the different grounds listed applies, the data subject shall have the right to obtain from the controller, i.e. the person or the entity controlling the data, the deletion of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay [28]. It is still unclear how distributed and decentralized system and blockchains will be impacted by the GDPR, since for the former no one really controls the data, while from the latter it is inherently impossible to alter or remove of data. We are also keen to say that the concept of right to be forgotten has generated an intense debate. Some researches and journalists accuse the right to be forgotten of restricting freedom of speech [34]. This is partly confirmed by the documents released by Google [37], which shows that European politicians have used the right to be forgotten to delete an incredible amount of articles about themselves from search results in Europe [45]. The practical and technical problems of putting into practice the right to be forgotten is also often a cause for discussion. On the other hand, the incredible ease with which we can access a huge quantity of documents and information poses serious problems for the life development of the people. An in-depth discussion about the impact of the GDPR or about the ethical and philosophical implications of the right to be forgotten are out of the scope of this report, but these two issues deserve attention and analysis in a future work.

9 Bonus: provide trusted data to smart contracts

One of the biggest weaknesses of decentralized smart contracts is their inability to reach outside the closed environment of the blockchain to retrieve informations. Smart contracts cannot, for example, query a web server for a given webpage. Therefore, any outside information has to be provided

to the chain from an outside entity, called oracle. The problem is that the source of the needed information has to be trusted, since the correctness of the provided data cannot be verified by the chain. It is here that an important contradiction is reached: we want to use a decentralized system for our contracts, but at the same time we have to trust a (centralized) oracle for the data used by the contract itself.

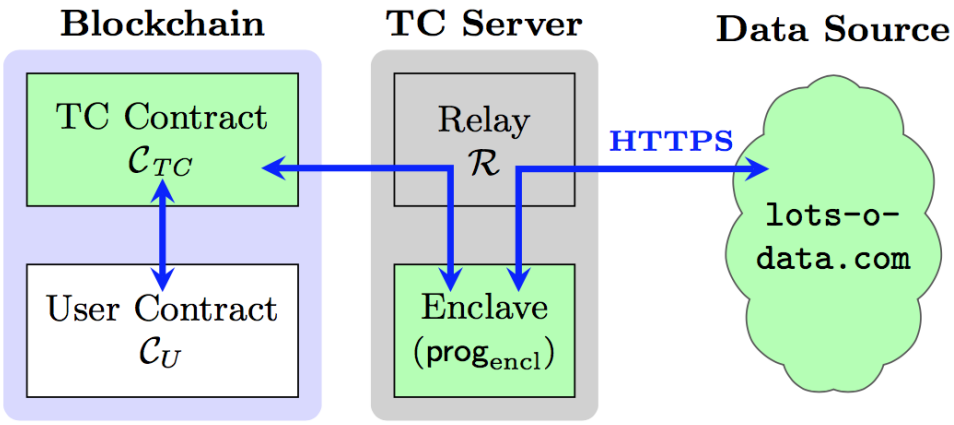

Figure 12: Overview of Town Crier. Source: `https://town-crier.readthedocs.io`.

One solution to this shortcoming is Town Crier (TC) [64], whose main objective is exactly to serve as a bridge between existing trust at sources of data, e.g. a webpage, and smart contracts. The big picture of TC is shown in Figure 12. TC relies on *trusted hardware*, namely the Intel SGX [17]. Discussing SGX or the trust placed in Intel is out of the scope of this report, but we think it is important to discuss the main limitation of TC: it is in fact a *centralized system*. As shown in Figure 12, TC relies on a central server to retrieve the data from the source and to provide them first to the TC contract, which is a smart contract deployed on the blockchain responsible to interface with the user contract, and then to the contract defined by the user. Centralization means that if the TC server is compromised, e.g. an attack against SGX is exploited, we cannot trust the data anymore. And not trusting the data implies that the result of the contract cannot be trusted.

The idea is to use the DecenArch roster and its consensus protocol instead of the TC server to provide trusted webpage's content to a smart contract. The overview of this alternative usage of DecenArch is shown in Figure 13, where we use the same figures as the TC image to highlight the similarities and differencies of the two approaches. Note that in this case we don't need to store the data on a blockchain, but simply to provide the verifiable consensus result to the user contract, using the DecenArch contract as frontend. The advantage of using the DecenArch roster instead

of the TC server is that we have a *decentralized* system providing trusted data to the smart contract. Indeed, in Figure 12 the bridge between the source of data and the blockchain is represented by a server, while our approach, as swhown by Figure 13, uses a group of independent servers. Due to lack of time we do not explore further this idea, but we argue that this is an interesting path to explore in a future work.

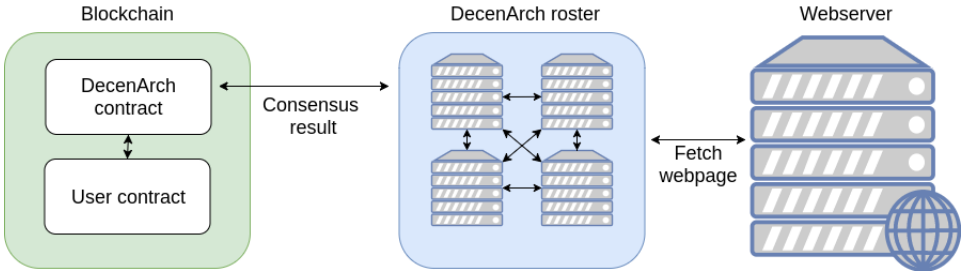

Figure 13: Simulation on 7 conodes with a binary tree topology on different standardized websites without any additional data.

10 Conclusions

In this report, we have presented, evaluated and discussed DecenArch, a decentralized system for privacy-conscious Web archiving providing censorship resistance.

DecenArch uses independent servers to determine the content of a webpage whose URL is submitted by a user through an API. The consensus protocol for the webpage and its external resources is defined as a threshold set-union over the content retrieved by every conode. During and after the execution of the protocol the privacy of the participating parties is protected and the result is correct in the Byzantine attack model [43], where up to $\lfloor \frac{n}{3} \rfloor$ of the n servers are malicious and collude to alter the result or to gain sensitive information about the honest nodes. To ensure the correctness of the result and the privacy for the parties different advanced cryptographic techniques are used. The webpage resulting from the consensus protocol, together with the CSS files and images attached to it, is archived on a skipchain, a distributed immutable ledger which ensures integrity for the saved data and allows anyone to access the stored content, thereby achieving censorship resistance.

Thanks to the analysis of empirical simulations we have seen that the time complexity of DecenArch increases polynomially with the size of the webpage and with the number of servers. This is the most negative result of the entire research project, but when dealing with malicious adversaries the complexity is inevitably destined to rise. However, we see that DecenArch is usable on real-life websites, whose archiving requires about 10

minutes with 16 conodes, which provide a good level of security and privacy guarantees. Therefore, DecenArch meets the censorship resistance goal we had at the beginning of the master thesis.

We also presented a possible alternative usage of DecenArch to provide a trusted source of data for smart contracts. These have in fact an intrinsic limitation, namely their inability to reach outside the closed environment of the blockchain to retrieve data and information needed to run the contract. In this sense, DecenArch could be used as a bridge between trusted sources of data and smart contracts.

Nevertheless, there is plenty of room of improvements, as we discussed in the final part of this report. Different aspects of DecenArch should be further analyzed in future work, e.g. the storage efficiency or the management of the webpage's external resources. DecenArch is a tentative of contributing to mitigate the risks of a centralized Internet, both from a privacy protection point of view and a censorship resistance one. Thanks to decentralization and the incredible power of cryptography we can, and should, build a better, more secure and more stable Internet to mitigate all the security risks arising with the predominant Internet design currently in place.

Acknowledgements

I would like to thank professor Bryan Ford for giving me the chance to work on this project in the laboratory he leads. I would also like to thank Eleftherios Kokoris-Kogias and Kirill Nikitin, my supervisors, and Linus Gasser and Kelong Cong, computer scientists at the DEDIS lab, for their help, patience and support throughout the entire project. I am grateful also to the developer of UnLynx [31] for their cryptography library based on DEDIS's Kyber library [23], which saved me a lot of coding time and greatly improves the efficiency of the entire system. Last but not least, I would like to thank Nicolas Plancherel for his work on the first version of the decentralized Internet archive, which gave me an important and solid starting point from which to develop the current version, DecenArch.

References

[1] Internet censorship in China. `https://en.wikipedia.org/wiki/Internet_censorship_in_China#cite_note-80`. Last accessed: 13-06-2018.

[2] Mininet — An Instant Virtual Network on your Laptop (or other PC). `http://mininet.org/`. Last accessed: 12-06-2018.

[3] The NSA files. https://www.theguardian.com/us-news/the-nsa-files. Last accessed: 14-06-2018.

[4] Websites blocked in mainland China. https://en.wikipedia.org/wiki/Websites_blocked_in_mainland_China. Last accessed: 13-06-2018.

[5] Distributed hash table (DHT). In *Encyclopedia of Parallel Computing*, page 573. 2011.

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, pages 119–135, 2013.

[7] J.P. Aumasson. *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, 2017.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[9] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, page 16, 2012.

[10] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[11] John Bohannon. Who's downloading pirated papers? Everyone. Science, https://www.sciencemag.org/news/2016/04/whos-downloading-pirated-papers-everyone, Apr. 2016. Last accessed: 20-06-2018.

[12] Sissi Cao. Apple Grants China Full Control of iCloud Data — and the Timing Couldn't Be Worse. Observer, http://observer.com/2018/03/apple-grants-china-full-control-icloud-data/, Feb. 2018. Last accessed: 13-06-2018.

[13] Nate Cardozo, Andrew Crocker, Gennie Gebhart, Jennifer Lynch, Kurt Opsahl, and Jillian C. York. Who Has Your Back? Censorship Edition 2018. EFF, https://www.eff.org/who-has-your-back-2018, May 2018. Last accessed: 20-06-2018.

[14] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 89–105, 1992.

[15] Justin Clark, Robert Fairs, Ryan Morrison-Westphal, Helmi Noman, Casey Tilton, and Jonathan Zittrain. The shifting landscape of global internet censorship. *Berkman Klein Center for Internet & Society Research Publication*, 2017.

[16] Saar Cohen and Yossi Matias. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 241–252, New York, NY, USA, 2003. ACM.

[17] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[18] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8(5):481–490, 1997.

[19] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, pages 119–136, 2001.

[20] Ivan Damgård. On Σ-protocols, 2010.

[21] DEDIS. Cothority. `https://github.com/dedis/cothority`, 2018.

[22] DEDIS. FtCoSi - Fault tolerant collective signing. `https://github.com/dedis/ftcosi`, 2018.

[23] DEDIS. Kyber - The advanced crypto library for Go. `https://github.com/dedis/kyber`, 2018.

[24] DEDIS. Onet - The cothority network library. `https://github.com/dedis/onet`, 2018.

[25] DEDIS. Skipchain implementation. `https://github.com/dedis/cothority/skipchain`, 2018.

[26] Yvo Desmedt. Threshold cryptography. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 1288–1293. 2011.

[27] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX*

Security Symposium, August 9-13, 2004, San Diego, CA, USA, pages 303–320, 2004.

[28] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[29] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[30] Will Fitzgerald. Go package implementing Bloom filters. `https://github.com/willf/bloom`, 2017.

[31] David Froelicher, Patricia Egger, João Sá Sousa, Jean Louis Raisaro, Zhicong Huang, Christian Vincent Mouchet, Bryan Ford, and Jean-Pierre Hubaux. Unlynx: A decentralized system for privacy-conscious data sharing. *Proceedings on Privacy Enhancing Technologies*, 4:152–170, 2017.

[32] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[33] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 295–310, 1999.

[34] Dan Gillmor. The 'right to be forgotten' doesn't mean we should be censoring Google results. The Guardian, `https://www.theguardian.com/commentisfree/2014/jun/24/right-to-be-forgotten-censoring-google-results`, Jun. 2014. Last accessed: 11-06-2018.

[35] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[36] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[37] Google. Transparency Report. Google, `https://transparencyreport.google.com`, 2018. Last accessed: 11-06-2018.

[38] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold paillier in the two-party setting. In *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, pages 313–331, 2012.

[39] Mike Isaac. Facebook Mounts Effort to Limit Tide of Fake News. New York Times, `https://www.nytimes.com/2016/12/15/technology/facebook-fake-news.html`, Dec. 2016. Last accessed: 13-06-2018.

[40] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better bloom filter. In *In Proc. the 14th Annual European Symposium on Algorithms (ESA 2006)*, pages 456–467, 2006.

[41] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 279–296, 2016.

[42] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[43] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[44] Alessandro Mantelero. The EU proposal for a general data protection regulation and the roots of the 'right to be forgotten'. *Computer Law & Security Review*, 29(3):229–235, jun 2013.

[45] David Meyer. People have asked Google to remove 2.4 million links about them. here's what they want to forget. Fortune, `http://fortune.com/2018/02/28/google-right-to-be-forgotten-europe-reasons-eu/`, Feb. 2018. Last accessed: 11-06-2018.

[46] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 245–254, New York, NY, USA, 2001. ACM.

[47] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: proactive software-update transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 1271–1287, 2017.

[48] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.

[49] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991.

[50] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, pages 522–526, Berlin, Heidelberg, 1991. Springer-Verlag.

[51] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity — a proposal for terminology. In *Designing Privacy Enhancing Technologies*, pages 1–9. Springer Berlin Heidelberg, 2001.

[52] Nicolas Plancherel. Decentralized internet archive using the cothority framework. Master's thesis, EPFL, 2018.

[53] GNU Project. GNU Gzip. `https://www.gnu.org/software/gzip/`, 2018.

[54] Reporters Without Borders. 2017 World Press Freedom Index, 2017.

[55] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. TLS-N: non-repudiation over TLS enabling - ubiquitous content signing for disintermediation. *IACR Cryptology ePrint Archive*, 2017:578, 2017.

[56] Dominic Rushe and Lauren Gambino. US regulator scraps net neutrality rules that protect open internet. The Guardian, `https://www.theguardian.com/technology/2017/dec/14/net-neutrality-fcc-rules-open-internet`, Dec. 2017. Last accessed: 04-06-2018.

[57] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[58] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 148–164, 1999.

[59] John Bryson Secretary and Charles H. Romine. FIPS PUB 180-4 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Secure Hash Standard (SHS), 2012.

[60] Sarah Lai Striland. CISCO leak: "great firewall" of China was a change to sell more routers. Wired, `https://www.wired.com/2008/05/leaked-cisco-do/`, May 2018. Last accessed: 13-06-2018.

[61] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 526–545, 2016.

[62] Serge Vaudenay. Lectures notes in Advanced Cryptography, 2016.

[63] Serge Vaudenay. Lectures notes in Cryptography and Security, 2016.

[64] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 270–282, New York, NY, USA, 2016. ACM.