Project Title	Trust-aware, Reliable and Distributed Information Security in the Cloud
Project Acronym	TREDISEC
Project No	644412
Instrument	Research and Innovation Action
Thematic Priority	Cybersecurity, Trustworthy ICT
Start Date of Project	01.04.2015
Duration of Project	36 Months
Project Website	www.tredisec.eu

D3.1 - REQUIREMENTS AND TRADE-OFFS BETWEEN VERIFIABILITY AND DATA REDUCTION

Work Package	WP 3, Verifiability with storage efficiency
Lead Author (Org)	Angelo De Caro, Alessandro Sorniotti (IBM)
Contributing Author(s) (Org)	Rosa Vieira, Carlos Hernan (ATOS), Kaoutar Elkhiyaoui, Melek Önen, Dimitrios Vasilopoulos (EURC), Panos Louridas (GRNET), Roch LESCUYER (MORPHO), Ghassan Karame (NEC)
Reviewers	Mathias Kohler (SAP), Ghassan Karame (NEC), Rodrigo Diaz Rodriguez (ATOS)
Due Date	31.03.2016
Date	25.03.2016
Version	Final

Dissemination Level

X	PU: Public
	CO: Confidential, only for members of the consortium (including the Commission)

The work described in this document has been conducted within the project TREDISEC. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 644412. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of its content.

Versioning and contribution history

Version	Date	Author	Notes
1.0	11.12.2015	IBM	Initial Outline
1.1	15.01.2016	EURC, IBM, NEC	Initial contributions to sections 2, 4 and 6
1.2	18.01.2016	MPH	Initial contributions to Section 5
1.3	25.01.2016	IBM	Initial contributions to Section 1
1.4	28.01.2013	ATOS, EURC	Updates on Section related to TPM
1.5	01.02.2016	ATOS, MPH	Initial contributions to Section 3.3. Updates on Section 5.1.4 related to TPM. New summary in Section 5.2
2.0	11.02.2016	IBM	Extended contributions to Section 6. Refinements on Section 2.
2.1	17.02.2016	EURC	Updates on Section 4
2.2	18.02.2016	GRNET	Initial contributions to Section 3.2
2.3	24.02.2016	MPH	Updates on Section 5 and Section 3.3
2.4	24.02.2016	ATOS	Updates on Section 5
2.5	29.02.2016	MPH	Updates on Sections 3.3 and 5.3
3.0	01.03.2016	ATOS, EURC, IBM, MPH	Merged contributions from the partners.
4.0	16.03.2016	ATOS, EURC, IBM, MPH	Review comments applied
5.0	25.03.2016	ATOS, EURC, IBM, MPH	Approval review comments applied
5.1	29.03.2016	ATOS	Quality review

Disclaimer

Table of Contents

List of Tables

List of Figures

Executive Summary

The current trend in cloud storage tells us that both the number of cloud users and their needs in terms of storage will keep on increasing. In order to accommodate these requirements in a cost-efficient manner, cloud storage providers must employ advanced data reduction techniques that are able to deal with all sorts of data, ranging from files to volumes. At the same time, additional services like encryption, secure deletion and verifiability mechanisms will be essential to foster cloud users' trust in the provided cloud services and encourage these users to outsource their sensitive and private data. Indeed, of one main objective of the TREDISEC project is to provide a unified framework that will integrates the various security primitives without sacrificing all the advantages that come from cloud computing.

In particular, the aim of this deliverable is to identify the specific verifiability requirements of TREDISEC uses cases and analyse the compatibility of existing verifiability solutions with data reduction techniques.

In order to save space, cloud storage providers typically use, as data reduction technique, data deduplication for eliminating duplicate copies of repeating data. When a cloud user uploads a file, the cloud storage provider checks if that file is already stored. If this is the case, the server just marks the cloud user as an owner of that file without storing any additional information. To potentially save also bandwidth, a cloud user can first sends to the cloud provider only a short identifier of the file and the provider checks if that identifier already exists in its database. If the identifier is not in its database then the provider asks for the entire file. Otherwise, since the file already exists at the server, the provider tells the client that there is no need to send the file. Either way the server marks the client as an owner of that file.

Among the many verifiability mechanisms, in this deliverable we are mainly interested in those that fall in the following categories:

- **Verifiable Storage:** Cloud users should be able to verify that their data is stored correctly and was not tampered by inadvertent or deliberate deletion. Verifiable storage allows a cloud user to check whether her (big) data is stored correctly at the cloud server provider. TREDISEC tackles this specific problem and currently investigates existing solutions that can be classified into two categories: Proof of Data Possession (PDP) and Proof of Retrievability (PoR).
- **Verifiable Computation:** When outsourcing computation, cloud users want to be assured that the result returned by the server is correctly computed. For that, users may require to either receive proofs of correct computations or to be assured that the computation was performed in a trusted environment.
- **Verifiable Ownership:** To avoid certain deduplication attacks, the concept of Proof of Ownership (PoW) has been introduced. More specifically, the idea is that if an outside adversary somehow obtains a bounded amount of information about a given target user file F via out-of-band leakage (i.e. F's short identifier), then the adversary cannot leverage this short information to obtain the whole file F by participating in the deduplication protocol with the cloud storage provider.

With the aim of pursuing the purpose of this deliverable, we have applied the following methodology:

- Starting from the requirements collected and organized in deliverable D2.2 [1], we have identified and reported the specific verifiability requirements of TREDISEC uses cases.
- Then, for each category described above, we have analysed the compatibility of existing verifiability solutions, meeting the verifiability requirements, with data reduction techniques.

Despite the relevance and novelty of many existing verifiability mechanisms, in most of the cases, they do not take in account the complexities of integrating verifiability solutions within a cloud

environment. This shows the importance of designing verifiability mechanisms that are compatible with data reduction techniques that are critical enablers for a number of popular and successful cloud storage services. In particular, verifiable storage and verifiable computation, among the three verifiability mechanisms analysed in this report, are the ones that have more compatibility issues with data reduction techniques.

Thanks to this analysis, the TREDISEC project can now focus on filling the gap in the harmonization of data reduction techniques with verifiability mechanisms.

Glossary of Terms

AES	Advanced Encryption Standard
AIK	Attestation Identity Key
APRF	Algebraic Pseudo-Random Function
BF	Bloom Filter
CA	Certification Authority
CDN	Content Distribution Network
CE	Convergent Encryption
CRS	Common Reference String
DES	Data Encryption Standard
ECC	Error Correcting Code
ECRH	Extractable Collision Resistant Hash Function
EU	European Union
FE	Functional Encryption
FHE	Fully Homomorphic Encryption
HE	Homomorphic Encryption
HMAC	Hash-based Message Authentication Code
IEEE	Institute of Electrical and Electronics Engineers
IP	Interactive Proof
MAC	Message Authentication Code
MLE	Message-Locked Encryption
PAKE	Password Authenticated Key Exchange
PCP	Probabilistic Checkable Proof
PCR	Platform Configuration Registers
PDP	Proof of Data Possession
PKI	Public Key Infrastructure
PoR	Proof of Retrievability
PoW	Proof of Ownership
PoL	Proof of Location
PPT	Probabilistic Polynomial Time
PRF	Pseudo-Random Function
QAP	Quadratic Arithmetic Program
RAM	Random-Access Memory
SNARK	Succinct Non-interactive ARgument of Knowledge
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TREDISEC	Trust-aware, REliable and Distributed Information SEcurity in the Cloud
UC	Use Case
VC	Verifiable Computing
VPN	Virtual Private Network
WP	Work Package
XTR	Efficient and Compact Subgroup Trace Representation

1 Introduction

1.1 Purpose and Scope

Since critical data storage and processing operations are performed remotely by potentially malicious cloud providers, users should receive guarantees on both the storage and processing of data whenever they request it by means of specific verifiability mechanisms. In addition, data reduction techniques usually imply the storage of a single (possible compressed) copy of data for several users: in this case, users having already outsourced data should prove their actual ownership. Additional complexity stems from the fact that solutions to the aforementioned challenges should not hinder the performance advantages offered by the cloud.

The purpose of this document is, therefore, to identify the specific verifiability requirements of TREDISEC uses cases and analyse the compatibility of existing verifiability solutions with data reduction techniques. The document first starts by summarizing the data reduction techniques currently available and then by identifying the specific verifiability requirements of TREDISEC use cases, described in deliverable D2.1 [2], extracted from the set of requirements collected and organized in deliverable D2.2 [1]. The remaining sections of this document focus on the existing solutions in the context of verifiable storage, verifiable computation and verifiable ownership, analysing the compatibility of these solutions with data reduction techniques.

1.2 Structure of the document

The document is structured as follows:

- Section 2 summarizes the data reduction techniques we will refer to analyze the compatibility of existing verifiability solutions.
- Section 3 identifies the specific verifiability requirements of TREDISEC use cases.
- Section 4 analyzes the compatibility of existing verifiable storage solutions with data reduction techniques described in Section 2.
- Section 5 focuses on existing solutions for verifiable computation.
- Section 6 focuses on verifiable ownership.
- The deliverable concludes with Section 7.

2 Data reductions techniques

Cloud computing is an emerging computational paradigm used to provide low-cost, scalable and location-independent services for cloud users. Many of these services allow outsourcing data to the cloud trying to avoid the upload and storage of repeating data, in an effort to minimize bandwidth and storage space usage. A promising technology in this direction, is *deduplication,* which stores only a single copy of repeating data. On the other side, however, cloud users are also interested in the privacy of their data and encryption provides the necessary tool to guarantee confidentiality. Compression of encrypted data can further reduce the storage space usage in specific settings.

2.1 Deduplication

In order to save space, cloud storage services perform deduplication across the files of all their users. In a typical storage system with deduplication, a client first sends to the server only a short identifier (e.g. a hash) of the file and the server checks if that identifier already exists in its database. If the identifier is not in its database, then the server asks for the entire file. Otherwise, since the file already exists at the server (potentially uploaded by someone else), the server tells the client that there is no need to send the file. Either way the server marks the client as an owner of that file that can therefore ask to restore the file at any time, regardless of whether he was asked to upload the file or not.

Deduplication strategies can be categorized according to the basic data units they handle. One category is file-level deduplication which eliminates redundant files [3]. The other is block-level deduplication, in which files are segmented into blocks and duplicate blocks are eliminated [4]. Deduplication strategies can also be categorized according to the location where deduplication happens. In server-side deduplication, all files are uploaded to the storage server, which then deletes the duplicates. Clients are unaware of deduplication. This strategy saves storage but not bandwidth. In client-side deduplication, a client uploading a file first checks the existence of this file on the server. Duplicates are not uploaded. This strategy saves both storage and bandwidth, but allows a client to learn if a file already exists on the server. Another important categorization is related to the ownership of the files on which deduplication is performed. In cross-user deduplication, each file or block is compared to the data of other users, and is deduped if an identical copy is already available at the server. In contrast, in the single-user deduplication setting, storage and bandwidth is saved only when the same user has multiple copies of the same data.

2.2 Secure deduplication

On the other side, however, users are interested in the privacy of their data and encryption provides the tool to guarantee data confidentiality. Unfortunately, conventional encryption makes deduplication impossible. We are therefore faced with the following scenario: Alice stores her file by encrypting it using her own secret key. Bob would store the encryption of the same file under his own secret key, different from that used by Alice. Two issues arise:

(1) How can the server detect that the data underlying the two ciphertexts is the same? And

(2) even if it can so detect, how to compute a short representation, in order to save space, of the two ciphertexts that allows both parties, based on their separate respective secret key, to recover the data from what is stored?

Douceur et al. [3] introduced the notion of Convergent Encryption (CE), a type of deterministic encryption in which a message is encrypted using a key derived from the plaintext itself. Convergent encryption does not achieve semantic security [5], the standard security definition for encryption, and only offers confidentiality for unpredictable messages. Later, Bellare et al., refined the concept of CE introducing a new primitive called Message-Locked Encryption (MLE) with a tailored security model to capture the best possible security achievable in this setting.

Keelveedhi et al. [6] proposed DupLESS, a server-aided encryption scheme (based on MLE) to perform data deduplication where the encryption key is obliviously computed based on the hash of the file and the private key of the assisting server.

In [7], Stanek et al. propose an encryption scheme which guarantees semantic security for unpopular data and weaker security (using convergent encryption) for popular files.

In [8], Liu et al. propose a secure cross-user deduplication scheme that supports client-side encryption without requiring any additional independent servers. The proposed scheme does not CE or MLE but introduces a key-distribution mechanism that uses in a novel and clever way a cryptographic primitive called PAKE (Password Authenticated Key Exchange) that allows two parties to agree on a shared secret key as long as the share the same password. Intuitively, if two cloud users deduplicate the same content, they will agree on a shared secret key.

In [9], Armknecht et al. propose a novel storage solution which allows a storage service provider to transparently attest to its customers the deduplication patterns of the (encrypted) data that it is storing. By doing so, this proposal enables cloud users to verify the effective storage space that their data is occupying in the cloud, and consequently to check whether they qualify for benefits such as price reductions, etc.

Finally, Puzio et al. in [10], propose PerfectDedup, a novel scheme for secure data deduplication, which takes into account the popularity of the data segments and leverages the properties of Perfect Hashing in order to assure block-level deduplication and data confidentiality at the same time. They show that the client-side overhead is minimal and the main computational load is outsourced to the cloud storage provider.

Both [9] and [10] are output of TREDISEC (WP4, Task 4.3).

2.3 Compression of encrypted data

In a traditional secure communication system, before being transmitted, data is first compressed and then encrypted. In most of the cases this approach is fine but there exist settings where there is a need to reverse the order in which data encryption and compression are performed. For example, consider a network of low-cost sensor nodes that transmit sensitive information over the internet to a cloud storage provider. The sensor nodes need to encrypt data to hide it from potential eavesdroppers, but they may not be able to perform compression as that would require additional hardware and thus higher implementation cost. On the other hand, the network operator that is responsible for transfer of data to the recipient wants to compress the data to maximize the utilization of its resources. Notice that, the network operator is not trusted and hence does not have access to the key used for encryption and decryption. If it had the key, it could simply decrypt data, compress and encrypt again.

Kilnc et al. [11] focus on compression of encrypted data where the encryption procedure utilizes block ciphers such as the Advanced Encryption Standard (AES) and Data Encryption Standard (DES). Loosely speaking, block ciphers operate on inputs of fixed length and serve as important building blocks that can be used to construct secure encryption schemes. Kilnc et al. [11] show that block ciphers in conjunction with the most commonly used chaining modes in practice are practically compressible for some types of sources.

Lenstra and Verheul [12] introduced the XTR public key system which is based on a new method to represent elements of a subgroup of a multiplicative group of a finite field. Application of XTR in cryptographic protocols leads to substantial savings both in communication and computational overhead without compromising security. Later, Rubin and Silverberg [13] introduced the concept of torus-based cryptography giving a new public key system called CEILIDH that improves on Diffie-Hellman, and also has some advantages over XTR. Later, Gentry [14] showed how to compress Rabin ciphertexts and signatures.

Homomorphic Encryption (HE) is a form of encryption that allows computations to be carried out on ciphertext. The result of the computation is still encrypted and when decrypted, it matches the result of computation performed on the plaintext. In this context, the problem of reducing the size of homomorphic ciphertexts as efficiently as possible has first been considered in [15]. In cloud applications, it is often assumed that some data is sent encrypted under a homomorphic encryption scheme to the cloud to be processed in some way. It is thus typical to consider, in the first step of

these applications, that Alice encrypts some data under some Bob's public key and sends some homomorphic ciphertext to Charlie, a third-party evaluator in the Cloud. The roles of Alice and Bob are clearly distinct, even though they might be played by the same entity in some applications. All HE schemes proposed so far suffer from a very large ciphertext expansion; bandwidth between Alice and Charlie is therefore a very significant bottleneck in practice. In [15], Alice first encrypts the data with a symmetric encryption scheme under some randomly chosen key, then Alice encrypts the symmetric key using HE, Charlie, then, exploits the homomorphic property of HE and recovers the original ciphertext by homomorphically evaluating the decryption circuit of the symmetric encryption scheme.

3 Use Case requirements

3.1 Overview

Within WP2 activities, six use-cases were defined (cf. D2.1 [2]) and categorised into two main groups (cf. D2.2 [1]): (1) **File sharing services:** This category subsumes UC 1, 2 and 3, and deal with data outsourcing to the cloud in a *multi-tenant* environment, and (2) **Big Data storage and secure processing services:** This class includes UC 4, 5 and 6, which mainly focus on the case where a single cloud customer outsources large amount of data to the cloud service provider for processing purposes. In addition, D2.2 [1] explored the various functional and non-functional requirements and identified not only the most relevant ones but also those that may not be met simultaneously.

This section, following the categorization given in D2.2, will identify the specific verifiability requirements of TREDISEC uses cases looking at those already collected and organized in deliverable D2.2, and possibly new ones.

3.2 Data sharing services

Use cases 1 through 3 are combining multi-tenant functionality with data reduction with deduplication. On the one hand, the privacy of tenants must be protected, but on the other hand common data among them can be exploited to reduce the storage costs for the cloud provider. This combination creates several requirements for WP3.

In Use Case 1 by GRNET, Securely Enhance Storage Efficiency, users upload new files on Pithos, a file-based cloud storage service. These files are split into blocks and each block is hashed to a content-addressable identity. Identical blocks are only stored once, therefore resulting in deduplication. Blocks already stored into the system need not be re-uploaded, therefore saving communication bandwidth and increasing file upload speed. However, there is an inherent privacy hazard in this scenario. A malicious user may try to guess the potential contents of a file by a user and ask the server to create such a file. If the server already possesses the block data it will not request them for upload, allowing the attacker to test for the existence of a document in the cloud.

This can be resolved by always requiring data upload, but that nullifies the important upload time optimization opportunity, which hurts user experience, especially for those users who make a lot of small-scale revisions to large datasets.

Therefore, a Proof-of-Ownership (PoW) mechanism should be developed according to requirement WP33-R1 (Efficient ownership verification) that will regulate the creation of new files before it proceeds with deduplication and any potential uploads. Use Case 3 by ARSYS has similar requirements but for file-based instead of block-based deduplication.

In both use cases, deduplication also creates a separate requirement in WP33-R4 (Verifiable ownership with data reduction). Deduplication gains are better if it is applied globally across tenants and therefore another requirement is made in WP33-R3 (Verifiable ownership with multi-tenancy).

Finally, to protect privacy in a multi-tenant environment, another restriction to verifiable ownership must be made according to WP33-R2 (Verifiable ownership with data confidentiality).

Because multi-tenancy is required by deduplication, Use Case 2 by GRNET is also a beneficiary of the same requirements from WP3 presented here.

From the tenant point of view, PoW must also be compatible with another use case requirement, secure deletion of data as codified in WP44-R1 (Secure deletion). Users may not trust a storage service if they cannot securely delete their content from it. Secure deletion, also called data erasure, ensures that data deleted by the user cannot be later recovered by another party, for example by inspecting raw data on disk blocks.

For example, in Use Cases 1 and 2, GRNET allows the storage not only of files, as described above, but also of virtual machine images and disk volumes. Secure data deletion should also apply on such

data, and not just on files, so that it encompasses all kinds of data handled by the GRNET cloud services.

The Use Cases suggest that verifiable ownership from WP3 and secure deletion from WP4 may interact with each other, because secure data deletion is also related to any kind of metadata that the service provider may need to store to provide a PoW functionality. When data is deleted, such metadata should be securely deleted as well, otherwise information on the data can be elicited indirectly by inspecting this metadata or traces of it. A further complication arises by the time dimension: should PoW be used to establish ownership of data in the past? That would be possible, for example, by analyzing logs kept by the server. If secure deletion applies not just to the data at present and from now on, but also to that data in the past, any past snapshots of the data and any logs associated with the data should also be securely deleted. Note, however, that altering logs may run counter to a host of regulatory requirements and may lead to operational problems for a cloud provider, as deleted logs will complicate or invalidate the effective usage of past logs. Such aspects should be carefully examined before any log altering solution is deployed into a production system. Solutions may involve anonymization and encryption of logs, which however, may fall outside the scope of the project.

3.3 Big Data services

Use-cases 4 to 6 consider processing of outsourced data. Ensuring correct storage of and computations over outsourced data raises several challenges and trade-offs between security and efficiency, even without data encryption. Achieving confidentiality of outsourced data together with verifiability properties leads to more complex challenges, and possible interactions with WP5.

The class of big data storage and secure processing services includes use-cases where a single cloud customer outsources data to a cloud service provider for processing purposes. Most of the time, data are sensitive and thus encrypted. UC4 considers the delegation of a biometric-based online authentication to a third party, who performs the authentication and, in addition, provides a proof that the authentication has been correctly performed. UC5 considers major upgrades of biometric systems, in which a large amount of encrypted biometric data is processed. The cloud generates a new database of encrypted biometric templates from a large amount of encrypted biometric images. UC6 describes the migration of a company's legacy data into a secure cloud environment, SQL queries can be executed over encrypted data stored in the cloud. Verifiability concerns are illustrated by UC4 and UC5. We now discuss in more details the relation between these use-cases and the trade-offs requirements

Computation integrity, aka verifiable computation, is mainly illustrated by UC4. The most important challenge here is to come up with efficient solutions. For this reason, requirements WP32-R5 (verifiable computation with efficiency at the cloud) and WP32-R6 (verifiable computation with efficiency at the client) are mandatory for UC4. As shown by the state of the art (cf. Section 5), the efficiency brought by current solutions is hardly satisfactory and clearly prevents for now the adoption of verifiable computing technologies. Then, data privacy issues in UC4 illustrate the need for verifiable techniques over encrypted data (requirement WP32-R8). This conflicting requirement is of interest for UC5 as well. As long as processing over encrypted data concerns are raised, interactions with WP5 are possible (cf. requirement WP53-R10: Privacy preserving data processing with verifiability). Finally, for UC5, verifiable techniques should also be efficient even with large databases (requirement WP32-R7).

As far as storage integrity is concerned, verifiable storage is illustrated by UC5. Although this requirement is not mandatory for UC5, tools for verifiable storage such as proof of possession and proof of retrievability may have proven useful. In this case, requirements WP31-R6 to WP31-R9 (respectively verifiable storage with efficiency at the cloud, dynamic data, storage efficiency, and multi-tenancy) might be considered.

4 Verifiable storage

Cloud storage has been gaining high popularity in recent years due to its elasticity and its pay-as-you-go approach. Customers of cloud services may easily and quickly adjust cloud resources to their needs without any pre-investment, and thus are relieved from the burden of storage. On the other hand, such advantages give rise to new security threats: Once users outsource their data, they can no longer control it, as the underlying storage system is deployed and maintained by a third party –the cloud provider– which cannot be totally trusted. Therefore, to foster the trust between customers and cloud providers, it is important to put in place effective mechanisms to check the correct storage of outsourced data.

The design of such mechanisms in TREDISEC is governed by the security requirements defined in D2.2. One of the goals that TREDISEC's solutions set out to achieve is to help the data owner verify the correct storage of her data efficiently (i.e. without downloading the entire data) and without hampering the performances at the cloud (cf. WP31-R1, WP31-R6).

Another goal that we aim to meet in TREDISEC is to accommodate data owners who wish to either delegate the data possession verification to a limited set of authorized users (for instance, for privacy reasons) (WP31-R4), or allow any third party to conduct the verification, in the case of public databases for example (WP31-R5).

One prominent approach to achieve these goals is Proofs of Retrievability (PoR), which offer guarantees of the retrievability of the outsourced data without requiring the data owner to download it (see WP31-R3). Given their high security guarantees, PoR mechanisms will serve as a starting point for the design of TREDISEC's solutions that will not only provide security, but also conform with the cloud's functional requirements.

4.1 Proofs of Retrievability

PoR enables the data owner to verify the integrity of her outsourced data by means of error-correcting codes, random-sampling, encryption and pseudo-random permutations. More specifically, a PoR scheme involves five polynomial-time algorithms[1]:

- $KeyGen(1^{\lambda}) \longrightarrow K$: It is probabilistic algorithm that is executed by the data owner. On input of security parameter λ, algorithm $KeyGen$ outputs a secret key K.

- $Encode(K, F) \longrightarrow (fid, F^{*})$: The data owner executes this algorithm to prepare the file for outsourcing. This algorithm takes the secret key K and the file F as input and returns an encoding F^{*} of file F and a unique file identifier fid. Algorithm $Encode$ should be invertible so that the data owner can recover the original file F from the encoding F^{*}.

- $Challenge(K, fid) \rightarrow (fid, chal)$: The data owner invokes this probabilistic algorithm to start an instance of the PoR protocol. This algorithm takes as input secret key K and a file identifier fid, and returns a challenge $chal$ along with the file identifier fid.

- $Prove(fid, chal) \longrightarrow \pi$: Upon receipt of challenge $chal$ and file identifier fid, the cloud server calls this algorithm to generate a proof of retrievability π for the file with identifier fid.

- $Verify(K, fid, cha, \pi) \longrightarrow b$: The data owner executes this deterministic algorithm to verify the validity of the proofs of retrievability provided by the cloud server. On input of secret key K,

[1] For the sake of simplicity, we only present here the algorithms of symmetric PoR.

file identifier fid, challenge $chal$ and proof π, algorithm $Verify$ outputs $b = 1$ if it is convinced that proof π is correct; otherwise, it returns $b = 0$.

We say that a PoR scheme is secure if it is both correct and sound.

4.1.1 Correctness

We say that a PoR scheme is correct if and only if, whenever the cloud server is honest (i.e. does not tamper with the outsourced files), then executing algorithm *Prove* always yields proofs of retrievability that will be accepted by algorithm $Verify$ (i.e. $Verify$ outputs $b = 1$).

4.1.2 Soundness

We say that a PoR scheme is sound, if for any malicious cloud server, the only way to convince the data owner that it is storing a file F is by keeping a retrievable version of that file. This means that if a cloud server is able to generate valid proofs of retrievability for some file *F* (i.e. proofs that will be accepted by algorithm $Verify$), then this cloud server must possess an encoding *F'* that can be used later by the data owner to recover the original file F.

4.2 State of the art

Before the PoR's formalisation of secure storage, other security definitions and proof of storage schemes were proposed: Notably, there is remote integrity checking, remote data possession checking and provable data possession. Each of these terms embodies different security requirements that the verification procedure should fulfil. We classify existing solutions into two main categories: (i) deterministic solutions that offer an undeniable guarantee of integrity and (ii) probabilistic solutions such as PoR that offer better performance results with guarantee of integrity under a certain (high) probability.

4.2.1 Deterministic solutions

The first category of solutions also referred to as remote integrity checking, offer the verifier the possibility to be certain of the integrity of the outsourced data. On the downside however,, deterministic solutions generally induce communication and computational costs that usually are linear in the size of the entire data. A simple solution consists of a classical integrity technique as: When the user uploads a file *F* to the cloud, he computes and locally stores the crypto checksum of *F*. At the time of verification, the user and the server engage in a challenge–response protocol that involves the transfer of *F* back to the user, the re–computation of the checksum of *F* by the user and finally the comparison of the two values. Recent work [16], [17], [18], [19] optimize the above approach by reducing the amount of communication required by the verification.

4.2.2 Probabilistic solutions

The second category of proposals aims at reducing the communication and computational complexity of the challenge–response protocol at the price of not offering an unconditional data integrity guarantee. The core idea of these proposals is to check the integrity of a subset of segments of the file *F* instead of checking the integrity of the entire file *F,* while still providing assurance for the integrity of the entire file with high probability (WP31-R1, WP31-R2, WP31-R6). Existing solutions either secretly insert some randomly generated blocks into the data and retrieve them during the verification phase or generate a tag per block and retrieve a subset of them together with the corresponding blocks.

4.2.2.1 Sentinel based solutions

Juels and Kaliski [20] proposed the proofs of retrievability (PoR) model that ensures that users at any point of time can retrieve outsourced data (WP31-R3). This property is achieved by means of error-

correcting codes (ECC) and randomly generated blocks. The scheme in [20] embeds sentinels (-pseudo–randomly generated blocks) into encrypted stored data, hence data blocks and sentinels are indistinguishable. To check the retrievability of the data, the verifier selects a random subset of sentinels and queries the server for them. This proposal only supports a bounded number of POR queries after which the server may discover all the embedded sentinels. Further proposals [21], [22] allow the client to batch updates to the data.
Azraoui et al. [23] pursue the approach in [20] with StealthGuard. In this scheme the data owner inserts pseudo-random blocks, called WatchDogs, before outsourcing the data to the cloud. To achieve unbounded number of verification queries StealthGuard uses a privacy–preserving word search scheme to verify the existence of WatchDogs without disclosing their position to the remote server.

4.2.2.2 Tag based solutions

Ateniese et al. [24] define the provable data possession (PDP) model, which ensures that a large portion of the outsourced data is stored in storage servers. The authors proposed a PDP scheme that uses linearly homomorphic tags as check–values for each data block. Thanks to the homomorphic property of the tags, the server can combine tags of multiple data blocks into a single value, reducing thus the communication overhead of the verification. This proposal was later extended in [25] where the notion of robust auditing was defined. Their new scheme integrates error-correcting codes to mitigate arbitrarily small file corruptions.
Following another direction, Ateniese et al. [26] proposed a symmetric PDP construction to support dynamic writes/updates on stored data. The idea is to precompute challenges and answers as metadata in advance, however, this limits the number of verifications the data owner can conduct. Erway et al. [27] proposed a dynamic PDP scheme that relies on authenticated dictionaries based on rank information to support dynamic data operations. Later on, Chen and Curtmola [28] extended the work in [25] in order to support data updates.
Most proposals for PoR employ homomorphic tags like in [24], in conjunction with error-correcting codes. Shacham and Waters [29] propose symmetric based as well as public based PoR schemes that use homomorphic authenticators to yield compact proofs. This work has been extended by Xu and Chang in [30] where a polynomial commitment protocol combined with homomorphic tags leads to lower communication complexity. Dodis et al [31] generalize the scheme of [20], [29] and introduce the notion of PoR codes, which couples concepts in PoR and hardness amplification. Cash et al. [32] proposed a dynamic PoR scheme that relies on oblivious RAM (ORAM) to perform random–access reads and writes in a private manner. Subsequently, Shi et al. [33] proposed a dynamic PoR scheme that considerably improves the performance of [32] by taking advantage of a Merkle tree (cf. Section 6.2.1).

4.2.2.3 PoR with public verifiability and reliability

Other contributions propose the notion of delegatable verifiability of PoR (WP31-R4). For instance, in [34], [35] the authors describe schemes that enable the user to delegate the verification of PoR and to prevent their further re-delegation. Furthermore, Wang et al. [36], [37] proposed a PoR scheme that supports dynamic operations and public verification on stored data (WP31-R5). Their construction uses Merkle trees to support data dynamics. Wang et al. [38] proposed privacy–preserving public auditing for data storage security in cloud computing. Their scheme uses a blinding technique to enable an external auditor to verify the integrity of a file F stored in a server without learning any information about the file contents. Armknecht et al. [39] introduce the notion of outsourced proofs of retrievability, an extension of the PoR model, in which users can task an external auditor to perform and verify PoR on behalf of the data owner (WP31-R4).
Bowers et al. [40] proposed a theoretical framework of designing PoR protocols. This framework employs two layers of error-correcting codes in order to recover user data from a series of responses. Their work improves previous results of PoR and PDP and has security proved in the Byzantine adversarial model.
To simultaneously achieve high availability and integrity verification for stored data, multiple replicas may be employed. Curtmola et al. [41] proposed a multiple–replica PDP that allows a user to verify

that at least *t* unique replicas of a file exist in storage servers. Later on, Bowers et al. [42] proposed HAIL, which provides a high–availability and integrity layer for cloud storage. In order to guarantee data retrievability among distributed storage servers, HAIL uses erasure codes on the single and multiple server layers respectively. Bowers et al. [43] propose a scheme that ensures that data is stored redundantly in multiple servers by measuring the time taken for a server to respond to a read request for a set of data blocks.

4.2.3 Proof of Location using PoR

Proofs of Location (PoL) [44], [45] aim at proving the geographic position of data, e.g., if it is stored on servers within a certain country. In [45], Watson et al. provide a formal definition for PoL schemes by combining the use of geolocation techniques together with the tag-based PoR schemes.

4.3 Summary

Proofs of retrievability combine random sampling with ECC (Error Correcting Codes) and cryptographic primitives including semantically secure encryption, random number generators or homomorphic authenticators, to enable lightweight data owners to verify the availability of their outsourced data efficiently, that is, without downloading the data in its entirety. While naturally data owners may push for the implementation of PoR mechanisms in the cloud, cloud service providers may not be as enthusiastic. Indeed, proofs of retrievability, be they sentinel-based or tag-based, hinder the financial advantages that the cloud service providers glean from using de-duplication.

More precisely, sentinel-based approaches rely on semantically secure encryption to hide the positions of the randomly generated sentinels. Such a powerful encryption solution is not compatible with deduplication and if instead, convergent encryption is used, the cloud can with high probability guess which blocks are the sentinels, as they will not be deduplicated. On the other hand, tag-based PoRs use encryption to hide the dependencies between original blocks and the redundancy blocks produced by the ECC; additionally, they incur a storage overhead due to the generation of PoR tags, that is linear in the size of the outsourced data and these tags cannot be de-duplicated because they are generated by each data owner separately.

As a conclusion, despite the relevance and novelty of existing PoR mechanisms, they overlook the intricacies of integrating PoR within a cloud environment, and suppose that cloud providers will adopt traditional PoR solutions even if it comes at the expense of their financial bottom-line. This shows the importance of designing PoR solutions that are compatible with de-duplication (WP31-R8).

In addition to devising PoR mechanisms that are compatible with deduplication, TREDISEC also strives at developing verifiable storage solutions that support efficient data updates (WP31-R7): The main problem raised by data updates when using PoR schemes is that when they are performed naively, they leak information about the dependencies between data blocks and redundancy blocks, and the cloud server can use this information to tamper with the outsourced files without risking detection. Existing solutions that couple PoR with updates either resort to ORAM, which offers provable security guarantees by allowing oblivious read and write operations, or use batch updates, which are more efficient than ORAM but whose security is only based on heuristics. On account of these shortcomings, it may be more suitable for TREDISEC to investigate more traditional verifiable storage techniques, especially if the updates concern relatively small amounts of data.

Finally, considering use-case scenarios of file sharing, TREDISEC should also make sure that at least one of our solutions for verifiable storage operates well within a multi-tenant environment (WP31-R9). Again we note here that in such scenarios it may be more efficient not to rely on PoR-based solutions.

5 Verifiable computation

The question of verifying computations that are delegated by a client to remote server is not new. This problem raised a lot of challenges, theoretical studies as well as practical solutions. From the practical point of view, existing classical techniques are often based on replication of the computations. Other concrete solutions assume trusted hardware, or are dedicated to very *specific* functionalities. On the other hand, theoretical solutions for verifying *generic* computation exist, but are far from being practical. Most of them are purely theoretical objects. In any case, even if one manages to come up with a practical solution, the most challenging requirement – from a concrete point of view – is being convinced of the correctness of a delegated computation faster than carrying out the computation itself. In a nutshell, according to the current state of the art, the trade-off requirements for verifiable computation with efficiency at the cloud (WP32-R5) and the client (WP32-R6) are hard to satisfy, even if the requirements for verifiable computation with big databases (WP32-R7) and encrypted data (WP32-R8) are not considered.

5.1 High-level definition of verifiable computing systems

Let F be a function. A non-interactive publicly verifiable computing system basically involves two entities, a prover and a verifier, and three procedures. After a setup procedure, where an evaluation key and a verification key are generated, a prover evaluates the function on some input, and generates a proof of correctness for this evaluation, thanks to the evaluation key. Then, anyone can verify, given the input/output of the function, the proof, and the verification key, the validity of the computation.

- $Setup(1^{\lambda},) \rightarrow (EK_F, VK_F)$. On input a security parameter λ, the setup procedure generates two set of public parameters, an evaluation key EK_F for the prover, and a verification key VK_F for the verifier.
- $Prove(EK_F, u) \rightarrow (y, \pi)$. On input an evaluation key EK_F, an input u for the function F, the prove procedure generates the output y of the function F evaluated on input u, together with a proof of computation π.
- $Verify(VK_F, u, y, \pi) \rightarrow d$. On input a verification key VK_F, an input u, an output y, and a proof π, the verification procedure outputs a decision $d \in \{0,1\}$.

A verifiable computing system should be correct and sound. The correctness states that honestly generated proofs are always accepted by the verification procedure. The soundness states that a malicious prover is not able to make a verifier accept an incorrect computation.

- Correctness. For any function F, any input u and enough large security parameter λ, for honestly generated parameter $Setup(1^{\lambda},) \rightarrow (EK_F, VK_F)$, and proofs $Prove(EK_F, u) \rightarrow (y, \pi)$ we have $Verify(VK_F, u, y, \pi) = 1$.
- Soundness. For any function F, any enough large security parameter λ, any parameters $Setup(1^{\lambda},) \rightarrow (EK_F, VK_F)$ and any probabilistic polynomial-time adversary A, $Pr[(u', y', \pi') \leftarrow A(EK_F, VK_F) : F(u') \neq y' \wedge Verify(VK_F, u', y', \pi) = 1] \leq negl(\lambda)$

We now sketch the main variants of verifiable computing systems.

Interactive vs. non-interactive. Interactions between the prover and the verifier might be possible. A non-interactive system might be seen an interactive system where the proof π in the definition above is the unique message sent from the prover to the verifier.

With or without pre-processing phase. A proof system might only be composed of some interactions between a prover and a verifier. In this case, there is no setup procedure. However, several systems attempts to amortize the cost of verifying computations by carrying a setup phase once.

Public / private verification. In publicly verifiable systems, anyone can check a proof of correctness. However, in some settings only a verifier knowing some secret material can perform the verification. In particular, it might be the case if a proof has to be performed on encrypted data.

Two or more entities. The prover and the verifier are the basic entities of a verifiable computing system. However, the roles might be split into more than two entities. For instance the entity who generates the public parameters might be a third party. Moreover, the prover might be split into two entities, some entity, which preprocesses the input of the function, generating a query for the proof engine.

5.2 State of the art

5.2.1 Theoretical solutions

5.2.1.1 Probabilistically checkable proofs

The problem of delegated computations is solved, in theory, thanks to Interactive Proofs (IP) [46] and Probabilistically Checkable Proofs (PCP) [47]. As their name implies, PCPs allow a remote server (acting as a super-polynomial prover) to generate a proof of correct computation that can be efficiently checked by a client (relatively weak verifier). Namely, the verifier needs to only check a few bits in the proof to decide whether the computation is correct.

5.2.1.2 Interactive arguments

On the downside however, PCPs generally yield proofs that are too large to be read by the verifier. To overcome this caveat, Kilian [48] proposed to focus rather on interactive arguments which are characterized by a polynomially bounded prover. The idea of Kilian is instead of providing the verifier with the actual proof, the prover commits to its (too large) proof using a Merkle tree whose root is then sent to the verifier. The prover afterwards reveals interactively the bits requested by the verifier. Thanks to the security of Merkle tree, the verifier securely checks whether the received bits are correct. In a more recent work, Goldwasser *et al.* [49] introduced Muggle proofs which albeit efficient (verification is linear in the size of the inputs), only support computations represented as circuits of poly-logarithmic depth.

5.2.1.3 Non-interactive arguments

In the Common Reference String (CRS) model, Micali [50] showed how to transform the interactive argument of Kilian into a non-interactive one in the random oracle model. To get around the random oracle and PCPs, Bitansky *et al.* [51] relied on extractable collision resistant hash functions (ECRHs) and built the first succinct non-interactive argument of knowledge (SNARK). While the security of this scheme is proved in the standard model, it is only ensured under a non-standard assumption (*i.e.* the existence of ECRHs). Unfortunately, in [51] and [52], the authors demonstrate that we cannot do better. Notably, Bitansky *et al.* [51] showed that SNARKs entail the existence of ECRHs.

5.2.1.4 Quadratic arithmetic programs

Gennaro *et al.* [53] also moved away from the PCP-based approach and proposed one of the first SNARKs for arithmetic circuits with constant-size proofs. This solution uses Quadratic Arithmetic Programs (QAP), whereby the circuit to be evaluated is described through three sets of polynomials. More precisely, each gate in the circuit is encoded by means of three polynomials: the first two polynomials encode the left and the right inputs, whereas the third encodes the output of the gate. The prover commits to the circuit via a target polynomial which is computed based on the three set of polynomials, and which verifies the following: at each evaluation of the circuit, the prover outputs a polynomial that divides the target polynomial if and only if the circuit is evaluated correctly. It follows that Gennaro *et al.* show that the proposed scheme is sound against adaptive adversaries under the knowledge of exponent and power Diffie-Hellman assumptions, and that it generates publicly verifiable proofs of constant size that can be verified in time linear in the size of the circuit inputs. [54] and [55] draw upon the techniques introduced in [53] and proposed general methodologies for designing SNARKs, which result in relatively more efficient solutions.

5.2.1.5 Homomorphic encryption and functional encryption

There are solutions for verifiable computation that tailor the properties of either Fully-Homomorphic Encryption (FHE) [56, 57] or variants of Functional Encryption (FE) [58]. Despite the considerable advances witnessed in both FHE and FE, the resulting schemes are far from being practical: they are only suitable for Boolean functions and generate proofs that are prohibitively large.

5.2.2 Practical Generic Solutions

Although the theory of verifiable computation is quite mature, only recently did it become interested in the practical aspects of implementation. We refer to [59] for an excellent survey of recent trends in verifiable computing.

5.2.2.1 Implementations of interactive proofs

[60] gives an efficient implementation of the Muggle protocol [49], far more efficient than a naive implementation, refined later by Thaler [61]. Generally, the work based on interactive proofs only handles a restrictive class of computation – the circuit must contain a form of regularity – but, for these circuits, the resulting systems are rather efficient (and optimal for some case with [61]). The AllSpice system [62] also follows this line of work with a relaxed condition on the regularity.

5.2.2.2 Implementations of non-interactive arguments

The efficient commitment-based protocol of (often referred as IKO) was followed by verifiable computing systems like Pepper [63] and Ginger [64]. As in the interactive proofs, a kind of regularity condition must be met for the computation to be processed. These restrictions are relaxed by the Zaatar system [65], through the integration of QAP techniques [53].

5.2.2.3 Implementations of quadratic arithmetic programs

One of the prominent attempts is Pinocchio [66] which improves the theoretical solution introduced in [53] and propose a tool-chain that transforms C code into an arithmetic circuit which is then encoded as QAP. As a follow-up, Gepetto [67] optimizes the proof generation of Pinocchio using a new technique called multi-QAPs that divides an arithmetic circuit into smaller circuits. Pinocchio and its successors (including [68]) and BCGTV [69] provide a kind of privacy. The underlying proof system can be made zero-knowledge, nearly for free. As a result, the prover can hide part of the inputs to the verifier while keeping the delegated computation verifiable.

5.2.2.4 Extending the model of computation

We stress however that the precedent work around QAP only consider computations that are modelled as arithmetic circuits, which is less expressive than standard programming language. The work of [70] and [69] applies the results in [53] and [54] to RAM programs. BCGTV [70] introduces a verifiable computation system for a simple RAM model, called TinyRAM. The class of computations that can be processed is very large. It includes notably random memory access and general loops, which are of primary importance in standard algorithms. However, the description of the computation is rather complex, and Pinocchio gives better performance than BCGTV. Pantry [68] and its successor Buffet [71] extend Zataar and Pinocchio to work with the RAM model of computation. Pantry does not allow general loops, but seems to perform better than BCGTV, except if there is too much memory accesses. A recent work [69] uses memory routing networks to achieve better memory management (Pantry uses hash trees).

5.2.2.5 Homomorphic authenticators

Another line of work exploits fully homomorphic authenticators (MACs and signatures) to authenticate the output of arbitrary functions. While for most homomorphic authenticators, the verification process is at least as expensive as the computation itself, recently designed solutions strive to accommodate efficient verification. For instance, Backes *et al.* [72] build upon Algebraic Pseudo-Random Functions (APRFs) to design efficiently verifiable homomorphic MACs. While the proposed solution is secure against adaptive adversaries, it only supports quadratic functions. Catalano *et al.* [73] use leveled multilinear maps to design adaptively secure homomorphic signatures suitable for polynomial

functions, and which can be verified efficiently. In their seminal work, Gorbunov *et al.* [74] propose the first homomorphic signature that authenticates arbitrary arithmetic circuits. The solution in [74] is based on standard lattices, and similarly to [73] gives way to a verification that does not depend on the size of the inputs and outputs of the function to be authenticated.

5.2.3 Practical Dedicated Solutions

The relatively high cost of the generic solutions led to an active area of research that focuses on the verifiability of specific functions, namely, polynomial evaluation, matrix multiplication, membership testing and set operations. The rationale behind this work is that these functions underpin many real world applications that range from data mining and signal processing to data-structures and keyword search. By leveraging the mathematical features of each function, one is able to devise solutions that outperform the generic ones. For instance, Fiore and Gennaro [75] draw upon some of the techniques used in [76], more specifically APRFs, and propose protocols for publicly verifiable polynomial evaluation and matrix multiplication that are sound against adaptive adversaries. As follow-up, [77] reduces the storage requirements at the prover by breaking up the computation to be verified into smaller sub-computations. This however comes at the price of a more expensive proof generation. Papamanthou *et al.* [78] designed an authenticated hash table that enables efficient membership testing through a combination of cryptographic accumulators and flat Merkle trees. The proposed scheme as a result produces constant-sized proofs than can be verified in constant time. Moreover, coupling polynomial-based commitments and flat Merkle trees led to nearly practical protocols for verifiable set operations [79, 80] that can aid the development of viable solutions for verifiable multiple-keyword search.

5.2.4 Hardware-based solutions

Some solutions have been designed, which rely on the use of some secure hardware [81, 82, 83, 84]. The security analyses of these solutions assume that the hardware protection cannot be broken. Different architectures have been proposed. [81] proposes “SP”-architecture (Secret-Protected), in which user secrets can be accessed on-line by a multitude of devices. [82] proposes XOM (eXecute Only Memory), a solution for copy- and tamper-resistant software, which enables to block unauthorized execution of the software. [84] introduces Pioneer, a solution for verifiable code execution on untrusted hosts. [83] explores a hybrid solution for verifiable computing where a trusted hardware is used with Secure Function Evaluation (SFE) to compute arbitrary functions on secret data. Below we detail the Trusted Platform Module architecture.

The TPM is a microcontroller that stores keys, passwords and digital certificates. It typically is affixed to the motherboard of a PC. It potentially can be used in any computing device that requires these functions. The nature of this silicon ensures that the information stored there is made more secure from external software attack and physical theft. Security processes, such as digital signature and key exchange, are protected through the secure TCG subsystem. Access to data and secrets in a platform could be denied if the boot sequence is not as expected. Critical applications and capabilities such as secure email, secure web access and local protection of data are thereby made much more secure. TPM capabilities also can be integrated into other components in a system [98].

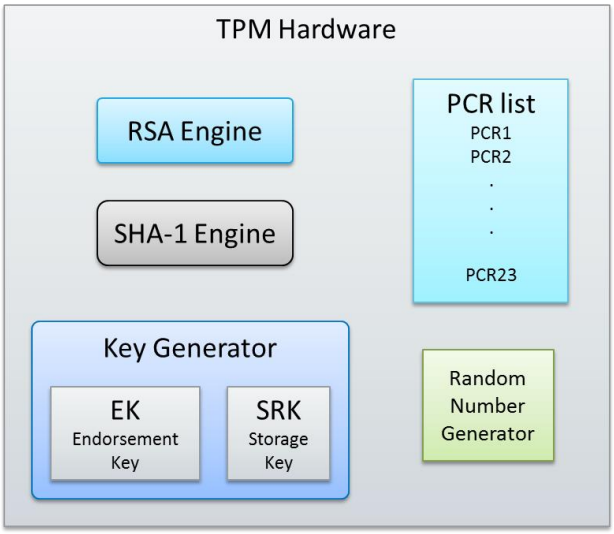

Figure 1: TPM components

The TPM components relevant to the verifiability purpose are shown in Figure 1.The TPM provides secure storage and key generation capabilities, similar to other hardware authentication devices, so it can be used to create and/or store both user and platform identity credentials for use in authentication. The TPM can also protect and authenticate user passwords, thereby providing an effective solution of integrating strong, multifactor authentication directly into the computing platform. With the addition of complementary technologies such as smart cards, tokens and biometrics, the TPM enables true machine and user authentication.[98]The Endorsement Key (EK) is a 2048-bit RSA public and private key pair, created randomly at manufacture time and cannot be changed.

The EK is unique to each TMP chip, and it's nearly impossible to read the private part without physically destroying the chip, therefore, each RSA key created is traceable to a unique TPM since all the created keys have the EK as parent.

The PCR (Platform Configuration Registers) is a list of SHA-1 hashes that can't be modified directly, except by a special method to extend their values, using a file content and the current value as input.

By taking measures on a handful of essential files, the TPM can be used to determinate if there was a change to previous configurations in the platform, and validate that the current state is the expected one. Additionally, the TPM can create a randomly generated data, to make a signature file bound to the current state of the PCRs, so the platform can be remotely validated (Remote Attestation) without actually giving away any crucial information from private files, [99].

5.3 Summary

Although the verification of computation is an old topic in cryptography, research has only supplied theoretical solutions for a long time, far beyond any practical level. Major theoretical breakthrough were achieved recently, around 2007-2008. A lot of successive optimizations gave rise to the first practical solutions, in the interactive setting as well as in the non-interactive setting. However, the practicality achieved by the current solutions is clearly not fully satisfactory.

Globally, the verification of proofs is quite efficient now. The bottleneck basically remains in the computational power needed for the generation of the proofs, even for solutions with a pre-processing step carried out once. The key size easily gets huge.

A lot of work has also to be done regarding the different classes of computations that can be processed by current techniques. Interactive solutions achieve better efficiency than non-interactive ones. However interactive solutions may not fit all the scenarios of interest. Moreover, the most efficient interactive solutions are only applicable to restricted class of computations. This motivates

the need to new solutions in both settings.

Then, data privacy is not addressed in the basic verifiable computation model. Although proofs of computation might sometimes be made zero-knowledge, this property does not affect the prover. Indeed, in current constructions, the prover must know all inputs and intermediate values of the computation in order to compute a proof.

As far as the compatibility with data reduction is concerned, different angles might be considered. If the concern is, for the client, to check the correctness of the data reduction performed by the cloud, then the issue is to exhibit verifiable computation systems able to process such computations. The standard problems of verifiable computing for general computations, sketched above, translate in the particular case of data reduction computations, deduplication as well as data compression.

More problems are added up if the compatibility takes the encryption level into account. Most of verifiable systems do not apply on encrypted data. When they do, data is encrypted with homomorphic encryption. Here an additional issue appears in the case of deduplication, since existing techniques for secure deduplication are based on other encryption tools such as convergent encryption.

6 Verifiable ownership

Deduplication can be abused by exploiting the connection between a file (or data block) and its corresponding short identifier. Recall that, in a typical storage system with deduplication, a file is deduplicated only if its unique short identifier (e.g., hash) matches the short identifier of a file already present in the cloud. This allows any user of the system to identify whether a certain file was previously uploaded. For instance, let's assume that Alice suspects that Bob has a sensitive file *F* which is unlikely to be at the possession of any other user (Bob's payslip). Then, Alice can use deduplication to check whether this is the case by trying to backup a copy of *F* and check whether deduplication occurs. The attack described above can be turned into a brute force attack by simply checking all the possible values of the content of the file (which reduces to Bob's salary for Bob's payslip case). Moreover, a malicious user could abuse the storage provider using it as a Content Distribution Network (CDN) in the following way: A user uploads a copy of a file (containing potentially illegal content) to the storage provider and publishes somewhere the short identifier of that file. Then anyone can gain the ownership of that file by attempting to upload it to the storage provider, presenting the hash to the service and be added as an owner of that file, and then asking to restore the file.

In this Section, in order to mitigate the abuses described above, we will present the concept of *Proof of Ownership* (PoW), walk through the literature for the state-of-the-art solutions in this field, and, as prescribed by requirement WP33-R4 [1] (Verifiable ownership with data reduction) and this deliverable, investigate the compatibility of PoW with data reduction techniques. The adoption of Proof of Ownership protocols should neither affect the data reduction function nor should it ideally reduce its efficiency. In doing so, we will also analyse PoW with the respect the following requirements from [1]: WP33-R1 (Efficient ownership verification), WP33-R2 (Verifiable ownership with data confidentiality), WP33-R3 (Verifiable ownership with multi-tenancy), WP33-R5 (Verifiable ownership with dynamicity).

6.1 Proof of ownership

The root-cause of the above threats lies in the fact that the proof of ownership solely relies on the knowledge of a static, short piece of information. In order to mitigate the above threats, Halevi et al. [85] introduced the concept of Proof of Ownership. Informally, the idea is that if an adversary somehow obtain a *bounded amount* of information about the target user file via out-of-band leakage, then the adversary cannot leverage this short information to obtain the whole file by participating in the client-side deduplication with the cloud storage server.

More specifically, a PoW scheme is a security protocol used by a *server* to verify that a *client* owns a file without the need for a full upload. It consists of a probabilistic algorithm S, called also summary function, and a protocol between two probabilistic interactive algorithms $\Pi(P,V)$ having the following semantics:

1. $S(1^{\lambda},F)$ takes as input a positive integer as security parameter λ in unary, and a file F, an output a short summary value ϕ, where the bit-length of ϕ depends solely on the security parameter and independent of the file size $|F|$.
2. $\Pi\left(P(1^{\lambda},F),V(\phi)\right) \rightarrow \{Accept, Reject\}$: The prover P, which takes as input the security parameter λ and a file F, interacts with the verifier V, which takes as input a short summary value ϕ, and outputs either $Accept$ or $Reject$.

The main efficiency parameters of a PoW are the following:

- *The size of the summary information* $\phi = S(1^{\lambda},F)$. Given the high server workload, an efficient PoW has to allow the server to store only an extremely short information per file that is sufficient to perform the protocol, without having to fetch the entire file contents for verification.

- *The communication complexity of the protocol*. In particular, the protocol must be bandwidth efficient consuming much less bandwidth than the size of the file (otherwise the client could just send the file itself to the server);
- *The computational complexity of* S *and* P (all with respect to the file size $|F|$ and the security parameter λ). Solutions should have linear dependence on the security parameter, and computation complexity of S and P linear in $|F|$ and everything else is at most (poly) logarithmic in $|F|$. Another efficiency parameter that can be important is the space complexity of S, P when viewed as one-pass algorithms that access the input file in a streaming fashion. (This is significant since for large files one would like to read them sequentially from disk only once, and enough memory to store them all is not available.)

A proof of ownership scheme must satisfy the following properties:

Correctness: For correctness, it is required that for all security parameters λ and for any binary file F of length polynomial in λ, $\Pi(P(1^\lambda, F), V(S(1^\lambda, F)))$ outputs $Accept$ with all but a negligible probability in the security parameter.

Security: Informally, a PoW scheme is considered secure if the probability of being able to dishonestly prove the ownership of a file is negligible in the security parameter, even if the attacker possesses a relevant part of the file.

More formally, security is defined as a game, between a challenger C and a PPT adversary A, with parameters λ, k, T (whose meaning is explained in the game definition), and defined as follow:

- *Setup Phase*: A submits a distribution D over $\{0,1\}^M$, where M is any positive-integer, with min-entropy at least k. C samples a file F according to distribution D and runs the summary algorithm to obtain $\phi = S(1^\lambda, F)$.
- *Learning Phase*: A can adaptively submit polynomially many queries to C, where each query type can be one of the following and concurrent queries of different types are not allowed.
  - *Prove-Query*: C, running the verifier algorithm V on input ϕ, interacts with A, which plays the role of the prover, to obtain $b = \Pi(A, V(\phi))$.
  - *Leakage-Query*: This query consists of a description of a PPT algorithm P'. C answers this query by running $P'(1^\lambda, F)^{V(\phi)}$ to obtain a string y, and sending y to A. Denote Y as the bit-length sum of all the y-responses.
- *Challenge Phase*: C, running the verifier algorithm V on input ϕ, interacts with A, which plays the role of the prover, to obtain $b = \Pi(A, V(\phi))$. A wins if $b = Accept$ and $Y \leq T$.

A proof of ownership scheme is $(\lambda, k, T, \epsilon)$-secure if for any positive-integer λ, any positive-integer k, any input distribution D with k bits of min-entropy, soundness error $\epsilon \in [0,1]$, and any PPT adversary A as above that receives less than T bits of leackage, the probability of A winning the above game is at most negligible in λ more than ϵ.

Informally, as long as the min-entropy in D is sufficiently larger, meaning that D is sufficiently unpredictable, compared to the number of bits leaked to the adversary, the adversary should not be able to convince the server that it has the file. Notice that, a PoW does not protect against an adversary who uses the rightful owner of a file as an interactive oracle to obtain the correct responses to a PoW challenge.

6.2 State of the art

6.2.1 Merkle tree-based solutions

Halevi et al. [85] presented three PoW schemes based on Merkle Trees built on top of the content of the original file.

A Merkle [86] or hash tree is constructed by splitting an input buffer (e.g., the file to be deduplicated) into multiple blocks which are hashed and become the leaves of the tree. These hash values are grouped in pairs and each pair is again hashed, producing the next level of the tree which will contain half of the nodes of the previous level. The process is repeated until reaching a level with a single node which represents the root of the Merkle tree. Each leaf node has a corresponding sibling path, which consists of the leaf's value and the values of all the siblings of nodes on the path from the given leaf node to the root of the tree. The values of all nodes on the path from a given leaf to the root can be computed from bottom to top, given just the index of the leaf and its sibling path. To verify that the client is in possession of a file, the server selects a number of leaf indexes and asks the client to provide the corresponding leaf values and the sibling path for each leaf. The server will accept the proof if all sibling paths are valid. A sibling path is said to be valid if its length (e.g., the number of nodes it contains) is the same as the tree height and if the root value computed on the sibling path corresponds to the root value stored on the server.

Halevi et al. proposed three different ways to build the Merkle tree which achieve different level of security and efficiency.

The first one builds the Merkle tree from an erasure coding of the original file in order to spread unknown blocks of the file over a high number of blocks of the corresponding erasure coded version. Here, the server ask for a super-logarithmic number of leaves, chosen at random. The erasure code guarantees that if the adversary is missing some part of the file then there are at least a fixed threshold of the leaves that it does not know. In addition, if the file has high min-entropy then the adversary is not even able to guess the content of the leaves it does not know. Therefore, with overwhelming probability the adversary will not succeed in convincing the server. On the client site, this solution is very costly. In fact, computing the erasure code requires random access to the file and if the file does not fit in RAM this requires many disk accesses. On the server side, this scheme requires the server to ask for a number of leaves which is super-logarithmic in the security parameter.

In order to avoid the inefficiencies of the erasure coding, the second scheme replaces it with a universal hash function. The solution then consists of hashing the potentially large file into a reduction buffer of size B, and then run the Merkle-tree protocol on the reduction buffer. This comes with a security requirement relaxation. In fact if the adversary gets B bits of leakage (the entire reduction buffer, let's say) then it can successfully convince the prover even in the case B is much smaller than the entropy of the file. The benefits are for the client to require only $O(B)$ bits of internal memory in order to process the file, even if the file itself is much larger, and the communication complexity of the protocol polylogarithmically depends on B and not on the size of the file. Halevi et al. [85] suggest that in order to have adequate security, B can be set to a relative high value (e.g., 64 Mbyte).

In the third scheme, the file is first preprocessed by performing a set of reduction and mixing phases over its content. Then the Merkle tree is computed over this preprocessed version of the file. The main benefit of this approach is that the preprocessed file can be computed in a one-pass fashion. The drawback is that security can be argued only for a more restrictive set of input distributions, and only under a (reasonable) assumption about the linear code obtained. Specifically, these distributions capture cases in which there are parts of the file that the attacker may know and other parts that are essentially random.

6.2.2 Client-side efficient solutions

To enhance client-side efficiency, Di Pietro et al. [87] proposed a PoW scheme based on a challenge/response mechanism. Every time a file is uploaded to the server, the server computes a set of challenges for that file and stores them. Each challenge is a seed of a pseudo-random function (PRF), used as compact source of randomness, that is expanded to a number of bit positions in the file, and the response is the concatenation of the bit values at the requested positions. Clients prove knowledge of a file by returning the correct string. This solution improves the one presented by Halevi et al. in terms of efficiency and bandwidth consumption at the client side, while requiring more computation at the server side (challenges have to be recomputed when they are exhausted). A similar proposal, but considering full file blocks, is proposed in a US patent [88]. Every time a user wants to prove ownership of a file, the server requests the content of a specific block of the file. In

order to prove the ownership of the file, the client must send the whole block to the server. If they match, the server considers that the client owns the file. After each request by a client, the server refreshes the indexes to ask for.

6.2.3 Server-side, client-side balanced solutions

Later, Blasco et al. [89] proposed an alternative design that strikes a balance between server-side efficiency and client-side efficiency. The strategy consists of using a Bloom filter (BF). a space-efficient randomized data structure, to store the server-side data-structure used in the Di Pietro et al. [87] scheme. Bloom filters [6] are probabilistic data-structures used to represent sets of elements and to perform membership queries over them. The main advantage of Bloom filters is their memory and time efficiency, since they require less space than other data structures to store elements in the set, and less time to perform membership queries. The additional efficiency is traded for accuracy, since an element that is not in the set may be recognized as being part of it (false positives). False negatives, on the other hand, cannot occur by construction.

Informally, the scheme has two separate phases: in the initialization phase, the server receives a file for the first time, initializes a Bloom filter, and splits the input file into chunks of equal size. Each chunk is used to generate a token, which is in turn used to seed a PRF; the PRF is evaluated on the chunk index and the output is inserted into the Bloom filter. In the challenge phase, the server requests the client to upload a number of tokens to prove its knowledge of the file. In particular, the server generates an array of J randomly chosen chunk indexes and sends it to the client. The client computes the token for each of the J chunk indexes and sends all tokens to the server. The server then uses each token to seed the PRF, invokes the PRF on the corresponding chunk index and checks whether each output string belongs to the Bloom filter: if all do, the server considers the PoW run successful.

Blasco et al. approach is more efficient at the client side than the solution proposed by Halevi et al. [85] and more efficient at the server side than the proposal by Di Pietro et al. [87], the size (in bits) of the Bloom filter does not depend on the file size.

6.2.4 Privacy preserving solutions

Halevi et al. [85]'s formulation does not address privacy protection of user data against the storage provider. Ng et al. [90] made the first attempt towards achieving privacy protection. In their approach, files are encrypted on the client side and the encryption keys are shared among a group of users, who know each other, applying existing schemes to do key management within the group. Proofs of ownership are then devised in a privacy preserving manner. Informally, their proof of ownership scheme works as follows: A file is divided into many blocks, and a commitment is computed for each of these blocks under a secret key. Then a Merkle Tree is computed over the commitments. At the end of Merkle Tree proof protocol, the verifier knows some commitment value, and the prover has to show that it has the knowledge of some secret value corresponding to that commitment, without revealing any information about the secret value to the verifier. Unfortunately, their solution is unsatisfactory [91] because privacy is formulated as a local property of each block. Therefore, their scheme suffers from *divide and conquer* attacks.

In [92], Xu and Zhou propose a generic and conceptually simple paradigm to construct PoW by revisting Halevi et al. [11]'s formulation and extending it in two aspects: (1) They shift a significant amount of workload from the server to the client. The rationale behind this is that the authors assume that the average computation power allocated to each online user by the server is typically smaller than the computation power of an average user. (2) They protect data privacy against the verifier (e.g. the server), during the interactive proof protocol. The scheme combines randomness extraction with proofs of retrievability [93] in a three players setting. In the original framework of Halevi et al. [85], the PoW protocol is a two-party protocol between a prover and a verifier. Xu and Zhou introduce a third player, called summarizer (in general, the first uploader), who is responsible to preprocess the data file during setup. The protocol works in the following way: The summarizer (e.g. data owner) runs the summary function S given the input of a security parameter λ and a file F. The output ψ is given to the verifier that can use it to run the two-party PoW protocol with the prover. Intuitively, a PoW scheme is privacy-preserving against the verifier, if everything about the file that the verifier can learn after

participating in the PoW protocol w.r.t. the file, can be computed from the short summary value of the file and some almost-perfect uniform random number. The security definition does not protect against brute force attacks from the verifier that can impersonate a verifier and test if ψ corresponds to a certain file. Moreover, the proposed scheme does not consider bandwidth and server space efficiency.

6.3 PoW and (secure) deduplication

PoW protocols typically focus on a scenario where multiple cloud users outsource unencrypted content. Therefore, as long as data is unencrypted, PoW and deduplication can coexist.

Nevertheless, as required by WP33-R2, TREDISEC will also investigate PoW protocols that can tolerate encrypted data as input. With this respect, PoW protocol can be deployed together with convergent encryption (or message-locked encryption). Informally, the owner of file F who firstly uploads F to the cloud storage, runs the summary function $S(F,\lambda)$ on input file F to generate a summary value ψ, and encrypts F using CE or MLE to generate the corresponding ciphertext. Then the first uploader sends both the ciphertext and the summary value ψ to the cloud storage server. Later, any other owner of F who tries to upload the file, will be asked to engage in the PoW protocol by running the prover algorithm on the CE or MLE version of F and interacts with the cloud storage server as usual in PoW. Recall that, in convergent encryption or message-locked encryption, the encryption key is deterministically generated from the plaintext, and the encryption method is deterministic.

In DupLESS [6], PoW can be easily integrated as describe above. However, it is not clear then how to prevent the assisting server to help those parties who are not in possession of the file for which they are requesting the corresponding secret key. Recall that, in DupLESS, is sufficient to know the hash of the file in order to contact the assisting server and obtaining the corresponding secret key.

Xu et al.'s result [92] can also be combined with CE or MLE, in order to construct strong leakage-resilient client-side deduplication scheme for encrypted data in cloud storage and thus protect data privacy against both outside adversary and curious cloud server. Nevertheless, their security definition does not protect against brute force attacks from the verifier that can impersonate a verifier and test if ψ corresponds to a certain file.

Finally, Gonzalez-Manzano and Orfila [94] elaborating on the work of Di Pietro et al. [87] proposes a PoW that incorporates directly a variation of CE and is resilient to honest-but-curious servers and poisoning attacks, secure in the Halevi et al. model [85].

In Liu et al. [8] proposal, no CE or MLE is used but rather a combination of a novel key-distribution mechanism together with standard semantic-secure encryption. It remains open the question of how to integrate PoW in their setting.

6.4 Summary

Proofs of ownership allows a client to efficiently prove to a server that she indeed holds a file, rather than just some short information about it. Cloud service providers may be interested in having PoW mechanisms in place to avoid, as seen above, various forms of service abuse as in the case of the CDN attack. At the same time, PoW does not break completely the connection between a file (or data block) and its corresponding short identifier. Alice can still perform the brute-force attack on Bob's payslip in order to learn Bob's salary. On the other hand, cloud users could be discouraged by using a cloud service provider that employs PoW if this costs too much for them in terms of resources to put in place to successfully run the protocol. Efficiency is a main concern here and is addressed by WP33-R1.

We have seen in the previous sections that the storage of a single (possible compressed) copy of data for several users is compatible with the concept of proof of ownership (WP33-R4). At the same time, only the scheme by Gonzalez-Manzano and Orfila [94] come with a unified analysis of PoW in the context of secure data deduplication (WP33-R2). A deeper investigation on how easy it is to integrate PoW in other protocols is required. For instance, integrating PoW in the scheme by Liu et al.

[8], that is based neither on CE nor on MLE, requires that the PoW protocol does not leak any information to the untrusted cloud provider through which all message exchanges by the cloud users are routed.

Regarding WP33-R1 (Efficient ownership verification), Table 1 contains a complexity analysis of the state-of-the-art schemes presented above. The table tells us that from a server-side point of view, Halevi et al. [85] is the most efficient one, but the Gonzalez-Manzano et al. [94] scheme guarantees a better trade-off between client and server. There is still room for improvements to have a scheme with the same server-side performance of the Halevi et al.'s scheme and better client performance.

	Halevi et al. [85]	Di Pietro et al. [87]	Blasco et al. [89]	Gonzalez-Manzano et al. [94]
Client computation	$O(\|F\|) \cdot hash$	$O(\|F\|) \cdot hash$	$O(\|F\|) \cdot hash$	$O(B) \cdot CE \cdot hash \cdot hash$
Client I/O	$O(\|F\|)$	$O(\|F\|)$	$O(\|F\|)$	$O(\|F\|)$
Server init computation	$O(\|F\|) \cdot hash$	$O(\|F\|) \cdot hash$	$O(\|F\|) \cdot hash$	$O(B) \cdot hash \cdot hash$
Server regular computation	$O(1)$	$O(n \cdot \lambda) \cdot PRF$	$O\left(\frac{l \cdot \lambda \cdot \log 1/p_f}{p_f}\right)$	$O(n \cdot l \cdot \lambda) \cdot PRNG$
Server init I/O	$O(\|F\|)$	$O(\|F\|)$	$O(\|F\|)$	$O(\|F\|)$
Server regular I/O	$O(0)$	$O(n \cdot \lambda)$	$O(0)$	$O(0)$
Server memory usage	$O(1)$	$O(n \cdot \lambda)$	$O\left(\frac{\log 1/p_f}{l}\right)$	$O(l \cdot n \cdot \lambda)$
Bandwidth	$O(\lambda \cdot log\ \lambda)$	$O(\lambda)$	$O\left(\frac{l \cdot \lambda}{p_f}\right)$	$O(l \cdot \lambda)$

Table 1: Complexity analysis of the state-of-the-art PoW schemes.

In Table 1, λ is the security parameter, n is the number of precomputed challenges in [87], l is the PRF output size, p_f is the false positive rate of the BF, B is the chunk size.

Finally, in Section 3.2, a new explicit requirement has been put forth. This requirement calls for the compatibility of PoW with secure deletion of data and can be considered covered, to some extent, by WP33-R5, that is related to verifiable ownership with dynamicity. Unfortunately, the literature does not provide any solution for this which remains an open problem to be addressed.

7 Conclusions

This report has identified the specific verifiability requirements of TREDISEC uses cases and analysed the compatibility of existing verifiability solutions with data reduction techniques.

In particular, verifiable storage and verifiable computation, among the three verifiability mechanisms analysed in this report, are the ones that have more compatibility issues with data reduction techniques.

Regarding verifiable storage (Section 4), while naturally data owners may push for the implementation of PoR mechanisms, cloud service providers may not be as enthusiastic. Indeed, proofs of retrievability, sentinel-based or tag-based, hinder the financial advantages that the cloud service providers glean from using data reduction techniques. In multi-tenant environment, the situation is even worse because current PoR solutions can significantly decrease the efficiency of the cloud storage services.

Better conclusions can be drawn for verifiable computation (Section 5). As long as data is not encrypted, verifiable computation mechanisms and data reduction techniques can coexist. When data confidentiality is taken into account then most of the verifiable computation solutions do not apply directly over encrypted data and when they do, data is encrypted with homomorphic encryption that is not directly compatible with data deduplication.

In a better position are certainly verifiable ownership solutions (Section 6). Despite the fact that PoW protocols typically focus on a scenario where multiple cloud users outsource unencrypted content, in some cases, they can also be employed in settings where CE or MLE is used. Nevertheless, the integration is not always straightforward. In some settings additional security properties might be required.

To wrap up, despite the relevance and novelty of many existing verifiability mechanisms, in most of the cases, they do not take into account the complexities of integrating verifiability solutions within a cloud environment. This shows the importance of designing verifiability mechanisms that are compatible with data reduction techniques that are critical enablers for a number of popular and successful cloud storage services.

8 References

[1] D. Vallejo García, B. G. Nicasio Crespo, R. M. Vieira Alvarez, H. Ritzdorf , K. Elkhiyaoui, M. Önen, D. Vasilopoulos, P. Louridas , A. De Caro, A. Kurmus, A. Sorniotti, R. Lescuyer, G. Karame, W. Li, A. Fischer, B. Fuhry and M. Kohler , "D2.1 Requirement analysis and consolidation," in *TREDISEC project*, 2015.

[2] C. Soriente, K. Doka, M. Kohler, J. F. Ruiz, R. M. Vieira, B. Gallego-Nicasio, J. Bringer, R. Lescuyer and D. Vallejo Garcia, "D2.1 Description of the context scenarios and use case definition," TREDISEC project, 2015.

[3] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," in *ICDCS*, 2002, pp. 617--624.

[4] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage}," in *Proceedings of the {FAST} '02 Conference on File and Storage Technologies*, Monterey, California, USA, 2002.

[5] M. Bellare, S. Keelveedhi and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," in *Advances in Cryptology - {EUROCRYPT} 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Athens, Greece, 2013.

[6] S. Keelveedhi, M. Bellare and T. Ristenpart, "DupLESS: Server-Aided Encryption for Deduplicated Storage," in *Proceedings of the 22th {USENIX} Security Symposium, Washington, DC,*, Washington, DC, USA, 2013.

[7] J. Stanek, A. Sorniotti, E. Androulaki and L. Kencl, "A Secure Data Deduplication Scheme for Cloud Storage," in *Financial Cryptography and Data Security - 18th International Conference FC 2014*, Christ Church, Barbados, 2014.

[8] J. Liu, N. Asokan and B. Pinkas, "Secure Deduplication of Encrypted Data without Additional Independent Servers," in *Proceedings of the 22nd {ACM} {SIGSAC} Conference on Computer and Communications Security*, Denver, CO, USA, 2015.

[9] F. Armknecht, J.-M. Bohli, G. O. Karame and F. Youssef, "Transparent Data Deduplication in the Cloud," in *Proceedings of the 22nd {ACM} {SIGSAC} Conference on Computer and Communications Security*, Denver, CO, USA, 2015.

[10] P. Puzio, R. Molva, M. Onen and S. Loureiro, "PerfectDedup: Secure data deduplication," in *DPM 2015, 10th International Workshop on Data Privacy Management*, Vienna, Austria, 2015.

[11] D. Klinic, C. Hazary, A. Jagmohan, H. Krawczyk and T. Rabit, *On Compression of Data Encrypted with Block Ciphers,* 2010.

[12] A. K. Lenstra and E. R. Verheul, "Key Improvements to XTR," in *Advances in Cryptology - {ASIACRYPT} 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security*, Kyoto, Japan, 2000.

[13] K. Rubin and A. Silverberg, "Torus-Based Cryptography," in *Advances in Cryptology - {CRYPTO} 2003, 23rd Annual International Cryptology Conference*, Santa Barbara, California, USA, 2003.

[14] C. Gentry, "How to Compress Rabin Ciphertexts and Signatures (and More)," in *Advances in Cryptology - {CRYPTO} 2004, 24th Annual International Cryptology Conference*, Santa Barbara, California, USA, 2004.

[15] M. Naehring, K. Lauter and V. Vaikuntanathan, "Can Homomorphic Encryption Be Practical?," in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, Chicago, Illinois, USA, 2011.

[16] Y. Deswarte and J.-J. Quisquater, "Remote integrity checking," *Integrity and Internal Control in Information Systems,* no. IV, pp. 1-11, 2004.

[17] G. Filho, D. Luiz and P. S. Liccia, "Demonstrating data possession and uncheatable data transfer," in *IARC Crytpology ePrint Archive*, 2006.

[18] F. Sebe, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte and J.-J. Quisquarter, "Efficient remote data possession checking in critical information infrastructures," *IEEE Transactions on Knowledge and Data Engineering,* pp. 1034-1038, 2008.

[19] M. A. Shah, M. Baker, J. C. Mogul and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," in *HotOS*, 2007.

[20] A. Jules and B. S. J. Kaliski, "PORs: Proofs of Retrievability for Large Files," in *Proceedinngs of the 14th ACM conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2007.

[21] E. Stefanof, M. Van Dijk, A. Oprea and A. Juels, "Iris: A Scalable Cloud File System with Efficient Intagrity Checks," *Annual Computer Security Applications Conference-ACASAC,* pp. 229-238, 2012.

[22] A. Juels and A. Oprea, "New Approaches to Security and Availability for Cloud Data," *Communcations of the ACM-CACM,* vol. 2, no. 56, pp. 64-73, February 2013.

[23] M. Azraoui, K. Elkhiyaoui, R. Molva and M. Onen, "Stealthguard: Proof of retrievability with hidden watchdogs," in *Computer Security-ESORICS 2014*, 2014.

[24] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson and D. Song, "Provable data possession at untrusted stores," *Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007,* pp. 598-609, 2007.

[25] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Informartion and System Security (TISSEC),* vol. 14, 2011.

[26] G. Ateniese, R. Di Pietro, L. Mancini and G. Tsudik, "Scalable and efficient provable data possession," *Proceedings of the 14th International Conference on Security and Privacy in Communication Networks, SecureComm 2008,* pp. 9:1-9:10, 2008.

[27] C. Erway, A. Kurcu, C. Papamanthou and R. Tamassia, "Dynamic procvable data possession," *Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009,* pp. 213-222, 2009.

[28] B. Chen and R. Curtmola, "Robust dynamic provable data possession," in *32nd International Conference on Distributed Computing Systems Workshops (ICSCSW)*, 2012.

[29] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *ASIACRYPT 2008,* vol. 5350, no. LNCS, pp. 90-107, 2008.

[30] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability in cloud storage," in *IEEE Transactions on Parallel and Distributed Systems*, 2011.

[31] Y. Dodis, S. P. Vadhan and D. Wichs, "Proofs of Retrievability via Hardness Amplification," *Theory of Cryptography, Sringer,* pp. 109-127, 2009.

[32] D. Cash, A. Kupcu and D. Wich, "Dynamic Proofs of Retrievabikity via Oblivious RAM," *Advances in Cryptology-EUROCRYPT 2013, Sringer,* pp. 279-295, 2013.

[33] E. Shi, E. Stefanov and C. Papamanthou, "Practical dynamic proofs of retrievability," *ACM Conference on Computer and Comunications Security,* pp. 325-336, 2013.

[34] Y. Ren, J. W. J. Xu and J.-U. Kim, "Designated-verifier provable data possession in public cloud storage," *International Journal of Security and Its Applications,* vol. 7, no. 6, pp. 11-20, 2013.

[35] S.-T. Shen and W.-G. Tzeng, "Delegable provable data possession for remote data in the clouds," in *ICICS*, 2011.

[36] Q. Wang, C. Wang, J. Li, K. Ren and W. Lou, "Enabling Public Veriafiability and Data Dynamics for Storage Security in Cloud Computing," *ESORICS 2009,* vol. 5789, no. LNCS, pp. 355-370, 2009.

[37] Q. Wang, C. Wang, K. Ren, W. Lou and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems,* pp. 847-859, 2011.

[38] C. Wang, Q. Wang, K. Ren and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," *Proceedings of the 29th Conference on Information Communications-INFOCOM 2010,* pp. 525-533, 2010.

[39] F. Armknecht, J.-M. Bohli, G. Karame, Z. Liu and C. Reuter, "Outsourced ptoofs of retrievability,"

Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, pp. 831-842, 2014.

[40] K. D. Bowers, A. Juels and A. Oprea, "Proofs of retrievability: theory and impementation," in *CCSW '09 Proceedings of the 2009 ACM workshop on Cloud computing security*, Chicago, 2009.

[41] R. Curtmola, R. Khan, O. Burns and G. Ateniese, "MR-PDP: Multiple-Rplica Provable Data Possession," *Proceedings of the 2008 28th International Confernce on Distributed Computing Systems, ICDCS 1008,* pp. 411-420, 2008.

[42] K. Bowers, A. Juels and A. Oprea, "HAIL: a High-availabiltiy and Integrity Layer for Cloud Storage," *Proceedings of the 16th ACM Conference on Computer and Communications Security, CSS 2009,* pp. 187-198, 2009.

[43] K. D. Bowers, M. Van Dijk, A. Juels, A. Oprea and R. Rivest, "How to tell if you cloud files are vulnerable to drive crashes.," *ACM Conference on Communications Security,* pp. 501-514, 2011.

[44] Z. Peterson, M. Gondree and R. Beverly, "A position paper on data sovereignty: The importance of geolocating data in the cloud," in *3rd USENIX Conference on Hot Topics in Cloud Computing*, Berkeley, CA, USA, 2011.

[45] G. Watson, R. Safavi-Naini, M. Alimomeni, M. Locasto and S. Narayan, "Lost: Location based storage," in *CCSW*, 2012.

[46] S. Goldwasser, S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM J. Comput.,* vol. 18, no. 1, pp. 186-208, 1989.

[47] S. Arora and S. Safra, "Probabilistic Checking of Proofs: A New Characterization of NP," *J. ACM,* vol. 45, no. 1, pp. 70-122, 1998.

[48] J. Kilian, "Improved Efficient Arguments," in *Advances in Cryptology, CRYPTO'95*, 1995.

[49] S. Goldwasser, Y. T. Kalai and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *ACM Symposium on Theory of Computing STOC*, 2008.

[50] S. Micali, "Computationally Sound Proofs," *SIAM J. Comput.,* vol. 30, no. 4, pp. 1253-1298, 2000.

[51] N. Bitansky, R. Canetti, A. Chiesa and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Innovations in Theoretical Computer Science 2012*, 2012.

[52] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *ACM Symposium on Theory of Computing, STOC'11*, 2011.

[53] R. Gennaro, C. Gentry, B. Parno and M. Raykova, "Quadratic Span Programs and Succinct NIZKs without PCPs," in *Advances in Cryptology - EUROCRYPT'13*, 2013.

[54] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky and O. Paneth, "Succinct Non-interactive Arguments via Linear Interactive Proofs," in *Theory of Cryptography Conference, TCC'13*, 2013.

[55] H. Lipmaa, "Succinct Non-Interactive Zero Knowledge Arguments from Span Programs and Linear Error-Correcting Codes," in *Advances in Cryptology, ASIACRYPT'13, Part I*, 2013.

[56] K.-M. Chung, Y. T. Kalai and S. P. Vadhan, "Improved Delegation of Computation Using Fully Homomorphic Encryption," in *Advances in Cryptology - CRYPTO*, 2010.

[57] R. Gennaro, C. Gentry and B. Parno, "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers," in *Advances in Cryptology - CRYPTO*, 2010.

[58] B. Parno, M. Raykova and V. Vaikuntanathan, "How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption," in *Theory of Cryptography - TCC'12*, 2012.

[59] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them: from theoretical possibility to near-practicality," *Electronic Colloquium on Computational Complexity (ECCC),* vol. 20, p. 165, 2013.

[60] G. Cormode, M. Mitzenmacher and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Innovations in Theoretical Computer Science 2012*, 2012.

[61] J. Thaler, "Time-Optimal Interactive Proofs for Circuit Evaluation," in *Advances in Cryptology - CRYPTO (2)*, 2013.

[62] V. Vu, S. T. V. Setty, A. J. Blumberg and M. Walfish, "A Hybrid Architecture for Interactive Verifiable Computation," in *IEEE Symposium on Security and Privacy - SP'13*, 2013.

[63] S. T. V. Setty, R. McPherson, A. J. Blumberg and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *Network and Distributed System Security Symposium, NDSS*, 2012.

[64] S. T. V. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg and M. Walfish, "Taking Proof-Based Verified Computation a Few Steps Closer to Practicality," in *Proceedings of the 21th USENIX Security Symposium*, 2012.

[65] S. T. V. Setty, B. Braun, V. Vu, A. J. a. P. B. Blumberg and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation," in *Eighth Eurosys Conference 2013, EuroSys*, 2013.

[66] B. Parno, J. Howell, C. Gentry and M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," in *IEEE Symposium on Security and Privacy - SP'13*, 2013.

[67] C. Costello, C. Fournet, J. Howell, M. a. K. B. Kohlweiss, M. Naehrig, B. Parno and S. Zahur, "Geppetto: Versatile Verifiable Computation," in *IEEE Symposium on Security and Privacy, SP'15*, 2015.

[68] B. Braun, A. J. Feldman, Z. Ren, S. T. V. a. B. A. J. Setty and M. Walfish, "Verifying computations with state," in *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP*, 2013.

[69] E. Ben-Sasson, A. Chiesa, E. Tromer and M. Virza, "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture," in *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[70] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer and M. Virza, "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge," in *CRYPTO (2)*, 2013.

[71] R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation," in *NDSS'15*, 2015.

[72] M. Backes, D. Fiore and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *2013 ACM Conference on Computer and Communications Security CCS*, 2013.

[73] D. Catalano, D. Fiore and B. Warinschi, "Homomorphic Signatures with Efficient Verification for Polynomial Functions," in *Advances in Cryptology - CRYPTO (1)*, 2014.

[74] S. Gorbunov, V. Vaikuntanathan and D. Wichs, "Leveled Fully Homomorphic Signatures from Standard Lattices," in *ACM Symposium on Theory of Computing, STOC'15*, 2015.

[75] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *ACM Conference on Computer and Communications Security, CCS*, 2012.

[76] S. Benabbas, R. Gennaro and Y. Vahlis, "Verifiable Delegation of Computation over Large Datasets," in *Advances in Cryptology - CRYPTO'11*, 2011.

[77] L. F. Zhang and R. Safavi-Naini, "Verifiable Delegation of Computations with Storage-Verification Trade-off," in *Computer Security, ESORICS'14, Part (1)*, 2014.

[78] C. Papamanthou, R. Tamassia and N. Triandopoulos, "Authenticated hash tables," in *ACM Conference on Computer and Communications Security, CCS'08*, 2008.

[79] C. Papamanthou, R. Tamassia and N. Triandopoulos, "Optimal Verification of Operations on Dynamic Sets," in *Advances in Cryptology - CRYPTO'11*, 2011.

[80] R. Canetti, O. Paneth, D. Papadopoulos and N. Triandopoulos, "Verifiable Set Operations over Outsourced Databases," in *Public-Key Cryptography - PKC'14*, 2014.

[81] R. B. Lee, P. C. S. Kwan, J. P. McGregor and J. S. a. W. Z. Dwoskin, "Architecture for Protecting Critical Secrets in Microprocessors," in *International Symposium on Computer Architecture, ISCA'05), 4-8 June 2005, Madison, Wisconsin, {USA}*, 2005.

[82] D. Lie, C. A. Thekkath, M. Mitchell, P. a. B. D. Lincoln, J. C. Mitchell and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS'00*, 2000.

[83] A.-R. Sadeghi, T. Schneider and M. Winandy, "Token-Based Cloud Computing," in *Trust and*

Trustworthy Computing, TRUST'10, 2010.

[84] A. Seshadri, M. Luk, E. Shi, A. Perrig and L. a. K. P. K. van Doorn, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," in *Symposium on Operating Systems Principles, SOSP'05*, 2005.

[85] S. Halevi, D. Harnik, B. Pinkas and A. Shlman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th {ACM} Conference on Computer and Communications*, Chicago, Illinois, USA, 2011 .

[86] R. Merkle, "A Certified Digital Signature," in *Advances in Cryptology - {CRYPTO} '89, 9th Annual International Cryptology*, Santa Barbara, California, USA, 1989.

[87] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *7th {ACM} Symposium on Information, Compuer and Communications Security*, Seoul, Korea, 2012.

[88] A. Juels, *Method and system for preventing de-duplication side-channel attacks in cloud storage systems,* Google Patents, 2013.

[89] J. Blasco, R. Di Pietro, A. Orifila and A. Sorniotti, "A tunable proof of ownership scheme for deduplication using Bloom filters," in *{IEEE} Conference on Communications and Network Security, {CNS} 2014*, San Francisco, CA, USA, 2014.

[90] W. K. Ng, Y. Wen and H. Zhu, "Private Data Deduplication Protocols in Cloud Storage," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, New York, NY, USA, 2012.

[91] J. Xu, E.-C. Chang and J. Zhou, "Weak Leakage-resilient Client-side Deduplication of Encrypted Data in Cloud Storage," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, Hangzhou, China, 2013.

[92] J. Xu and J. Zhou, "Leakage Resilient Proofs of Ownership in Cloud Storage, Revisited," in *Applied Cryptography and Network Security - 12th International Conference*, Lausanne, Switzerland, 2014.

[93] A. Juels and B. S. J. Kaliski, "Pors: Proofs of Retrievability for Large Files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2007.

[94] L. Gonzalez-Manzano and A. Orfila, "An efficient confidentiality-preserving Proof of Ownership for deduplication," *J. Network and Computer Applications,* vol. 50, pp. 49--59, 2015.

[95] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber and E. R. Weippl, "Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space," in *20th {USENIX} Security Symposium*, San Francisco, CA, USA, 2011.

[96] W. van der Laan, *Dropship. Open source project.,* 2001.

[97] S. Chen, R. Wang, Z. Want and K. Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2010.