# Efficient Simultaneous Privately and Publicly Verifiable Robust Provable Data Possession from Elliptic Curves[⋆]

Christian Hanser and Daniel Slamanig

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology (TUG), Inffeldgasse 16a, 8010 Graz, Austria
{christian.hanser|daniel.slamanig@tugraz.at}

**Abstract.** When outsourcing large sets of data to the cloud, it is desirable for clients to efficiently check, whether all outsourced data is still retrievable at any later point in time without requiring to download all of it. Provable data possession (PDP)/proofs of retrievability (PoR), for which various constructions exist, are concepts to solve this issue. Interestingly, by now, no PDP/PoR scheme leading to an efficient construction supporting both private and public verifiability simultaneously is known. In particular, this means that up to now all PDP/PoR schemes either allow public or private verifiability exclusively, since different setup procedures and metadata sets are required. However, supporting both variants simultaneously seems interesting, as publicly verifiable schemes are far less efficient than privately verifiable ones. In this paper, we propose the first simultaneous privately and publicly verifiable (robust) PDP protocol, which allows the data owner to use the more efficient private verification and anyone else to run the public verification algorithm. Our construction, which is based on elliptic curves, achieves this, as it uses the same setup procedure and the same metadata set for private and public verifiability. We provide a rigorous security analysis and prove our construction secure in the random oracle model under the assumption that the elliptic curve discrete logarithm problem is intractable. We give detailed comparisons with the most efficient existing approaches for either private or public verifiability with our proposed scheme in terms of storage and communication overhead, as well as computational effort for the client and the server. Our analysis shows that for choices of parameters, which are relevant for practical applications, our construction outperforms all existing privately and publicly verifiable schemes significantly. This means, that even when our construction is used for either private or public verifiability alone, it still outperforms the most efficient constructions known, which is particularly appealing in the public verifiability setting.

**Keywords:** Provable data possession, proofs of retrievability, remote data checking, simultaneous public and private verifiability, outsourced storage, elliptic curves, ECDLP, provable security

## 1 Introduction

Cloud storage is an increasingly popular means for archiving, backup, sharing of data, synchronization of multiple devices and it is also envisioned for future primary storage of (enterprise) data. Despite the advantages of cloud storage being among others ubiquitous access to data, immediate scalability and the pay-per-usage billing model, there are still concerns, which hinder a widespread adoption. These concerns are mainly devoted to missing or inadequate security and privacy related features, requiring customers to fully trust in the integrity of the cloud provider as well as the provider's security practices. Among these issues is the availability of outsourced data. Recent incidents [11] indicate that, despite the assumed high availability guarantees of the cloud, outages occur in practice. One way to mitigate this problem is to introduce redundancy in order to improve availability [24]. Another crucial aspect in the context of availability is to verify whether all outsourced data is still retrievable and intact. A naive solution to this problem would be to download all outsourced data and, thereby, check the completeness from time to time. However, for large data sets this is apparently not feasible. Thus, the concepts of provable data possession (PDP) and proofs

⋆ This paper appears in the proceedings of SECRYPT'13.

of retrievability (PoR) have been introduced. The goal of the aforementioned approaches is that a client can regularly challenge the storage server to provide a proof that assures that the outsourced data is still retrievable without having access to the data itself locally. In contrast to the naive approach, this strategy aims at reducing the communication as well as the computational overhead significantly. Ideally, such proofs are of constant length (independent of the size of the data) and the verification of these proofs requires only a small and constant number of computations at the client. Such schemes are said to support private verifiability, if only the client, who has previously outsourced the data (the data owner), is able to verify proofs of data possession from the storage server using a private key. In contrast, one speaks of public verifiability if the data owner provides additional parameters into the public key, such that any third party is able to request and verify proofs of data possession without the client giving away its private key. Consequently, no third party is able to compute valid verification metadata for the data and, thus, cannot modify outsourced data such that valid proofs can still be given. It should be noted that publicly verifiable PDP schemes in general are far more expensive than privately verifiable schemes.

The first construction tailored for the use within outsourced storage denoted as provable data possession (PDP) was presented in [2,1]. A PDP protocol works by constructing homomorphic verifiable tags (HVTs), which are computed by the client prior to outsourcing the data and are stored as metadata in addition to the data at the server. Typically, the proof then requires the storage server (prover) to prove the knowledge of a linear combination of randomly challenged data blocks. It can be efficiently verified by the client by using compact verification data sent by the verifier, whose size is independent of the data size. Although elegant, this RSA-style construction imposes a rather large computational burden on the verifier (client), i.e., a number of large integer exponentiations and inversions linear in the number of challenged blocks. Independently to PDP, proofs of retrievability (PoR) [18] were introduced, further refined in [7] and generalized as well as studied from a coding theoretic point of view in [13,20]. PORs, in their original sense, pursue another approach, namely, check-values (so called sentinels) are inserted in random positions into the data and then the entire file is encrypted and permuted before outsourcing. A proof amounts to requiring the server to send some of these sentinels to the client, who can then check them locally. While PORs are restricted to a limited number of challenges for given data, PDPs usually support an unlimited number of challenges, which is clearly desirable. We note that by using private information retrieval (PIR) in order to hide the exact positions of the accessed sentinels, one can also realize PoR schemes supporting an unlimited number of challenges. However, this approach is of theoretical interest only, since PIR requires the storage server to access the entire data, which is clearly undesirable, and the computational effort for the server as well as the client renders this approach impractical.

Furthermore, PORs typically employ a coding theoretic approach, i.e., a file is encoded prior to outsourcing, whereas PDPs initially were not concerned with encoding (and thus corrections of minor corruptions), but only to handle the detection of corruptions of larger parts of the outsourced file. While PORs come with an explicit knowledge extraction algorithm $\mathsf{Extract}$ to retrieve the file from a sequence of challenges, PDPs only implicitly require such a knowledge extractor in the course of their security proofs. Therefore, the security guarantees made by the original PDP constructions are slightly weaker than those of a POR. However, we note that in recent works both approaches seem to converge to a single unified approach as it is quite straightforward to combine PDPs with suitable codes and thus obtain robustness against small corruptions as well. What we denote as robust PDP also refers to this converged model and thus also may be seen as a "modern" PoR.

## 1.1 Related Work

In [3], the authors provide a generic construction of PDP protocols from any homomorphic identification protocol. The authors of [22] present a privately verifiable PDP construction from pseudo-random functions in the standard model and a publicly verifiable construction from BLS signatures [6] in the random oracle model. Based on this approach, the authors of [25] introduce a public auditing scheme, which extends the classical publicly verifiable PDP/PoR model with the properties of privacy-preservation and batch auditing. The former means that an auditor (verifier) can not learn anything about the stored data during the auditing process. The latter means that a third party performing the challenges on behalf of several clients is able to batch all single challenges in order to obtain improved performance. Recently, [26] introduced a new privately verifiable PDP protocol based on polynomial commitments in the standard model. Another scheme based on polynomial commitments for public verifiability has been introduced in [27]. There are also constructions for a distributed storage setting, that is, considering multiple storage servers [12,29]. The original PDP setting applies only to static (or append-only) files or only supports limited updates with a bounded number of challenges [4]. Dynamic provable data possession (DPDP), in contrast, extends the PDP model to support provable updates to stored data [14,10,8] including insertions at arbitrary positions, updates on existing blocks, revision control [28], etc.

### 1.2 Contribution

By now no PDP leading to an efficient construction supporting both private and public verifiability simultaneously is known. In particular, this means that by now all PDP either allow public or private verifiability only, since different setup procedures and metadata sets are required. However, supporting both variants simultaneously seems interesting, as publicly verifiable schemes are far less efficient than privately verifiable ones. In this paper, we propose the first simultaneous privately and publicly verifiable (robust) PDP protocol, which allows the data owner to use the more efficient private verification and anyone else to run the public verification algorithm. Our construction, which is based on elliptic curves, achieves this, as it uses the same setup procedure and the same metadata set for private and public verifiability. To the best of our knowledge, this is the only construction supporting both features at the same time. Clearly, a trivial alternative to the feature of providing private und public verifiability in parallel would be to use a privately verifiable PDP protocol resulting in one set of metadata (tags) and a second publicly verifiable PDP protocol resulting in a second set of metadata (tags) and to store both metadata sets at the storage site. Then, the data owner could run the protocol on the first set of metadata and all other parties on the second set. However, besides inducing a doubled storage overhead for the metadata, which may be quite significant, this trivial solution suffers from additional deficiencies. Namely, one needs to rely on different PDP schemes likely requiring a different setting, e.g., the used groups, and providing security under potentially unrelated cryptographic assumptions, the data owner has to maintain more private key material as well as public parameters and the data owner has to run the computation of metadata twice. The latter issue does not only apply to the preprocessing when outsourcing data but also for the recomputation of tags when updating any already stored data. Clearly, this also results in an unnecessary computational overhead for the data owner.

In contrast, our construction relies on a single well-established cryptographic assumption and requires none of the aforementioned overheads. We provide a construction, which supports efficient privately und publicly verifiable robust PDP on the same set of metadata and based on the same setup procedure. Both versions can be shown to be secure in the random oracle model under the assumption that the ECDLP is intractable. Moreover, we give detailed comparisons of the most efficient existing approaches for either private or public verifiability [1,22,26] with our proposed construction in terms of storage and communication overhead as well as computational effort for the client and the server. Our analysis shows that our construction outperforms all existing privately and publicly verifiable schemes significantly. This means, that even when our construction is used for either private or public verifiability alone, it still outperforms the most efficient constructions known, which is particularly appealing in the public verifiability setting.

### 1.3 Outline

Section 2 discusses the mathematical and cryptographic preliminaries. Section 3 introduces the formal model of provable data possession and the corresponding security model. Then, Section 4 details our construction for simultaneous private and public verifiability. In Section 5, we compare our results to related approaches, and, finally, Section 6 concludes the paper and lists open issues for future work.

## 2 Preliminaries

In this section, we give an overview of required mathematical and cryptographic preliminaries.

### 2.1 Elliptic Curves and Pairings

An elliptic curve $E$ over the finite field $\mathbb{F}_q$ is a plane, smooth algebraic curve usually defined by a Weierstrass equation. The set $E(\mathbb{F}_q)$ of points $(x, y) \in \mathbb{F}_q^2$ satisfying this equation plus the point at infinity $\mathcal{O}$, which is the neutral element, forms an additive Abelian group, whereas the group law is determined by the *chord-and-tangent* method [23].

Furthermore, if $G$ is a cyclic group and $p$ a divisor of its group order, then there exists a subgroup of order $p$, which we subsequently denote by $G[p]$.

**Definition 1 (Bilinear Map).** Let $G_1, G_2, G_T$ be three cyclic groups of the same prime order $p$, where $G_1, G_2$ are additive groups and $G_T$ is a multiplicative group. We call the map $e : G_1 \times G_2 \to G_T$ a *bilinear map* or *pairing*, if the following conditions hold:

**Bilinearity:** For all $P_1, P_2 \in G_1$ and $P_1', P_2' \in G_2$ we have:
- $e(P_1 + P_2, P') = e(P_1, P') \cdot e(P_2, P')$ for all $P' \in G_2$,
- $e(P, P_1' + P_2') = e(P, P_1') \cdot e(P, P_2')$ for all $P \in G_1$.

**Non-degeneracy:** If $P$ is a generator of $G_1$ and $P'$ a generator of $G_2$, then $e(P, P')$ is a generator of $G_T$, i.e., $e(P, P') \neq 1_{G_T}$.
**Efficiently computable:** $e$ can be computed efficiently.

If $G_1 = G_2$, then $e$ is called *symmetric* and *asymmetric* otherwise. The former type is also called *Type-1* pairing, whereas in case of the latter we distinguish between *Type-2* and *Type-3* pairings. For Type-2 pairings there is an efficiently computable isomorphism $\Psi : G_2 \to G_1$ [9] and for Type-3 pairings such an efficiently computable isomorphism does not exist. Furthermore, let $G_T = \mathbb{F}_{q^k}^*[p]$, which is an order $p$ subgroup of $\mathbb{F}_{q^k}^*$. Note that $k$, the so called *embedding degree*, is defined as $k = \min\{\ell \in \mathbb{N} : p \mid q^\ell - 1\}$.

**Definition 2 (Elliptic Curve Discrete Logarithm Problem (ECDLP)).** Let $E(\mathbb{F}_q)[p]$ be an elliptic curve group of prime order $p$ generated by $P \in E(\mathbb{F}_q)[p]$. Given elements $P, aP \in E(\mathbb{F}_q)[p]$ compute $a \in \mathbb{Z}_p$.

### 2.2 Erasure Codes

An $(n, k, d)$-*erasure code* is a code that transforms a message of $k$ symbols into a codeword of $n$ symbols, such that the minimum Hamming distance of any two codewords is $d$. In general, this allows to detect up to $d - 1$ and to correct up to $(d - 1)/2$ erroneous symbols per codeword. A standard choice for erasure codes, are Reed-Solomon codes [21], which are based on polynomials over finite fields $\mathbb{F}_q = \mathbb{F}_{p^n}$. For this particular erasure code, we have $n = q - 1$ and a minimum distance of $d = n - k + 1$. Consequently, up to $n - k$ erroneous symbols can be detected and up to $(n - k)/2$ can be corrected.

## 3 Provable Data Possession

The goal of a provable data possession scheme is that a client $C$ can outsource data to some storage server $S$ (typically a cloud provider), then delete the local copy of the data while being able to regularly challenge $S$ to provide a proof that the outsourced data is still retrievable. Ideally, such proofs are of constant length (independent of the size of the data) and the verification of these proofs requires only a small and constant number of computations at $C$. This is achieved by requiring $C$ to compute verification metadata (tags) for the data prior to outsourcing and storing the data together with the tags at $S$. Furthermore, $S$ should not need to access the entire data for generating a proof and, therefore, a probabilistic spot checking approach is used. This means that $C$ challenges $S$ to prove the possession of a randomly sampled subset of data blocks, such that the best strategy $S$ can follow is to store the entire data. Otherwise, $C$ will detect this misbehavior with high probability (see Section 3.1 for a discussion of the choice of parameters). Furthermore, the data is encoded prior to outsourcing to obtain robustness against minor corruptions, which would not be detected by means of spot checking. Subsequently, we give a formal definition of such a provable data possession scheme and in the remainder we denote an outsourced data unit as file.

### 3.1 Spot Checking and Robustness

Spot checking means that the client asks the server to prove the possession of a subset of $c$ randomly sampled file blocks of the entire file. This allows a client to detect, whether the server has corrupted a larger portion of the file. Now, one can ask how the choice of $c$ should be made when a file consists of $\ell$ blocks and that the server has corrupted/deleted $\beta$ blocks. As discussed in [1], the probability $P$ that at least one of $c$ blocks sampled by the client matches one of the blocks corrupted/deleted by the server can be analyzed by an urn experiment and can be shown to be bounded by

$$1 - \left(1 - \frac{\beta}{\ell}\right)^c \leq P \leq 1 - \left(1 - \frac{\beta}{\ell - c + 1}\right)^c.$$

For instance, let us assume that we have a file consisting of $\ell = 10^6$ file blocks (of $t$ elements each) and we assume that the server has corrupted $\beta = 10^3$ of these blocks, i.e., 0.1% of all blocks, then to achieve $P \approx 0.99$ we have to set the challenge size to $c = 4600$.

However, when the server only corrupts a very small fraction of the file , e.g., a single block, this can not be efficiently recognized via spot checking. Therefore, erasure codes can be applied to a file before outsourcing in order to resolve this problem (cf. [1,18] for a discussion). PDP schemes that also take resistance against small corruptions into account, typically by means of erasure codes, are called *robust* PDP schemes [10].

### 3.2 PDP Protocol and Security Model

**Definition 3 (Provable Data Possession Scheme (PDP)).** A PDP scheme is a tuple of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Prove}, \mathsf{Verify})$ so that:

$\mathsf{KeyGen}(\kappa)$**:** This probabilistic algorithm gets the security parameter $\kappa \in \mathbb{N}$ and returns a public and private key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Tag}(\mathsf{pk}, \mathsf{sk}, \mathsf{id}, i, \boldsymbol{m_i})$**:** This deterministic algorithm takes a key pair $(\mathsf{pk}, \mathsf{sk})$, a file identifier $\mathsf{id}$, the index $i$ of the file block $\boldsymbol{m_i}$ as input and returns a verification tag $T_i$.

$\mathsf{Prove}(\mathsf{pk}, \mathcal{M}, \mathcal{T}, \mathcal{C})$**:** This deterministic algorithm gets as input the public key $\mathsf{pk}$, a file $\mathcal{M}$ (whose $\mathsf{id}$ is determined by $\mathcal{C}$), the sequence of corresponding tags $\mathcal{T}$, and the challenge $\mathcal{C}$. It returns a proof of possession $\pi$ for the blocks determined by the challenge $\mathcal{C}$.

$\mathsf{Verify}(\mathsf{pk}, \mathsf{sk}, \mathcal{C}, \pi)$**:** This deterministic algorithm takes as input a key pair $(\mathsf{pk}, \mathsf{sk})$, a challenge $\mathcal{C}$ and a proof of data possession $\pi$. It returns $\mathsf{accept}$ if $\pi$ is a correct proof of possession for the blocks determined by the challenge $\mathcal{C}$ and $\mathsf{reject}$ otherwise.

A PDP scheme is called *correct*, if for any honestly generated proof of possession $\pi$ using honestly generated tags $\mathcal{T}$, the probability that the verify algorithm accepts is 1. Using the definition of a PDP scheme, we can now specify the interaction between a client $C$ and a server $S$ by means of the following generic PDP protocol.

**Definition 4 (Provable Data Possession Protocol).** A PDP protocol is a tuple of interactive polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Store}, \mathsf{Challenge})$ so that:

$\mathsf{Setup}$**:** The client $C$ obtains a key pair $(\mathsf{pk}, \mathsf{sk})$ by running $\mathsf{KeyGen}(\kappa)$, publishes $\mathsf{pk}$ and keeps $\mathsf{sk}$ private.

$\mathsf{Store}$**:** Given a file $\mathcal{F}$ identified by $\mathsf{id}$, encode the file using a suitable erasure code and obtain the file $\mathcal{M}$. Then, divide it into $\ell = n/t$ elements and execute $\mathsf{Tag}(\mathsf{pk}, \mathsf{sk}, \mathsf{id}, i, \boldsymbol{m_i})$ on every file block $\boldsymbol{m_i}$ of $t$ elements in $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_{\lceil \frac{n}{t} \rceil})$. Finally, send $(\mathsf{id}, \mathcal{M}, \mathcal{T})$ to the server $S$.

Challenge: The challenger $V$ (not necessarily the client $C$) generates a challenge $\mathcal{C} = (\mathsf{id}, I, l)$, where $\mathsf{id}$ is the file identifier, $I$ is a subset of block indexes $I \subseteq \{1, \ldots, \lceil \frac{n}{t} \rceil\}$ and $l$ is a randomly chosen coefficient. $V$ sends the challenge $\mathcal{C}$ to $S$ and $S$ runs $\mathsf{Prove}(\mathsf{pk}, \mathcal{M}, \mathcal{T}, \mathcal{C})$ to generate the proof $\pi$. $S$ sends $\pi$ back to $V$ and $V$ checks the proof via $\mathsf{Verify}(\mathsf{pk}, \mathsf{sk}, \mathcal{C}, \pi)$.

We emphasize that in a privately verifiable PDP protocol Store and Challenge can only be run by the data owner, while in a publicly verifiable PDP protocol Challenge can be run by any (third) party and Store only by the data owner.

Now, we state the security for a PDP protocol using a game that captures what we require for this protocol to be secure. Loosely speaking, a server should only be able to provide a valid proof, if it holds all challenged data and corresponding tags and can only forge valid proofs for files he does not possess with at most negligible probability. Our security model adopts the security model of [26].

**Definition 5 (Data Possession Game).** The data possession game is comprised of the following consecutive phases:

**Setup:** The challenger $V$ executes $\mathsf{KeyGen}(\kappa)$, gives $\mathsf{pk}$ to the adversary $B$ and keeps $\mathsf{sk}$ private.

**Query:** The adversary $B$ makes adaptive tagging and verification queries. $B$ can perform tagging queries for potentially different file $\mathsf{id}$'s, i.e., $B$ chooses a file block $\boldsymbol{m_i}$, sends it to the challenger, who returns $T_i$ obtained by running $\mathsf{Tag}(\mathsf{pk}, \mathsf{sk}, \mathsf{id}, i, \boldsymbol{m_i})$. Per file $\mathsf{id}$, $B$ is only allowed to query consecutive file blocks. For each $\mathsf{id}$ the adversary stores these blocks and the sequence of corresponding tags. $B$ is restricted to query only unique $(\mathsf{id}, i)$ pairs.

**Retrieve:** $V$ challenges $B$ $\lambda$ times for some previously queried file $\mathcal{M}^*$ identified by $\mathsf{id}^*$, where the challenged indexes have been queried before and sends it to $B$. $B$ computes the according proofs $\pi_1, \ldots, \pi_\lambda$ for the file $\mathcal{M}^*$ identified by $\mathsf{id}^*$ and challenge $\mathcal{C}_i$ and returns them to $V$. From the file blocks obtained in these proofs, $V$ extracts a file $\mathcal{M}'$ using some PPT knowledge extractor. $B$ wins the game if $\mathcal{M}' \neq \mathcal{M}^*$.

An adversary is called $\epsilon$-*admissible*, if the probability that it is able to convince $V$ to accept a proof in the retrieve phase of the above game is at least $\epsilon$.

Now, we state what constitutes a secure PDP protocol:

**Definition 6 (Secure PDP Protocol).** A PDP protocol (Setup, Store, Challenge) built upon a PDP scheme (KeyGen, Tag, Prove, Verify) guarantees robust provable data possession, if it is correct and if for any $\epsilon$-admissible PPT adversary there is a value $\lambda$ for the number of queries in the retrieve phase, which is bounded by some polynomial in the number of file blocks, such that the probability that $B$ wins the data possession game is negligibly small in the security parameter $\kappa$.

## 4 Construction

In this section, we present our construction for simultaneous private and public verifiability. The intuition behind our protocol in general is that $S$ is required to prove the knowledge of a linear combination of file blocks (indicated by the challenge), where the coefficients are based on a value randomly chosen by the client in each protocol run. This makes storing linear combinations of file blocks instead of file blocks impractical. Along with this linear combination, $S$ aggregates the tags corresponding to the challenged file blocks, which enable verification at $C$ without having access to the actual file blocks.

In the following, we identify each file block with a vector. Therefore, as it is common, we split the file $\mathcal{M} = (m_1, \ldots, m_n)$ represented as elements of $\mathbb{Z}_p$ into $\ell = \frac{n}{t}$ consecutive vectors $\boldsymbol{m_i} = (m_{i,1}, \ldots, m_{i,t})$ for $1 \leq i \leq \ell$ of $t$ subsequent elements of $\mathbb{Z}_p$, where $t$ is a parameter chosen by the user to adjust the storage overhead. We assume that the length $n$ of $\mathcal{M}$ is a multiple of $t$, whereas $\mathcal{M}$ is padded with an appropriate number of elements of the form $0 \in \mathbb{Z}_p$ if this condition is not satisfied. Doing so, we obtain a representation $\mathcal{M}'$ of $\mathcal{M}$ such that

$$\mathcal{M}' = \begin{bmatrix} \boldsymbol{m_1} \\ \vdots \\ \boldsymbol{m_i} \\ \vdots \\ \boldsymbol{m_{\frac{n}{t}}} \end{bmatrix} = \begin{bmatrix} m_{1,1} & \cdots & m_{1,t} \\ \vdots & & \vdots \\ m_{i,1} & \cdots & m_{i,t} \\ \vdots & & \vdots \\ m_{\frac{n}{t},1} & \cdots & m_{\frac{n}{t},t} \end{bmatrix}.$$

For each vector $\boldsymbol{m_i}$, we compute a tag $T_i$, i.e., every tag aggregates $t$ elements of $\mathbb{Z}_p$. We emphasize that the challenge in designing PDP protocols, which aggregate vectors into single tags, is to prevent the storage server from storing the sum of the vectors components instead of all components thereof.

Scheme 4.1 shows the detailed construction of our scheme for simultaneous private and public verifiability, which is used as building block for Protocol 4.1. Note that for the data owner it is considerably cheaper to run the private verification, since it, firstly, does not involve pairing evaluations and, secondly, saves a considerable amount of scalar multiplications and point additions, as the data owner has access to the private key.

KeyGen: On input $\kappa$, choose an elliptic curve $E(\mathbb{F}_q)$ with a subgroup of large prime order $p$ generated by $P \in E(\mathbb{F}_q)[p]$, such that the bitlength of $p$ is $\kappa$. Choose an asymmetric pairing $e : E(\mathbb{F}_q)[p] \times G_2 \to \mathbb{F}^*_{q^k}[p]$ with $G_2$ being a $p$-order elliptic curve subgroup over (an extension of) the field $\mathbb{F}_q$ with generator $P'$, where the choice of $G_2$ depends on the specific instantiation of the pairing. Now, let elements $s_1, s_2, \alpha \in_R \mathbb{Z}_p$, let $Q'_1 = s_1 P'$, $Q'_2 = s_2 P'$, compute $\alpha P, \ldots, \alpha^t P$, choose two cryptographic hash functions $h : \{0,1\}^* \to \mathbb{Z}_p$ and $H : \{0,1\}^* \to E(\mathbb{F}_q)[p]$ and output $\mathsf{pk} = (E(\mathbb{F}_q), G_2, e, p, P, P', Q'_1, Q'_2, \alpha P, \ldots, \alpha^t P, h, H)$ as well as $\mathsf{sk} = (s_1, s_2, \alpha)$.

Tag: Given $\mathsf{pk}, \mathsf{sk}$, a file identifier $\mathsf{id}$, a vector index $i$ and a vector $\boldsymbol{m_i} = (m_{i,j})_{j=1}^t$, compute the corresponding tag as $T_i = (s_1 H(\mathsf{id}\|i) + s_2 h(\mathsf{id}\|i) \sum_{j=1}^t m_{i,j} \alpha^j P)$ and output $T_i$.

Prove: On input $\mathsf{pk}$, $\mathcal{M} = (\boldsymbol{m_1}, \ldots, \boldsymbol{m_{\frac{n}{t}}})$, $\mathcal{T}$ and challenge $\mathcal{C} = (\mathsf{id}, I, l)$, compute

$$\mu = (\mu_j)_{j=1}^t = \big(h(\mathsf{id}\|i) \sum_{i \in I} m_{i,j} l^i\big)_{j=1}^t \quad \text{and } \tau = \sum_{i \in I} l^i T_i,$$

where $m_{i,j}$ is the element with index $(i,j)$ in the representation $\mathcal{M}'$ of $\mathcal{M}$. Return $\pi = (\mu, \tau) \in \mathbb{Z}_p^t \times E(\mathbb{F}_q)$.

Verify$_{\mathsf{Priv}}$: Given $\mathsf{pk}, \mathsf{sk}$, challenge $\mathcal{C}$ and proof $\pi$, check whether the relation

$$s_1 \sum_{i \in I} l^i \cdot H(\mathsf{id}\|i) + (s_2 \sum_{j=1}^t \mu_j \alpha^j) P = \tau$$

holds and return $\mathsf{accept}$ on success and $\mathsf{reject}$ otherwise.

Verify$_{\mathsf{Pub}}$: Given $\mathsf{pk}, \mathsf{sk} = \mathsf{null}$, challenge $\mathcal{C}$ and proof $\pi$, check whether the relation

$$e(\sum_{i \in I} l^i \cdot H(\mathsf{id}\|i), Q'_1) \cdot e(\sum_{j=1}^t \mu_j (\alpha^j P), Q'_2) = e(\tau, P')$$

holds and return $\mathsf{accept}$ on success and $\mathsf{reject}$ otherwise.

**Scheme 4.1:** PDP scheme with simultaneous private and public verifiability.

## 4.1 Security Analysis

For Protocol 4.1 we are able to prove the following statement.

**Theorem 1.** *Assuming the hardness of the ECDLP, Protocol 4.1 guarantees robust provable data possession in the random oracle model.*

The proof of Theorem 1 can be found in Appendix A.

Setup: $C$ runs $\mathsf{KeyGen}(\kappa)$ and obtains the key pair $(\mathsf{pk},\mathsf{sk})$, where $\mathsf{pk} = (E(\mathbb{F}_q), G_2, e, p, P, P', Q'_1, Q'_2, \alpha P, \ldots, \alpha^t P, h, H)$ and $\mathsf{sk} = (s_1, s_2, \alpha)$. $C$ publishes $\mathsf{pk}$ in an authentic way and keeps $\mathsf{sk}$ private.

Store: Apply a Reed-Solomon code [21] to the file $\mathcal{F}$ and obtain an encoded file $\mathcal{M}$. For every vector $\boldsymbol{m_i}$ of $t$ elements in $\mathcal{M}$ identified by $\mathsf{id}$, $C$ invokes $\mathsf{Tag}(\mathsf{pk},\mathsf{sk},\mathsf{id},i,\boldsymbol{m_i})$ to build the sequence of tags $\mathcal{T}$. Then, $C$ sends $(\mathsf{id},\mathcal{M},\mathcal{T})$ to the server $S$ and removes $\mathcal{M}$ and $\mathcal{T}$ locally.

$\mathsf{Challenge}_{\mathsf{Priv}}$: $C$ requests a proof of possession for file $\mathcal{M}$ with identifier $\mathsf{id}$ by spot checking $c$ vectors of $\mathcal{M}$ as follows:

- $C$ picks an index set $I \subseteq \{1, \ldots, \frac{n}{t}\}$ of $c$ elements, a random element $l \in \mathbb{Z}_p$ and sends the challenge $\mathcal{C} = (\mathsf{id}, I, l)$ to the server $S$.
- On receiving $\mathcal{C}$, $S$ runs $\mathsf{Prove}(\mathsf{pk},\mathcal{M},\mathcal{T},\mathcal{C})$ to obtain $\pi$ and sends it to $C$.
- Finally, $C$ runs $\mathsf{Verify}_{\mathsf{Priv}}(\mathsf{pk},\mathsf{sk},\mathcal{C},\pi)$.

$\mathsf{Challenge}_{\mathsf{Pub}}$: $V$ requests a proof of possession for file $\mathcal{M}$ with identifier $\mathsf{id}$ by spot checking $c$ vectors of $\mathcal{M}$ as follows:

- $V$ picks an index set $I \subseteq \{1, \ldots, \frac{n}{t}\}$ of $c$ elements, a random $l \in \mathbb{Z}_p$ and sends the challenge $\mathcal{C} = (\mathsf{id}, I, l)$ to $S$.
- On receiving $\mathcal{C}$, $S$ runs $\mathsf{Prove}(\mathsf{pk},\mathcal{M},\mathcal{T},\mathcal{C})$ to obtain $\pi$ and sends it to $V$.
- Finally, $V$ runs $\mathsf{Verify}_{\mathsf{Pub}}(\mathsf{pk},\mathsf{null},\mathcal{C},\pi)$.

**Protocol 4.1:** PDP protocol with simultaneous private and public verifiability.

## 4.2 On Efficient Implementations

In our construction, we make use of a hash function $H : \{0,1\}^* \to E(\mathbb{F}_q)[p]$, which maps to an elliptic curve group. We note that there are well-known strategies to hash into elliptic curve groups [17]. However, in our concrete scenario, we are able to choose $H$ to be of a particular form, which allows us to obtain very efficient implementations of our construction. In particular, we choose $H$ in such a way that $H(x) = h(0\|x) \cdot P$, whereas $h$ is the cryptographic hash function mapping to the integers modulo the group order used in Scheme 4.1. Note that prepending 0 to the input of $h$ yields a hash function, which is independent from $h$ itself. This is necessary to prevent tags from being malleable.

The above hash function instantiation allows us to simplify Scheme 4.1 as follows:

Tag:

$$T_i = (s_1 h(0\|\mathsf{id}\|i) + s_2 h(\mathsf{id}\|i) \sum_{j=1}^{t} m_{i,j}\alpha^j)P$$

$\mathsf{Verify}_{\mathsf{Priv}}$:

$$(s_1 \sum_{i\in I} h(0\|\mathsf{id}\|i)l^i + s_2 \sum_{j=1}^{t} \mu_j \alpha^j)P = \tau$$

$\mathsf{Verify}_{\mathsf{Pub}}$:

$$e((\sum_{i\in I} h(0\|\mathsf{id}\|i)l^i)P, Q'_1) \cdot e(\sum_{j=1}^{t} \mu_j(\alpha^j P), Q'_2) = e(\tau, P')$$

| Operation | Semantics | Operand Size | Description |
|---|---|---|---|
| P | $e(P_1,P'_2)$ | 224 | Pairing computation |
| E | $b^d$ | 2048 | Large integer exponentiation |
| S | $d{\cdot}P$ | 224 | Scalar multiplication |
| A | $P_1{+}P_2$ | 224 | Point addition |
| I | $b^{-1} \pmod N$ | 2048 | Large integer modular inversion |
| M | $b_1{\cdot}b_2$ | 2048 | Large integer multiplication |
| i | $b^{-1}$ | 224 | Field inversion |
| m | $b_1{\cdot}b_2$ | 224 | Field multiplication |
| H | $H(m)$ | 224 | Hash or PRF function evaluation |

**Table 1.** Symbols for costs of arithmetical operations.

As one can see, this allows us to trade expensive elliptic curve scalar multiplications for inexpensive field multiplications in $\mathbb{Z}_p$. Furthermore, using Horner's method for the polynomial evaluations, the number of field multiplications in the algorithms $\mathsf{Prove}$, $\mathsf{Verify}_{\mathsf{Priv}}$ and $\mathsf{Verify}_{\mathsf{Pub}}$ can be kept at a minimum. Moreover, note that the algorithms $\mathsf{Prove}$ and $\mathsf{Verify}_{\mathsf{Pub}}$ are well-suited for the application of simultaneous multiple point multiplication [16], which improves their computational efficiency considerably.

Notice, that we can use such an instantiation of the hash function $H$ without sacrificing the security of the overall construction, as we incorporate the random and unknown value $s_1$ in the computation of the tags. An implication of this particular choice of $H$ is that $e$ needs to be a Type-3 pairing, in order to prevent $Q_1'$ and $Q_2'$ to be mapped to the group $E(\mathbb{F}_q)[p]$, as, otherwise, the tag construction is no longer secure. Nevertheless, Type-3 pairings are the best choice from a security and performance perspective [9].

Finally, we emphasize that after applying these optimizations, the data owner still benefits significantly from using the private verification relation, which will be clear from the analysis in Section 5.

| **Scheme** | **Key Size** | **Tagging** | **Server** | **Client** |
|---|---|---|---|---|
| Private Verifiability | | | | |
| S-PDP [1] | $\kappa$=2048 | $\ell(2\kappa\mathsf{E}+2\mathsf{M}+\mathsf{H})$ | $(2ct+c)\mathsf{E}+2(c-1)\mathsf{M}+\mathsf{H}$ | $(c+2)\mathsf{E}+\mathsf{I}+c\mathsf{M}+(c+1)\mathsf{H}$ |
| SPOR [22] | $\kappa$=2048 | $\ell t\mathsf{M}+(\ell+1)\mathsf{H}$ | $c(t+1)\mathsf{M}$ | $(c+t)\mathsf{M}+(c+1)\mathsf{H}$ |
| EPOR [26] | $\kappa$=224 | $\ell(t+1)\mathsf{m}+\ell\mathsf{H}$ | $(t-1)(\mathsf{S}+\mathsf{A})+(ct+c+t)\mathsf{m}$ | $2\mathsf{S}+\mathsf{i}+(c+1)\mathsf{m}+c\mathsf{H}$ |
| Scheme 4.1 | $\kappa$=224 | $\ell(\mathsf{S}+(t+3)\mathsf{m}+2\mathsf{H})$ | $c\mathsf{S}+(c-1)\mathsf{A}+c(t+2)\mathsf{m}+c\mathsf{H}$ | $\mathsf{S}+(c+t+2)\mathsf{m}+c\mathsf{H}$ |
| Public Verifiability | | | | |
| P-PDP [1] | $\kappa$=2048 | $2n(\kappa\mathsf{E}+\mathsf{M}+\mathsf{H})$ | $c\mathsf{E}+2(c-1)\mathsf{M}$ | $(c+2)\mathsf{E}+\mathsf{I}+2(c-1)\mathsf{M}+2c\mathsf{H}$ |
| PPOR [22] | $\kappa$=224 | $\ell((t+1)\mathsf{S}+t\mathsf{A}+\mathsf{H})$ | $c\mathsf{S}+(c-1)\mathsf{A}+ct\mathsf{m}$ | $2\mathsf{P}+(c+t)\mathsf{S}+(c+t-1)\mathsf{A}$ |
| Scheme 4.1 | $\kappa$=224 | $\ell(\mathsf{S}+(t+3)\mathsf{m}+2\mathsf{H})$ | $c\mathsf{S}+(c-1)\mathsf{A}+c(t+2)\mathsf{m}+c\mathsf{H}$ | $3\mathsf{P}+(t+1)\mathsf{S}+(t-1)\mathsf{A}+c\mathsf{m}+c\mathsf{H}$ |

**Table 2.** Comparison of computational complexity of PDP schemes with private and public verifiability.

### 4.3 Remarks

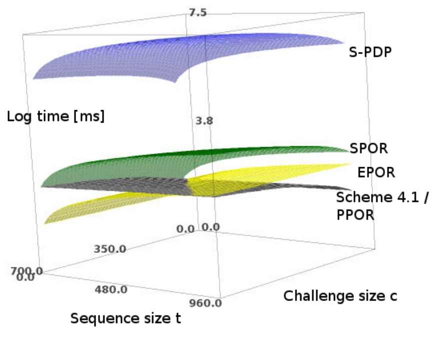
- In our challenge, we have included the index set $I$. For sake of reduced communication bandwidth, it can be generated by the server from a compact seed by using the $(\delta, \gamma)$-hitter construction given by Goldreich [15] or by using pseudo-random functions (PRFs) as in [1].
- We suggest point compression for all transmitted and stored curve points.
- Note that the proposed scheme can be easily adapted to batch challenges [25] over multiple files, which yields a constant communication overhead independent of the number of challenged files.

| **Scheme** | **Key Size** | **Communication Overhead** | **Storage Overhead** |
|---|---|---|---|
| Private Verifiability | | | |
| S-PDP [1] | $\kappa$=2048 | $(c+1)\kappa+h$ | $\ell\kappa$ |
| SPOR [22] | $\kappa$=2048 | $(2t+c+1)\kappa+h$ | $\ell\kappa+t\kappa$ |
| EPOR [26] | $\kappa$=224 | $(c+3)\kappa$ | $\ell\kappa$ |
| Scheme 4.1 | $\kappa$=224 | $(t+1)\kappa$ | $\ell\kappa$ |
| Public Verifiability | | | |
| P-PDP [1] | $\kappa$=2048 | $(c+2)\kappa$ | $\geq n\kappa$ |
| PPOR [22] | $\kappa$=224 | $(2t+c+2)\kappa$ | $\ell\kappa+(t+1)\kappa$ |
| Scheme 4.1 | $\kappa$=224 | $(t+1)\kappa$ | $\ell\kappa$ |

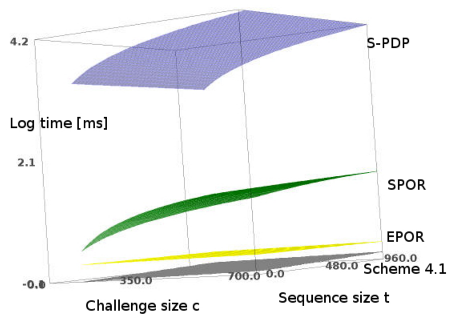
**Table 3.** Comparison of communication and storage overhead of PDP schemes with private and public verifiability.

## 5 Comparative Analysis

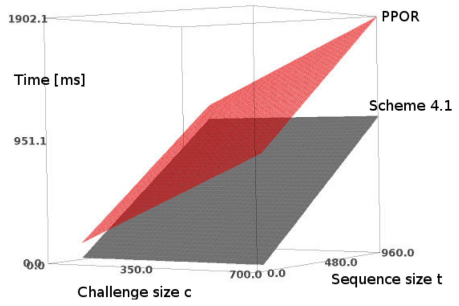
In this section we draw a comparison between existing approaches and our construction in terms of storage and communication overhead as well as computational effort. We point out that existing literature typically uses far too small security parameters for the intended use of provable data possession, i.e., outsourcing large datasets for long-term storage. In particular, all works we are



(a) Computational costs for Prove in the case of public and private verifiability.



(b) Computational costs for Verify in the case of private verifiability.



(c) Computational costs for Verify in the case of public verifiability.

**Fig. 1.** Figure 1(a) illustrates the computational costs of the Prove algorithms of all schemes in Table 2 by varying both parameters $c$ and $t$. Figure 1(b) illustrates the computational costs of the Verify algorithms of all privately verifiable schemes in Table 2 by varying both parameters $c$ and $t$. The time is in logarithmic scale with radix 10. Figure 1(c) illustrates the costs of the Verify algorithms of the publicly verifiable schemes PPOR and Scheme 4.1 in Table 2 by varying both parameters $c$ and $t$. Furthermore, we set $\mathsf{H} = 0$.

aware of suggest parameter sizes of 1024 bits for RSA-based/DL-based approaches and 160 bits for ECDL-based approaches. However, having the long-term characteristic in mind, it is more natural to choose at least 2048 and 224 bits security, respectively, as suggested by NIST in [5]. Subsequently, $\kappa$, $t$ and $\ell$ stand for the security parameter, the number of file elements, which are aggregated into one tag and $\ell = n/t$ the number of file blocks (vectors), respectively. Furthermore, let the challenged index set of file blocks (vectors) $I$ be of size $c$.

## 5.1 Computational Effort

In Table 2, we compare our proposed scheme with existing approaches in terms of computational effort. The symbols for the operands and their respective meanings are illustrated in Table 1.

Figure 1 illustrates the performance comparison of our proposed scheme with existing approaches. We conducted the experiments on an Intel Core i5-2540M equipped with 8GB RAM running Ubuntu 12.10/amd64 and OpenJDK 6/amd64. For the 2048-bit integer arithmetics we use the standard Java$^{\text{TM}}$ BigInteger class. Furthermore, we were using the jPBC library [1] version 1.2.1. We have chosen an MNT curve [19] with a group size of 224 bits and embedding degree $k = 6$ over a prime field and used the Tate pairing in order to perform our benchmarks. In order to guarantee fairness and simplicity for the comparisons illustrated in Figure 1, we have omitted the costs of hash function evaluations in our benchmarks. Figure 1 shows that our scheme is the most efficient scheme for private and public verifiability with respect to server and client computations for reasonable values of the challenge size $c$ and vectors size $t$. It is important to note that an efficient Verify algorithm,

[1] `http://gas.dia.unisa.it/projects/jpbc`

as achieved by our scheme, is the most important aspect with respect to practicality. This is due to the fact that the client can be assumed to be far more resource constraint than the server (the cloud), since the client could, for instance, be a smart phone.

In Table 2, one can see that for the data owner it is considerably cheaper to run the private verification, as the data owner has access to the private key. More precisely, the data owner can trade three pairing evaluations, $t$ scalar multiplications and $t-1$ additions for $t+2$ cheap multiplications in $\mathbb{Z}_p$.

### 5.2 Storage Overhead

In Table 3, we give an analysis of the storage and communication overhead of our scheme compared to existing approaches. In the following, $h$ stands for the output length of a hash function or HMAC of suitable size. As one can see from Table 3, our proposed scheme is as efficient as the most efficient previous schemes, which either support only private or public verifiability with respect to communication and storage overhead. Here, we need to note that when one wants to have private and public verifiability simultaneously than for all other schemes except ours the storage overhead will be the sum of the storage overheads of the respective privately and publicly verifiable PDP schemes.

## 6 Conclusions

In this paper we have presented a novel construction for privately and publicly verifiable robust provable data possession. Our construction is based on elliptic curves and is provable secure in the random oracle model assuming the intractability of the elliptic curve discrete logarithm problem. We have shown that our scheme is the most efficient (robust) scheme with respect to server and client computations for reasonable values of challenge and block size for private as well as public verifiability. To the best of our knowledge our construction is the first to support the use of simultaneous private and public verifiability on the same set of metadata. This means that the data owner can use the more efficient scheme with private verification, while any other party can run the publicly verifiable variant at the same time without having access to the owner's private key. Thereby, both versions use the same parameters as well as metadata (tag) sets.

### 6.1 Future Work

The original PDP setting applies only to static (or append-only) files or only supports limited updates with a bounded number of challenges [4]. Dynamic provable data possession (DPDP), in contrast, extends the PDP model to support provable updates to stored data [14,10,8] including insertions at arbitrary positions, updates on existing blocks, revision control [28], etc. Future work includes investigating our construction in the DPDP model.

## References

1. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Khan, O., Kissner, L., Peterson, Z., Song, D.: Remote data checking using provable data possession. ACM Trans. Inf. Syst. Secur. 14(1), 12:1–12:34 (Jun 2011)
2. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X.: Provable data possession at untrusted stores. In: ACM CCS. pp. 598–609 (2007)
3. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: ASIACRYPT. pp. 319–333 (2009)
4. Ateniese, G., Pietro, R.D., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: SecureComm 2008 (2008)
5. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: NIST SP800-57: Recommendation for Key Management Part 1: General(Revised). Tech. rep. (Mar 2007), http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

6. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: ASIACRYPT. pp. 514–532 (2001)
7. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: CCSW. pp. 43–54 (2009)
8. Cash, D., Küpçü, A., Wichs, D.: Dynamic Proofs of Retrievability via Oblivious RAM. In: EUROCRYPT 2013. LNCS, Springer (2013)
9. Chatterjee, S., Menezes, A.: On cryptographic protocols employing asymmetric pairings - the role of $\psi$ revisited. Discrete Applied Mathematics 159(13), 1311–1322 (2011)
10. Chen, B., Curtmola, R.: Robust dynamic provable data possession. In: ICDCS Workshops. pp. 515–525 (2012)
11. Cloud Outages: `http://www.crn.com/slide-shows/cloud/231000954/the-10-biggest-cloud-outages-of-2011-so-far.htm` (2011)
12. Curtmola, R., Khan, O., Burns, R.C., Ateniese, G.: Mr-pdp: Multiple-replica provable data possession. In: ICDCS 2008. pp. 411–420 (2008)
13. Dodis, Y., Vadhan, S.P., Wichs, D.: Proofs of retrievability via hardness amplification. In: TCC. pp. 109–127 (2009)
14. Erway, C.C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: CCS. pp. 213–222 (2009)
15. Goldreich, O.: A sample of samplers - a computational perspective on sampling (survey). ECCC 4(20) (1997)
16. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2003)
17. Icart, T.: How to hash into elliptic curves. In: CRYPTO. pp. 303–316 (2009)
18. Juels, A., S. Kaliski Jr., B.: Pors: proofs of retrievability for large files. In: ACM CCS. pp. 584–597 (2007)
19. Miyaji, Nakabayashi, Takano: New Explicit Conditions of Elliptic Curve Traces for FR-Reduction. TIE-ICE: IEICE Transactions on Communications/Electronics/Information and Systems (2001)
20. Paterson, M.B., Stinson, D.R., Upadhyay, J.: A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. Cryptology ePrint Archive, Report 2012/611 (2012), `http://eprint.iacr.org/`
21. Reed, I., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8(2), 300–304 (1960)
22. Shacham, H., Waters, B.: Compact proofs of retrievability. In: ASIACRYPT. pp. 90–107 (2008)
23. Silverman, J.: The Arithmetic of Elliptic Curves, Graduate Texts in Mathematics, vol. 106. Springer (1986)
24. Slamanig, D., Hanser, C.: On Cloud Storage and the Cloud of Clouds Approach. In: ICITST-2012. pp. 649 – 655. IEEE (2012)
25. Wang, C., Chow, S.S.M., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. IEEE Trans. Computers 62(2), 362–375 (2013)
26. Xu, J., Chang, E.C.: Towards efficient proofs of retrievability. In: AsiaCCS. ACM (2012)
27. Yuan, J., Yu, S.: Proofs of retrievability with public verifiability and constant communication cost in cloud. In: International Workshop on Security in Cloud Computing. ACM (2013)
28. Zhang, Y., Blanton, M.: Efficient dynamic provable possession of remote data via balanced update trees. In: AsiaCCS. pp. 183–194. ACM (2013)
29. Zhu, Y., Hu, H., Ahn, G.J., Yu, M.: Cooperative provable data possession for integrity verification in multicloud storage. IEEE Trans. Parallel Distrib. Syst. 23(12), 2231–2244 (2012)

## A Proof of Theorem 1

*Proof.* This proof consists of three parts addressing the correctness, the unforgeability of the tags, via a reduction to the ECDLP in $E(\mathbb{F}_q)[p]$, and the retrievability of the file $\mathcal{F}$.

At first, we show the correctness of Scheme 4.1. From the verification relation in $\mathsf{Verify}_{\mathsf{Priv}}$ we get:

$$\begin{aligned}
&s_1 \sum_{i \in I} l^i \cdot H(\mathsf{id}\|i) + (s_2 \sum_{j=1}^{t} \mu_j \alpha^j)P = \\
&s_1 \sum_{i \in I} l^i \cdot H(\mathsf{id}\|i) + (s_2 \sum_{j=1}^{t} \sum_{i \in I} l^i h(\mathsf{id}\|i) m_{i,j} \alpha^j)P = \\
&\sum_{i \in I} l^i \cdot (s_1 H(\mathsf{id}\|i) + s_2 \sum_{j=1}^{t} h(\mathsf{id}\|i) m_{i,j} \alpha^j P) = \sum_{i \in I} l^i T_i = \tau
\end{aligned}$$

Furthermore, from the verification relation in $\mathsf{Verify}_{\mathsf{Pub}}$ we get:

$$\begin{aligned}
&e(\sum_{i \in I} l^i \cdot H(\mathsf{id}\|i), Q_1') \cdot e(\sum_{j=1}^{t} \mu_j (\alpha^j P), Q_2') = \\
&e(s_1 \sum_{i \in I} l^i \cdot H(\mathsf{id}\|i), P') \cdot e(s_2 \sum_{j=1}^{t} \mu_j (\alpha^j P), P') = \\
&e(s_1 \sum_{i \in I} l^i \cdot H(\mathsf{id}\|i) + s_2 \sum_{j=1}^{t} \mu_j (\alpha^j P), P') = \\
&e(s_1 \sum_{i \in I} l^i \cdot H(\mathsf{id}\|i) + s_2 \sum_{j=1}^{t} \sum_{i \in I} l^i h(\mathsf{id}\|i) \cdot m_{i,j} \cdot \alpha^j P, P') = \\
&e(\sum_{i \in I} l^i (s_1 \cdot H(\mathsf{id}\|i) + s_2 \sum_{j=1}^{t} h(\mathsf{id}\|i) \cdot m_{i,j} \cdot \alpha^j P), P') = \\
&e(\sum_{i \in I} l^i T_i, P') = e(\tau, P')
\end{aligned}$$

This demonstrates the correctness of both verification relations.

Secondly, we prove that the tags in our scheme are unforgeable. We do so by showing that any PPT adversary $B$ winning the data possession game for a file not equal to the original file, can be turned into an efficient PPT algorithm $A$ that solves arbitrary instances of the ECDLP in $E(\mathbb{F}_q)[p]$. In the following, we describe how this algorithm simulates the environment of the challenger when interacting with the adversary.

Algorithm $A$ is given an arbitrary instance $(P, R = rP)$ of the ECDLP in $E(\mathbb{F}_q)[p]$. Then, $A$ sets the public and private keys as $\mathsf{pk} = (E(\mathbb{F}_q), G_2, e, p, P, P', Q_1', Q_2', \alpha P, \ldots, \alpha^t P, h, H)$, where $A$ chooses $\alpha^j P = (\phi_j P + \psi_j R)$ with $\phi_j, \psi_j \in_R \mathbb{Z}_p$ for $j = 1, \ldots t$, as well as $\mathsf{sk} = (s_1, s_2)$ and gives $\mathsf{pk}$ to $B$. Note that this choice of the values $\alpha^j P$ in the simulation is indistinguishable from the values chosen in the real game. It will be clear from the simulation of the hash function why we use values $\alpha^j P$ of this particular form.

Furthermore, $A$ simulates the tagging and hash oracle queries for $B$, whereas $B$ is allowed to run the public verification algorithm for all generated tags. Now, if $A$ receives a tagging query for a file block $\boldsymbol{m_i}$ identified by $(\mathsf{id}, i)$, $A$ checks whether a previous query has already been made for $(\mathsf{id}, i)$. If so, $A$ retrieves the recorded tuple $(\mathsf{id}, i, \boldsymbol{m_i}, t_i, T_i)$. Otherwise, $A$ chooses an element $t_i \in_R \mathbb{Z}_p$ and computes the tag as the point $T_i = t_i P \in E(\mathbb{F}_q)[p]$ and records the tuple $(\mathsf{id}, i, \boldsymbol{m_i}, t_i, T_i)$. In both cases $A$ returns $T_i$. $A$ answers $B$'s hash oracle queries for the hash function $h$ as follows. If $A$ receives a hash query for some value $\mathsf{id}\|i$, then $A$ checks whether a previous query has already been made for $\mathsf{id}\|i$. If so, $A$ retrieves the recorded tuple $(\mathsf{id}\|i, x_{\mathsf{id},i})$ and otherwise $A$ chooses a value $x_{\mathsf{id},i} \in_R \mathbb{Z}_p$ and records the tuple $(\mathsf{id}\|i, x_{\mathsf{id},i})$. In both cases $A$ returns $x_{\mathsf{id},i}$. $A$ answers $B$'s hash oracle queries for the hash function $H$ as follows. If $A$ receives a hash query for some value $\mathsf{id}\|i$, then $A$ checks whether a previous query has already been made for $\mathsf{id}\|i$. If so, $A$ retrieves the recorded tuple $(\mathsf{id}\|i, H(\mathsf{id}\|i))$. Otherwise, $A$ retrieves $(\mathsf{id}, i, \boldsymbol{m_i}, t_i, T_i)$ from the list of recorded tagging queries and the required tuples $(\mathsf{id}\|i, x_{\mathsf{id},i})$ from the list of recorded hash queries for the hash function $h$ and computes

$$H(\mathsf{id}\|i) = s_1^{-1}(t_iP - s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} m_{i,j}(\alpha^j P)) \in E(\mathbb{F}_q)[p]$$

and records the tuple $(\mathsf{id}\|i, H(\mathsf{id}\|i))$. In both cases, $A$ returns the recorded value $H(\mathsf{id}\|i)$.

In the retrieve phase, $A$ generates challenges $\mathcal{C}_i = (\mathsf{id}^*, I, l)$ for $1 \le i \le \lambda$ for a file $\mathcal{M}^*$ identified by $\mathsf{id}^*$. Now, if $B$ delivers proofs $\pi_i = (\mu, \tau)$ for file $\mathcal{M}' \neq \mathcal{M}^*$ and challenge $\mathcal{C}_i$ then $A$ proceeds as follows. W.l.o.g. we demonstrate the reduction by means of a single proof $\pi$ with $\mu = (\mu_j)_{j=1}^t = \left(\sum_{i\in I} l^i h(\mathsf{id}^*\|i)\cdot m^*_{i,j}\right)_{j=1}^t$ and $\tau = \sum_{i \in I} l^i T_i$. Recall that the verification relation, after substituting the simulated hash function values, looks as follows:

$$\sum_{i\in I} l^i \cdot (t_iP - s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} m_{i,j}(\alpha^j P)) + s_2 \sum_{j=1}^{t} \mu_j(\alpha^j P) = \tau$$

By replacing the values $\alpha^j P$, we obtain:

$$\sum_{i\in I} l^i \cdot (t_iP - s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} m_{i,j}(\phi_j P + \psi_j R)) + s_2 \sum_{j=1}^{t} \mu_j(\phi_j P + \psi_j R) = \tau$$

Simplifications of the left-hand-side yield:

$$\sum_{i\in I} l^i (t_iP - s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} m_{i,j}(\phi_j P + \psi_j R)) + s_2 \sum_{j=1}^{t} \mu_j(\phi_j P + \psi_j R) =$$

$$\sum_{i\in I} l^i (t_iP - s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} m_{i,j}(\phi_j P + \psi_j R) + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} m^*_{i,j}(\phi_j P + \psi_j R)) =$$

$$\sum_{i\in I} l^i (t_iP + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} (m^*_{i,j} - m_{i,j})(\phi_j P + \psi_j R)) =$$

$$\sum_{i\in I} l^i (t_iP + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}(\phi_j P + \psi_j R)) =$$

$$\sum_{i\in I} l^i (t_iP + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\phi_j P) + \sum_{i\in I} l^i (t_iP + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\psi_j R)$$

where $\delta_{i,j} = m_{i,j} - m^*_{i,j}$. Equating the so obtained simplification with the right-hand-side and subtracting the right-hand-side, we get:

$$\sum_{i\in I} l^i (t_i + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\phi_j)P + (\sum_{i\in I} l^i s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\psi_j)R = \mathcal{O}$$

From this it follows that

$$R = rP = -\frac{\sum_{i\in I} l^i (t_i + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\phi_j)}{(\sum_{i\in I} l^i s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\psi_j)} P.$$

Consequently, if $B$ provides a forged proof, i.e., if there is at least one pair $m_{i,j} \neq m^*_{i,j}$ implying that $\sum_{j=1}^{t}(m_{i,j} - m^*_{i,j}) \neq 0$, then $A$ can compute $r \in \mathbb{Z}_p$, which is the solution to the given instance $(P, R = rP)$ of the ECDLP in $E(\mathbb{F}_q)[p]$. $A$ returns $(P, R, r)$.

Note that the reduction can be performed analogously using the public verification relation, which would lead to the following relation:

$$g^r = g^{-\frac{\sum_{i\in I} l^i (t_i + s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\phi_j)}{(\sum_{i\in I} l^i s_2 x_{\mathsf{id},i} \sum_{j=1}^{t} \delta_{i,j}\psi_j)}}.$$

$A$ returns $(P, R, r)$ which is a valid solution to the ECDLP, since $E(\mathbb{F}_q)[p] \simeq \mathbb{F}_{q^k}^*[p] \simeq \mathbb{Z}_p$ and $g = e(P, P')$ is a generator of $\mathbb{F}_{q^k}^*[p]$.

Finally, we need to show that for sufficiently large $\lambda$ the original file $\mathcal{F}$ can be reconstructed. As shown in [13], it suffices to prove that if the encoding of the file (primary encoding) as well as the response from the server (secondary encoding) are efficiently erasure-decodable, then the original file can be efficiently reconstructed. The primary encoding of the file is done using a Reed-Solomon code and is, thus, efficiently erasure decodable. By looking at the server's response, in particular at the value

$$\mu = (\mu_j)_{j=1}^t = \big(h(\mathsf{id}\|i)\sum_{i\in I} m_{i,j} l^i\big)_{j=1}^t$$

it is clear that the verifier can eliminate the values $h(\mathsf{id}\|i)$ giving the sequence $\big(\sum_{i\in I} m_{i,j} l^i\big)_{j=1}^t$, whose elements constitute Reed-Solomon encodings of the sequences $(m_{i,j})_{i\in I}$. This means that our secondary encoding is also efficiently erasure decodable. Consequently, by applying Lemma 6 and then applying Lemma 7 of [13], the desired result follows.