Securely Outsourcing Exponentiations with Single Untrusted Program for Cloud Storage

Yujue Wang[1,2,3], Qianhong Wu[3,1], Duncan S. Wong[2], Bo Qin[4], Sherman S.M. Chow[5], Zhen Liu[2], and Xiao Tan[2]

[1] Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education, School of Computer, Wuhan University, Wuhan, China
[2] Department of Computer Science City University of Hong Kong, Hong Kong
wyujue2-c@my.cityu.edu.hk, {duncan,zhenliu7}@cityu.edu.hk, xiaotan4@gapps.cityu.edu.hk
[3] School of Electronic and Information Engineering Beihang University, Beijing, China
qianhong.wu@buaa.edu.cn
[4] School of Information, Renmin University of China, Beijing, China
bo.qin@ruc.edu.cn
[5] Department of Information Engineering Chinese University of Hong Kong, Hong Kong
sherman@ie.cuhk.edu.hk

Abstract. Provable Data Possession (PDP) allows a file owner to outsource her files to a storage server such that a verifier can check the integrity of the outsourced file. Public verifiable PDP schemes allow any one other than the file owner to be a verifier. At the client side (file owner or verifier), a substantial number of modular exponentiations is often required. In this paper we make PDP more practical via proposing a protocol to securely outsource the (most generic) variable-exponent variable-base exponentiations in *one* untrusted program model. Our protocol demonstrates advantages in efficiency or privacy over existing protocols coping with only special cases in two or single untrusted program model. We then apply our generic protocol to Shacham-Waters PDP and a variant of Yuan-Yu PDP. The analyses show that our protocol makes PDP much more efficient at the client side.

Keywords: Offloading computation, Verifiable computation, Modular exponentiation, Provable Data Possession, Cloud storage.

1 Introduction

A recent trend to reduce the storage costs is to outsource it to a cloud service provider. It is beneficial to the (cloud service) clients since the outsourced files can be easily shared with others. There are also increasing concerns about the security of their outsourced files. In addition to the many secrecy issues [1–3], there are concerns about whether those outsourced files are still kept intact.

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part I, LNCS 8712, pp. 326–343, 2014.

Traditional primitives like signature or signcryption (e.g., [4, 5]) are insufficient for this purpose since it requires the signed message for verification.

Considerable efforts have been devoted to addressing these concerns. A promising one is to verify the outsourced file integrity via *provable data possession* (PDP) [6–11]. In many scenarios, *publicly verifiable* PDP is preferable. However, existing such schemes need many *modular exponentiations*, especially in file processing and integrity verification. Exponentiations are relatively expensive for devices with limited computation capacity such as mobile phones or tablet, albeit outsourcing files is attractive to mobile terminals due to their limited storage space. This motivates us to consider how to make PDP more affordable by securely outsourcing exponentiation to a *computation server*.

1.1 Our Contributions

Secure Exponentiations Outsourcing. We present the first *generic* scheme that allows to securely outsource variable-exponent variable-base multi-exponentiations to just one untrusted computation server. Although a few schemes have been proposed for securely outsourcing *variable-exponent variable-base exponentiations*, they treat the special cases of our setting and are not satisfactory enough for outsourcing multi-exponentiations in practice. Both Hohenberger-Lysyanskaya scheme [12] and Chen *et al.*'s scheme [13] are presented in *two untrusted program model*. This seems a strong assumption hard to be met as the client needs to outsource her exponentiations to two computation servers who will not collude. Our scheme is implemented in *single* untrusted program model, and is also superior since less interactions are needed between the client and the computation server. Although Dijk *et al.*'s scheme [14] is presented in a *single untrusted program model*, it cannot ensure the privacy of the queried input since the base of the outsourced exponentiation is known to the untrusted server. In contrast, our scheme is computationally more efficient and ensures higher privacy level.

Before showing our algorithm of outsourcing exponentiations, we provide two preprocessing subroutines which generate random pairs. The first one, called BPV^{+}, generates statistically indistinguishable random pairs and is suitable to implement outsourcing schemes over cyclic groups of large prime order. The other one, standard multiplicative base list (SMBL), is more efficient especially for applications over finite groups on elliptic curves.

Secure Offloading Provable Data Possession. Built on our scheme of outsourcing generic exponentiations, we investigate how to efficiently and securely offload PDP schemes. As we know, most existing publicly verifiable PDP schemes take many expensive exponentiations. Specifically, those exponentiations intensively occur in two stages. One is the file processing algorithm $\mathsf{ProFile}$, which is carried out by the file owner to generate the verifiable metadata for a given file before uploading it to the storage server. The other one is the verification algorithm Vrfy, which is executed by a verifier to check whether the outsourced file is kept intact. Thus, to speed-up PDP schemes at the client side, we let the file owner and the verifier securely outsource exponentiations to an untrusted computation

server. To showcase the effectiveness of our protocol, we show how to securely offload Shacham-Waters PDP [8] and a variant of Yuan-Yu PDP [11]. Analyses show that for both offloaded PDP schemes, the computational efficiency at the client side is greatly improved compared with the plain ones. Furthermore, the saving computation cost increases with the number of elements involved in a multi-exponentiation.

1.2 Related Work

Provable Data Possession. The concept of PDP was first introduced by Ateniese *et al.* [6], which allows the clients to check the integrity of an outsourced file without retrieving its entire version from the cloud storage server. For responding to integrity queries, the cloud server does not need to access the entire file. There are some attempts in outsourcing operations in PDP. Wang *et al.* [15] considered PDP in identity-based setting to relieve the users from complicated certificate management. In Wang *et al.*'s privacy-preserving publicly-verifiable PDP scheme [10], a third party auditor (TPA) is introduced to securely carry out verification algorithm on behalf of file owners. Here the privacy of the file is preserved from the view of the TPA. In another work with privacy concern, Wang *et al.* [9] considered a scenario such that the members of an organization can perform the file processing for PDP with the help of a security-mediator (SEM). As a side effect, part of the computation workloads is also outsourced. The privacy is preserved from the view of the SEM.

Proofs of Retrievability (PoR) is a closely related notion to PDP. PoR was first introduced by Juels and Kaliski [7], which enables the storage server to convince its clients that the outsourced files can be entirely retrieved. In their scheme [7], the clients can only submit a limited number of integrity queries, because the corresponding responses are produced by checking whether the special sentinels have been modified. Shacham and Waters [8] presented (both privately and publicly) verifiable PoR schemes, which are the first ones that being proved in the strongest model. Based on polynomial commitments [16], Yuan and Yu [11] proposed a public verifiable PoR scheme with constant communication costs for integrity verification. Benabbas, Gennaro and Vahlis [17] investigated verifiable delegation of computations for high degree polynomial functions to an untrusted server, and based on which a PoR scheme is discussed where the file blocks are represented as the coefficients in a polynomial. For reducing the computation costs at the client sides, Li *et al.* [18] introduced a semi-honest cloud audit server into PoR framework. Specifically, the audit server takes charge of preprocessing the data for generating metadata as well as auditing the data integrity on behalf of data owners.

Securely Outsourcing Exponentiations. Dijk *et al.* [14] considered outsourcing algorithms of *variable-exponent fixed-base* and *fixed-exponent variable-base exponentiations* in one untrusted server model. Specifically, the computations are outsourced to one powerful but untrusted server, where the variable parts are blinded before sending to the server. Ma, Li and Zhang [19] also proposed secure algorithms of outsourcing these two types of exponentiations by using two

non-collusion untrusted servers. An algorithm of outsourcing variable-exponent variable-base exponentiations was also presented in [14], where the outsourced base is known to the server. Both the schemes of Hohenberger and Lysyanskaya [12] and Chen *et al.* [13] considered outsource-secure algorithms of variable-exponent variable-base exponentiations in one-malicious version of two untrusted program model, that is, the computations are securely outsourced to two servers one of which is trusted and will not collude with the other dishonest one. Chen *et al.* [13] also studied how to securely and efficiently outsource *simultaneous exponentiations* in this security model.

Other Secure Outsourcing Schemes. Tsang, Chow and Smith [20] proposed the concept of batch pairing delegations for securely outsourcing expensive pairing operations in batch. Canard, Devigne and Sanders [21] also showed delegating a pairing can be both secure and efficient. The main operation required by the client is exponentiation. With our new protocol, it provides a "complete solution" of outsourcing many pairing-based schemes, in particular, the PDP schemes we concerned in this work. Xu, Amariucai and Guan [22] considered a scenario in which the burdensome computations are delegated to a server P, and the verification on the outputs of P is also outsourced to another server V. Gennaro, Gentry and Parno [23] first considered verifiable computation by combining Yao's Garbled Circuits with a fully-homomorphic encryption scheme, such that the evaluation of a function can be securely outsourced to a remote untrusted server. Carter *et al.* [24] also considered securely outsourcing function evaluation by using an efficient outsourced oblivious transfer protocol. Zhang and Safavi-Naini [25] considered special cases of securely outsourcing function evaluation, i.e., univariate polynomial evaluation and matrix multiplication, without fully-homomorphic encryption, yet with multilinear maps. Recently, Wang *et al.* [26, 27] showed how to compute over data encrypted under multiple keys in the two-server model. Two-server model is also used in other work such as efficient privacy-preserving queries over distributed databases [28].

2 Definitions and Security Requirements

In this section, we review the definitions of outsource-secure algorithms as well as the corresponding security requirements [12, 13].

An algorithm Alg to be outsourced is divided into two parts, namely, a trusted part T which should be efficient compared with Alg and is carried out by the outsourcer, and an untrusted part U which is invoked by T. Following the works [12, 13], we use the same notations in the upcoming sections. Specifically, T^U denotes the works that carried out by T by invoking U. The adversary A is modelled by a pair of algorithms $A = (E, U')$, where E represents the adversarial environment, and generates adversarial inputs for Alg; we denote U' an adversarial software. It is invoked in the same way as U and thus it is used to mirror the view of U during the execution of T^U.

Definition 1 (Algorithm with Outsource-IO). *An outsourcing algorithm Alg takes five inputs and produces three outputs, i.e.,* $Alg(x_{hs}, x_{hp}, x_{hu}, x_{ap}, x_{au}) \rightarrow (y_s, y_p, y_u)$.

Inputs: *All the inputs are classified by how they are generated and how much the adversary* $A = (E, U')$ *knows about them. The first three inputs are generated by an honest party, while the last two are generated by the adversarial environment* E. *Specifically, the* honest, secret *input* x_{hs} *is unknown to both* E *and* U; *the* honest, protected *input* x_{hp} *may be known to* E, *but is protected from* U; *the* honest, unprotected *input* x_{hu} *may be known by both* E *and* U; *the* adversarial, protected *input* x_{ap} *is known by* E, *but protected from* U; *and the* adversarial, unprotected *input* x_{au} *may be known by both* E *and* U.

Outputs: *Similarly, all the outputs are classified by how much the adversary* $A = (E, U')$ *knows about them. The first one* y_s *is called the* secret *output and unknown to both parties of A; the* protected *output* y_p *may be known by* E, *but unknown to* U; *and* y_u *is the* unprotected *output known by both* E *and* U.

It is assumed that the adversary A consists of two parties E and U'. Both can only make direct communications before the execution of T^U. In any other cases if necessary, they should be communicated via T. An outsource-secure algorithm (T, U) requires that neither party of A can learn anything interesting during the execution of T^U. This requirement is captured by the *simulatability* of (T, U). In other words, for any probabilistic polynomial-time (PPT) adversary $A = (E, U')$, the view of E on the execution of T^U can be simulated in a computationally indistinguishable way given the protected and unprotected inputs, and similarly, the view of U' can also be simulated but only given the unprotected inputs.

Definition 2 (Correctness). *Let Alg be an algorithm with outsource-IO. A pair of algorithms* (T, U) *is said to be a* correct *implementation of Alg, if* $Alg = T^U$ *for any honest, secret, or honest, protected, or adversarial, protected inputs.*

Definition 3 (λ-security). *Let Alg be an algorithm with outsource-IO. A pair of algorithms* (T, U) *is said to be a* λ*-outsource-secure implementation of Alg if: for any PPT adversary* $A = (E, U')$, *both the views of* E *and* U' *can be simulated on the execution of* T^U, *i.e., there exist PPT algorithms* (S_1, S_2) *such that the following two pairs of random variables are computationally indistinguishable under the security parameter* λ,

$$\Pr[EView_{real} \sim EView_{ideal}] \geq 1 - 2^{-\lambda},$$

and

$$\Pr[UView_{real} \sim UView_{ideal}] \geq 1 - 2^{-\lambda}.$$

Pair One: $EView_{real} \sim EView_{ideal}$, *which means that* E *learns nothing during the execution of* T^U. *They are defined by the following processes that proceed in rounds, where the notation "*$\leftarrow$*" denotes the outputs of the procedure in the right hand side.*

- *The i-th round of* real *process consists of the following steps, in which I is an honest, stateful process that the environment E cannot access:*
  - $(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^\kappa, istate^{i-1})$;
  - $(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow E(1^\kappa, EView^{i-1}_{real}, x^i_{hp}, x^i_{hu})$;
  - $(tstate^i, ustate^i, y^i_s, y^i_p, y^i_u) \leftarrow T^{U'(ustate^{i-1})}(tstate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$.

  Thus, the view of E in the i-th round of the real process is $EView^i_{real} = (estate^i, y^i_p, y^i_u)$ *and the overall view is just its view in the last round, i.e.,* $EView_{real} = EView^i_{real}$ *for some i such that* $stop^i = True$.
- *The i-th round of* ideal *process consists of the following steps. In which, the stateful algorithm* S_1 *is given all the non-secret outputs which Alg generates in i-th round, but knows nothing about the secret input* x^i_{hs}. *Finally,* S_1 *outputs* (y^i_p, y^i_u) *or some other values* (Y^i_p, Y^i_u), *which is captured by using a boolean indicator* ind^i.
  - $(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^\kappa, istate^{i-1})$;
  - $(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow E(1^\kappa, EView^{i-1}_{real}, x^i_{hp}, x^i_{hu})$;
  - $(astate^i, y^i_s, y^i_p, y^i_u) \leftarrow Alg(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$;
  - $(sstate^i, ustate^i, Y^i_p, Y^i_u, ind^i) \leftarrow S_1^{U'(ustate^{i-1})}(sstate^{i-1}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au}, y^i_p, y^i_u)$;
  - $(z^i_p, z^i_u) = ind^i(Y^i_p, Y^i_u) + (1 - ind^i)(y^i_p, y^i_u)$.

  Thus, the view of E in the i-th round of the ideal process is $EView^i_{ideal} = (estate^i, z^i_p, z^i_u)$ *and the overall view is just its view in the last round, i.e.,* $EView_{ideal} = EView^i_{ideal}$ *for some i such that* $stop^i = True$.

Pair Two: $UView_{real} \sim UView_{ideal}$, *which means that the untrusted software* U' *learns nothing during the execution of* $T^{U'}$.

- *By the definition of Pair One,* U'*'s view in the* real *process is* $UView_{real} = ustate^i$ *for some i such that* $stop^i = True$.
- *The i-th round of* ideal *process consists of the following steps, in which the stateful algorithm* S_2 *is just given the unprotected outputs which Alg generates in i-th round:*
  - $(istate^i, x^i_{hs}, x^i_{hp}, x^i_{hu}) \leftarrow I(1^\kappa, istate^{i-1})$;
  - $(estate^i, j^i, x^i_{ap}, x^i_{au}, stop^i) \leftarrow E(1^\kappa, EView^{i-1}_{real}, x^i_{hp}, x^i_{hu})$;
  - $(astate^i, y^i_s, y^i_p, y^i_u) \leftarrow Alg(astate^{i-1}, x^{j^i}_{hs}, x^{j^i}_{hp}, x^{j^i}_{hu}, x^i_{ap}, x^i_{au})$;
  - $(sstate^i, ustate^i) \leftarrow S_2^{U'(ustate^{i-1})}(sstate^{i-1}, x^{j^i}_{hu}, x^i_{au})$.

  Thus, U'*'s view in i-th round of ideal process is* $UView^i_{ideal} = (ustate^i)$, *and the overall view is just its view in the last round, i.e.,* $UView_{ideal} = UView^i_{ideal}$ *for some i such that* $stop^i = True$.

As we discussed, by employing the outsource-secure techniques, the clients can relieve from carrying out resource-intensive computations locally. Thus, it is reasonable to measure the efficiency of an outsource-secure algorithm (T, U) by comparing the works that T undertakes to those for the state-of-the-art execution of Alg.

Definition 4 (α-efficient, λ-secure outsourcing). *For a pair of λ-outsource-secure algorithms (T, U) which implement an algorithm Alg, they are α-efficient if for any inputs x, the running time of T is less than an α-multiplicative factor of that of $Alg(x)$.*

Definition 5 (β-checkable, λ-secure outsourcing). *For a pair (T, U) of λ-outsource-secure algorithms which implement an algorithm Alg, they are β-checkable if for any inputs x, T can detect any deviations of U' from its advertised functionality during the execution of $T^{U'(x)}$ with probability at least β.*

In practice, the cloud server can only misbehave with very small probability. Otherwise, it will be caught after outsourcing invocations.

Definition 6 ((α, β, λ)-outsource-security). *A pair of algorithms (T, U) are an (α, β, λ)-outsource-secure implementation of an algorithm Alg if they are both α-efficient and β-checkable, λ-secure outsourcing.*

3 Secure Modular Exponentiation Outsourcing

3.1 Preprocessing

In [13, 12], a subroutine *Rand* is used to generate random pairs. On each invocation, *Rand* takes a prime p, a base $g \in \mathbb{Z}_p^*$ and possibly some other values as inputs, and outputs a random, independent pair $(a, g^a \bmod p)$ for some $a \in_R \mathbb{Z}_p^*$. For security, the distribution of *Rand* outputs should be computationally indistinguishable from truly random ones. There are two ways to realize this subroutine. One is to use a trusted server to generate a number of random and independent pairs for T, and the other is let T generate those random pairs by using EBPV generator [29].

In this paper, we provide two preprocessing subroutines for generating random pairs. Both of them take a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order p and possibly some other values as inputs, and output a random, independent pair (a, g^a) for some $a \in_R \mathbb{Z}_p^*$. Besides, those subroutines maintain two tables, i.e., a *static table* ST and a *dynamic table* DT.

***BPV*$^+$:** The first one is called BPV$^+$ which is derived from the BPV generator [30], i.e., by running BPV or EBPV generator totally offline. In detail, BPV$^+$ is described as follows.

- **ST**: T chooses n random numbers $\beta_1, \cdots, \beta_n \in \mathbb{Z}_p^*$, and computes $\nu_i = g^{\beta_i}$ for each $i \in [1, n]$. T stores these n pairs (β_i, ν_i) in the *static* table ST.
- **DT**: T maintains a dynamic table DT, of which each element (α_j, μ_j) can be produced as follows. T chooses a random subset $\mathbb{S} \subseteq \{1, \cdots, n\}$ such that $|\mathbb{S}| = k$ and computes $\alpha_j = \sum_{i \in \mathbb{S}} \beta_i \mod p$. If $\alpha_j \neq 0 \mod p$, then compute $\mu_j = \prod_{i \in \mathbb{S}} \nu_i$; otherwise, discard α_j and repeat this procedure. On each invocation of BPV$^+$, T just picks a pair (α, μ) and removes it from DT, and then replenishes some fresh random pairs in its idle time.

SMBL: Another preprocessing algorithm is called *standard multiplication base list* (SMBL), which produces truly random pairs.

- **ST**: T computes $\nu_i = g^{2^i}$ for every $i \in \{0, \cdots, \lceil \log p \rceil\}$ and stores these pairs (i, ν_i) in the *static* table ST. In fact, $\nu_i = \nu_{i-1} \cdot \nu_{i-1}$ for every $i \in \{1, \cdots, \lceil \log p \rceil\}$.
- **DT**: T maintains a dynamic table DT, of which each element (α_j, μ_j) can be produced as follows. T chooses a random value $\alpha_j \in \mathbb{Z}_p^*$ and denotes its i-th bit by $\alpha_{i,j}$. Let $\mathbb{A} \subseteq \{0, \cdots, \lceil \log p \rceil\}$ be the set of i such that $\alpha_{i,j} = 1$. Computes $\mu_j = \prod_{i \in \mathbb{A}} \nu_i$. On each invocation of SMBL, T just picks a pair (α, μ) and removes it from DT, and then replenishes some fresh random pairs afterwards.

Note that, the preprocessing algorithms, i.e., *Rand*, BPV$^+$ and SMBL just deal with exponentiations with some fixed-base g. The comparison between the proposed subroutines is shown in Table 1 in terms of computation and storage costs and randomness of the generated pairs. Here, $|DT|$ and $ES_{\mathbb{G}}$ denote the cardinality of table DT and the element size of group $\mathbb{G}$, respectively. It is easy to see that BPV$^+$ produces statistically indistinguishable random pairs, while SMBL generates truly random ones. Both the computations and the storage of BPV$^+$ with regard to ST are more costly than its counterpart in SMBL. However, if p is large, then the computations of DT in BPV$^+$ are more efficient than that in SMBL, since the parameter k is relatively small compared with p. Thus, BPV$^+$ will be more effective when used in preprocessing or outsourcing schemes that are designed over cyclic groups of large prime orders, such as DSA scheme, El Gamal scheme as well as RSA-type schemes, etc. While SMBL is well-suited for other cases, e.g., in the applications over ECC such as that discussed in Section 4, where a relatively small p is secure enough.

Table 1. Comparison of preprocessing subroutines BPV$^+$ and SMBL

Subroutines		Computation costs	Storage costs	Randomness
BPV$^+$	ST	$n\mathtt{E}$	$n \log p + nES_{\mathbb{G}}$	-
	DT	$\lvert DT\rvert(k-1)\mathtt{A} + \lvert DT\rvert(k-1)\mathtt{M}$	$\lvert DT\rvert \log p + \lvert DT\rvert ES_{\mathbb{G}}$	Statistically indistinguishable
SMBL	ST	$\lceil \log p \rceil \mathtt{M}$	$(\lceil \log p \rceil + 1) \log \log p$ $+(\lceil \log p \rceil + 1)ES_{\mathbb{G}}$	-
	DT	$\lvert DT\rvert \lceil \log p \rceil / 2\mathtt{M}$	$\lvert DT\rvert \log p + \lvert DT\rvert ES_{\mathbb{G}}$	Truly random

($\mathtt{A}$, $\mathtt{M}$, and $\mathtt{E}$ denote addition, multiplication, and exponentiation, respectively.)

3.2 Generic Algorithm for Outsourcing Exponentiations

Let $\mathbb{G}$ be a cyclic group of prime order p and g be a generator. Takes $a_{i,j} \in_R \mathbb{Z}_p$ and $u_{i,j} \in_R \mathbb{G}$ $(1 \le i \le r, 1 \le j \le s)$ as inputs, the algorithm $GExp$ outputs $(\prod_{j=1}^{s} u_{1,j}^{a_{1,j}}, \cdots, \prod_{j=1}^{s} u_{r,j}^{a_{r,j}})$, i.e.,

$$GExp((a_{1,1},\cdots,a_{1,s};u_{1,1},\cdots,u_{1,s}),\cdots,\ (a_{r,1},\cdots,a_{r,s};u_{r,1},\cdots,u_{r,s})) \rightarrow (\prod_{j=1}^{s} u_{1,j}^{a_{1,j}},\cdots,\prod_{j=1}^{s} u_{r,j}^{a_{r,j}}),$$

where $\{a_{i,j} : 1 \le i \le r, 1 \le j \le s\}$ may be secret or (honest/adversarial) protected, $\{u_{i,j} : 1 \le i \le r, 1 \le j \le s\}$ are distinct and may be (honest/adversarial) protected, and $\{\prod_{j=1}^{s} u_{1,j}^{a_{1,j}} : 1 \le i \le r\}$ may be secret or protected.

Step 1: T invokes the algorithm BPV$^+$ or SMBL to generate four pairs $(\alpha_1,\mu_1),\cdots,(\alpha_4,\mu_4)$ where $\mu_i = g^{\alpha_i}$. Pick a random value χ such that $\chi \ge 2^\lambda$ where λ is a security parameter, e.g. $\lambda = 64$. For every pair (i,j) such that $1 \le i \le r$ and $1 \le j \le s$, pick a random number $b_{i,j} \in_R \mathbb{Z}_p^*$ and compute the following values

- $c_{i,j} = a_{i,j} - b_{i,j}\chi \mod p$;
- $w_{i,j} = u_{i,j}/\mu_1$;
- $h_{i,j} = u_{i,j}/\mu_3$;
- $\theta_i = \left(\alpha_1 \sum_{j=1}^{s} b_{i,j} - \alpha_2\right)\chi + \left(\alpha_3 \sum_{j=1}^{s} c_{i,j} - \alpha_4\right) \mod p$.

Step 2: T invokes BPV$^+$ or SMBL to obtain $(t_1, g^{t_1}),\cdots,(t_{r+2}, g^{t_{r+2}})$ and queries the server U in random order as:
$U(\theta_i/t_i, g^{t_i}) \rightarrow B_i$, for every i $(1 \le i \le r)$;
$U(\theta/t_{r+1}, g^{t_{r+1}}) \rightarrow A$, where $\theta = t_{r+2} - \sum_{i=1}^{r} \theta_i \mod p$;
$U(b_{i,j}, w_{i,j}) \rightarrow C_{i,j}$, for every i, j $(1 \le i \le r, 1 \le j \le s)$;
$U(c_{i,j}, h_{i,j}) \rightarrow D_{i,j}$, for every i, j $(1 \le i \le r, 1 \le j \le s)$.

Step 3: T checks whether $A \cdot \prod_{i=1}^{r} B_i \stackrel{?}{=} g^{t_{r+2}}$. If it holds, then compute the results as follows, for $1 \le i \le r$,

$$\prod_{j=1}^{s} u_{i,j}^{a_{i,j}} = \left(\mu_2 \prod_{j=1}^{s} C_{i,j}\right)^{\chi} B_i \mu_4 \prod_{j=1}^{s} D_{i,j}\ ;$$

otherwise it indicates that U has produced wrong responses, and thus T outputs "error".

3.3 Security Analysis

Lemma 1 (Correctness). *In single untrusted program model, the above algorithms (T,U) are a* correct *implementation of GExp, where the inputs $\{(a_{i,1},\cdots,a_{i,s};u_{i,1},\cdots,u_{i,s}) : 1 \le i \le r\}$ may be honest, secret; or honest, protected; or adversarial, protected.*

Proof. If U performs honestly, we have

$$A \cdot \prod_{i=1}^{r} B_i = g^{\theta} \cdot \prod_{i=1}^{r} g^{\theta_i} = g^{t_{r+2} - \sum_{i=1}^{r} \theta_i} \cdot \prod_{i=1}^{r} g^{\theta_i} = g^{t_{r+2}},$$

and for every $1 \le i \le r$, we have

$$\begin{aligned}
&\left(\mu_2 \prod_{j=1}^{s} C_{i,j}\right)^{\chi} B_i \mu_4 \prod_{j=1}^{s} D_{i,j} \\
&= \left(g^{\alpha_2} \prod_{j=1}^{s} w_{i,j}^{b_{i,j}}\right)^{\chi} g^{\left(\alpha_1 \sum_{j=1}^{s} b_{i,j} - \alpha_2\right)\chi + \left(\alpha_3 \sum_{j=1}^{s} c_{i,j} - \alpha_4\right)} g^{\alpha_4} \prod_{j=1}^{s} h_{i,j}^{c_{i,j}} \\
&= \left(\prod_{j=1}^{s} w_{i,j}^{b_{i,j}} g^{\alpha_1 \sum_{j=1}^{s} b_{i,j}}\right)^{\chi} g^{\alpha_3 \sum_{j=1}^{s} c_{i,j}} \prod_{j=1}^{s} h_{i,j}^{c_{i,j}} \\
&= \left(\prod_{j=1}^{s} w_{i,j}^{b_{i,j}} \prod_{j=1}^{s} \mu_1^{b_{i,j}}\right)^{\chi} \prod_{j=1}^{s} \mu_3^{c_{i,j}} \prod_{j=1}^{s} h_{i,j}^{c_{i,j}} \\
&= \left(\prod_{j=1}^{s} (\mu_1 w_{i,j})^{b_{i,j}}\right)^{\chi} \prod_{j=1}^{s} (\mu_3 h_{i,j})^{c_{i,j}} \\
&= \left(\prod_{j=1}^{s} u_{i,j}^{b_{i,j}}\right)^{\chi} \prod_{j=1}^{s} u_{i,j}^{c_{i,j}} = \prod_{j=1}^{s} u_{i,j}^{b_{i,j}\chi + c_{i,j}} = \prod_{j=1}^{s} u_{i,j}^{a_{i,j}}.
\end{aligned}$$

Thus, the correctness follows. □

Theorem 1 (λ-security). *In single untrusted program model, the above algorithms (T, U) are a* λ-outsource-secure *implementation of GExp, where the inputs $\{(a_{i,1}, \cdots, a_{i,s}; u_{i,1}, \cdots, u_{i,s}) : 1 \le i \le r\}$ may be honest, secret; or honest, protected; or adversarial, protected, and all the bases are distinct.*

The proof of Theorem 1 is given in the full version of this paper.

Theorem 2. *In single untrusted program model, the above algorithms (T, U) are an $\left(O\left(\frac{rs+r\log\chi+r}{rs\ell}\right), \frac{r+1}{2rs+r+1}, \lambda\right)$-outsource-secure implementation of GExp.*

Proof. On one hand, the well-known square-and-multiply method to calculate one exponentiation u^a takes roughly 1.5ℓ multiplications, where ℓ denotes the bit-length of a. Accordingly, by using this method, it requires roughly $1.5rs\ell$ multiplications for calculating r multi-exponentiations $(\prod_{j=1}^{s} u_{1,j}^{a_{1,j}}, \cdots, \prod_{j=1}^{s} u_{r,j}^{a_{r,j}})$. On the other hand, $GExp$ makes $(r+3)$ inversions and $(5rs+6r+1.5r\log\chi+1)$ multiplications for calculating the same exponentiations. Thus, the algorithms (T, U) are an $O\left(\frac{rs+r\log\chi+r}{rs\ell}\right)$-efficient implementation of $GExp$.

Since U cannot distinguish the test queries from the other real queries that T makes, if it deviates the execution of $GExp$, the deviations of U will be detected with probability $\frac{r+1}{2rs+r+1}$. □

3.4 Comparisons

We conduct thorough comparisons between our scheme $GExp$ and the up-to-date schemes [13, 14, 12] on outsourcing variable-exponent variable-base

exponentiations, in terms of computation and communication costs at the client side, and security properties.

The schemes with regard to computing just one exponentiation are summarized in Table 2, in which $ES_{\mathbb{G}}$ denotes the element size of $\mathbb{G}$. All of those schemes enjoy results checkability of certain levels. Both schemes [12, 13] implemented in two untrusted program model make several invocations to subroutine *Rand*. For each invocation of *Rand*, the online phase will take roughly $(2k + h - 4)$ multiplications by using their suggested EBPV generator, where k is the same parameter as that in BPV$^+$ and $h \geq 1$. Dijk *et al.*'s scheme [14] takes about $(3(3\log s + 2\log w_s)/2 + 5)$ multiplications where w_s is determined by the security parameter s. In their scheme, one may note that T makes two rounds of interactions with just one untrusted server U for querying 4 powers. Although Dijk *et al.*'s scheme [14] is presented in single untrusted program model, the base g is known to the server U.

Table 2. Comparison of securely outsourcing single exponentiation u^a

	Scheme [12]	Scheme [13]	Scheme [14]	Ours
Multiplications	$6O(Rand) + 9$	$5O(Rand) + 7$	$4.5\log s$ $+3\log w_s + 5$	$12 + 1.5\log\chi$
Inversions	5	3	1	4
Queries to U	8	6	4	4
Communications	$8\log p + 16ES_{\mathbb{G}}$	$6\log p + 12ES_{\mathbb{G}}$	$2\log n + 7ES_{\mathbb{G}}$	$4\log p + 8ES_{\mathbb{G}}$
Privacy	✓	✓	×	✓
Checkability	1/2	2/3	1	1/2
Security Model	Two UP	Two UP	Single UP	Single UP

("Two/Single UP" denotes Two / Single Untrusted Program Model respectively)

Chen *et al.* [13] also presented an algorithm for outsourcing simultaneous exponentiations in two untrusted program model. A comparison between their scheme [13] and ours is shown in Table 3.

Table 3. Comparison of securely outsourcing simultaneous exponentiation $u_1^{a_1}u_2^{a_2}$

	Scheme [13]	Ours
Multiplications	$5O(Rand) + 10$	$17 + 1.5\log\chi$
Inversions	3	4
Queries to U	8	6
Communications	$8\log p + 16ES_{\mathbb{G}}$	$6\log p + 12ES_{\mathbb{G}}$
Privacy	✓	✓
Checkability	1/2	1/3
Security Model	Two UP	Single UP

4 Securely Offloading PDP

We first review the definition of PDP (e.g., [6, 8]).

Definition 7 (PDP). *A Provable Data Possession scheme consists of five polynomial time computable algorithms, i.e.,* KeyGen, $\mathsf{ProFile}$, Chall, PrfGen *and* Vrfy.

- $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$*: on input* 1^κ *where* $\kappa \in \mathbb{N}$ *is a security parameter, the (randomized) key generation algorithm, which is carried out by the cloud clients, generates a pair of public and secret key* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{ProFile}(\mathsf{sk}, M) \to (t, M^*)$*: on input a file* M *and the secret key* sk*, the processing file algorithm, which is carried out by the file owner, generates a file tag* t *and a processed file* M^* *for* M.
- $\mathsf{Chall}(\mathsf{pk}, t) \to Q$*: on input the public key* pk *and a file tag* t*, the challenge generation algorithm, which is carried out by the verifier, produces a challenge* Q.
- $\mathsf{PrfGen}(\mathsf{pk}, t, M^*, Q) \to R$*: on input the public key* pk*, a file tag* t*, a processed file* M^* *and a challenge* Q*, the proof generation algorithm, which is carried out by the cloud storage server, produces a response* R.
- $\mathsf{Vrfy}(\mathsf{pk}, \mathsf{sk}, t, Q, R) \to \{0, 1\}$*: on the public key* pk*, the secret key* sk*, a file tag* t *and a challenge-response pair* (Q, R)*, the deterministic verification algorithm outputs "1" if* R *is a valid response for* Q*, or "0" otherwise.*

Our schemes are built from bilinear pairings reviewed below. Suppose $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The group $\mathbb{G}$ is said to be bilinear if there exists a cyclic group $\mathbb{G}_T$ and a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that: (1) Bilinearity: $\forall \mu, \nu \in \mathbb{G}$, and $\forall a, b \in \mathbb{Z}_p$, $\hat{e}(\mu^a, \nu^b) = \hat{e}(\mu, \nu)^{ab}$; (2) Non-degeneracy: $\hat{e}(g, g) \neq 1$ is a generator of $\mathbb{G}_T$.

4.1 Securely Offloading Shacham-Waters PDP

Let $H : \{0,1\}^* \to \mathbb{G}$ be the collusion-resistant map-to-point hash function (to be modelled as a random oracle) and $\Sigma = (\mathsf{SKG}, \mathsf{SSig}, \mathsf{SVer})$ be the Boneh-Lynn-Shacham signature scheme [31]. We are ready to describe how to securely offload the Shacham-Waters PDP scheme.

$\mathsf{KenGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$**:** First generate a random signing key pair $(spk, ssk) \leftarrow \Sigma.\mathsf{SKG}(1^\kappa)$. Then random pick $\alpha \leftarrow_R \mathbb{Z}_p^*$ and compute $v = g^\alpha$. Thus, the public key and secret key are $\mathsf{pk} = (v, spk)$ and $\mathsf{sk} = (\alpha, ssk)$, respectively.

$\mathsf{ProFile}(\mathsf{sk}, M) \to (t, M^*)$**:** Given a file M, split it into blocks such that each block has s sectors, i.e., $M = \{M_i = (m_{i,1}, \cdots, m_{i,s}) : 1 \leq i \leq n\}$. Parse sk to get (α, ssk). Then, choose a random file name $name \in_R \mathbb{Z}_p^*$ and s random elements $u_1, \cdots, u_s \in_R \mathbb{G}$. Let $t_0 = name \parallel n \parallel u_1 \parallel \cdots \parallel u_s$. Compute the file tag as $t \leftarrow t_0 \parallel \Sigma.\mathsf{SSig}_{ssk}(t_0) = t_0 \parallel GExp(ssk; H(t_0))$. For each file block M_i $(1 \leq i \leq n)$, compute $h_i = H(name \parallel i)$ and invoke $GExp$ to generate metadata σ_i as

$$\sigma_i \leftarrow GExp(\alpha, \alpha m_{i,1}, \cdots, \alpha m_{i,s}; h_i, u_1, \cdots, u_s).$$

Then, send the processed file $M^* = \{m_{i,j}\}_{1\le i\le n, 1\le j\le s} \cup \{\sigma_i\}_{1\le i\le n}$ to the cloud storage server.

$\mathsf{Chall}(\mathsf{pk}, t) \to Q$**:** Parse pk as (v, spk) and use spk to validate t. If it is invalid, output 0 and terminate; otherwise, parse t to obtain $(name, n, u_1, \cdots, u_s)$. Pick a random subset $I \subseteq [1, n]$ and a random value $v_i \in_R \mathbb{Z}_p^*$ for each $i \in I$. Send $Q = \{(i, v_i) : i \in I\}$ to the cloud storage server.

$\mathsf{PrfGen}(\mathsf{pk}, t, M^*, Q) \to R$**:** Parse the processed file M^* as $\{m_{i,j}\}_{1\le i\le n, 1\le j\le s} \cup \{\sigma_i\}_{1\le i\le n}$, and the challenge Q to obtain $\{(i, v_i) : i \in I\}$. Compute

$$\mu_j = \sum_{(i,v_i)\in Q} v_i m_{i,j} \in \mathbb{Z}_p, \text{ and } \sigma = \prod_{(i,v_i)\in Q} \sigma_i^{v_i} \in \mathbb{G}.$$

Then send $R = (\mu_1, \cdots, \mu_s, \sigma)$ to the verifier.

$\mathsf{Vrfy}(\mathsf{pk}, \mathsf{sk}, t, Q, R) \to \{0, 1\}$**:** Parse R to obtain $(\mu_1, \cdots, \mu_s) \in (\mathbb{Z}_p)^s$ and $\sigma \in \mathbb{G}$. If parsing fails, output 0 and terminate. Otherwise, compute $h_i = H(name \parallel i)$ for each $i \in I$ and

$$\rho = GExp\Big((v_i)_{(i,v_i)\in Q}, \mu_1, \cdots, \mu_s; (h_i)_{(i,v_i)\in Q}, u_1, \cdots, u_s\Big).$$

Check whether $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\rho, v)$ holds; if so, output 1; otherwise, output 0.

Correctness. If the computation server performs honestly, we have

$$\begin{aligned}
\sigma_i &= GExp(\alpha, \alpha m_{i,1}, \cdots, \alpha m_{i,s}; h_i, u_1, \cdots, u_s) \\
&= h_i^{\alpha} \cdot \prod_{j=1}^{s} u_j^{\alpha m_{i,j}} = \left(H(name \parallel i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}}\right)^{\alpha}. \\
\rho &= GExp\Big((v_i)_{(i,v_i)\in Q}, \mu_1, \cdots, \mu_s; (h_i)_{(i,v_i)\in Q}, u_1, \cdots, u_s\Big) \\
&= \prod_{(i,v_i)\in Q} h_i^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j} = \prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j}.
\end{aligned}$$

The correctness of the file tag generation is straightforward.

4.2 Securely Offloading a Variant of Yuan-Yu PDP

In the following, for a given vector $\boldsymbol{c} = (c_0, \cdots, c_{s-1})$ for $c_i \in \mathbb{Z}_p$, we use $f_{\boldsymbol{c}}(x)$ to denote the polynomial defined as $f_{\boldsymbol{c}}(x) = \sum_{i=0}^{s-1} c_i x^i$ over $\mathbb{Z}_p$.

$\mathsf{KenGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$**:** First generate a random signing key pair $(spk, ssk) \leftarrow \Sigma.\mathsf{SKG}(1^\kappa)$. Then pick two random values $\alpha, \beta \leftarrow_R \mathbb{Z}_p^*$, and compute $\gamma = g^\beta$, $\lambda = g^{\alpha\beta}$ and $\{g^{\alpha^j} : j \in [0, s-1]\}$. Thus, the public key and secret key are $\mathsf{pk} = (\gamma, \lambda, spk, g, g^\alpha, \cdots, g^{\alpha^{s-1}})$ and $\mathsf{sk} = (\alpha, \beta, ssk)$, respectively.

$\mathsf{ProFile}(\mathsf{sk}, M) \to (t, M^*)$**:** Given a file M, split it into blocks such that each block has s sectors, i.e., $M = \{M_i = (m_{i,0}, \cdots, m_{i,s-1}) : 1 \le i \le n\}$. Choose a random file name $name \in_R \mathbb{Z}_p^*$ and set $t_0 = name \parallel n$. Compute the file tag as $t \leftarrow t_0 \parallel \Sigma.\mathsf{SSig}_{ssk}(t_0) = t_0 \parallel GExp(ssk; H(t_0))$. For each file block M_i $(1 \le i \le n)$:

– compute $h_i = H(name \parallel i)$ and $f_i = \beta \cdot f_{\boldsymbol{\pi}_i}(\alpha) = \beta \sum_{j=0}^{s-1} m_{i,j}\alpha^j \mod p$;
– invoke $GExp$ to generate metadata, i.e., $\sigma_i \leftarrow GExp(\beta, f_i; h_i, g)$.

Then, send the processed file $M^* = \{m_{i,j}\}_{1\le i\le n, 0\le j\le s-1} \cup \{\sigma_i\}_{1\le i\le n}$ to the cloud storage server.

$\mathsf{Chall}(\mathsf{pk}, t) \to Q$**:** Parse pk to obtain spk and use it to validate the signature on t. If it is invalid, output 0 and terminate; otherwise, parse t to obtain $(name, n)$. Pick a random subset $I \subseteq [1, n]$ and a random value $v_i \in_R \mathbb{Z}_p^*$ for each $i \in I$. Choose another random value $r \in_R \mathbb{Z}_p^*$ and send $Q = \{r, (i, v_i) : i \in I\}$ to the cloud storage server.

$\mathsf{PrfGen}(\mathsf{pk}, t, M^*, Q) \to R$**:** Parse the processed file M^* as $\{m_{i,j}\}_{1\le i\le n, 1\le j\le s} \cup \{\sigma_i\}_{1\le i\le n}$, and the challenge Q to obtain $\{r, (i, v_i) : i \in I\}$. Compute

$$\mu_j = \sum_{(i,v_i)\in Q} v_i m_{i,j} \in \mathbb{Z}_p, \text{ and } \sigma = \prod_{(i,v_i)\in Q} \sigma_i^{v_i} \in \mathbb{G}.$$

Define a polynomial $f_{\boldsymbol{\mu}}(x) = \sum_{j=0}^{s-1} \mu_j x^j \mod p$ and calculate $y = f_{\boldsymbol{\mu}}(r)$. Then compute the polynomial $f_{\boldsymbol{\omega}}(x) = \frac{f_{\boldsymbol{\mu}}(x) - f_{\boldsymbol{\mu}}(r)}{x - r}$ using polynomial long division, and denote its coefficient vector as $\boldsymbol{\omega} = (\omega_0, \cdots, \omega_{s-2})$. Compute $\psi = g^{f_{\boldsymbol{\omega}}(\alpha)} = \prod_{j=0}^{s-2} (g^{\alpha^j})^{\omega_j}$ and send $R = (\psi, y, \sigma)$ to the verifier.

$\mathsf{Vrfy}(\mathsf{pk}, \mathsf{sk}, t, Q, R) \to \{0, 1\}$**:** After receiving the proof response R, the verifier parses it to obtain (ψ, y, σ) and parses t to obtain $(name, n)$. If parsing fails, output 0 and halting. Otherwise, compute $h_i = H(name \parallel i)$ for each $i \in I$, and invoke $GExp$ to compute

$$\rho = GExp\Big(y, -r, (v_i)_{(i,v_i)\in Q}; g, \psi, (h_i)_{(i,v_i)\in Q}\Big).$$

If $\hat{e}(\sigma, g) = \hat{e}(\psi, \lambda)\hat{e}(\rho, \gamma)$ holds, then output 1; otherwise, output 0.

Correctness. It is easy to see that, if both the computation server and the storage server perform honestly, we have

$$\begin{aligned}
\sigma_i &= GExp(\beta, f_i; h_i, g) = h_i^{\beta} \cdot g^{\beta f_{\boldsymbol{\pi}_i}(\alpha)} = H(name \parallel i)^{\beta} \cdot g^{\beta \sum_{j=0}^{s-1} m_{i,j}\alpha^j} \\
&= \Big(H(name \parallel i) \cdot g^{\sum_{j=0}^{s-1} m_{i,j}\alpha^j}\Big)^{\beta} = \Big(H(name \parallel i) \cdot \prod_{j=0}^{s-1} g^{m_{i,j}\alpha^j}\Big)^{\beta}. \\
\rho &= GExp\Big(y, -r, (v_i)_{(i,v_i)\in Q}; g, \psi, (h_i)_{(i,v_i)\in Q}\Big) \\
&= g^y \psi^{-r} \prod_{(i,v_i)\in Q} h_i^{v_i} = g^y \psi^{-r} \prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}.
\end{aligned}$$

The correctness of file tag generation is straightforward. Specifically,

$$\begin{aligned}\hat{e}(\sigma, g) &= \hat{e}\Big(\prod_{(i,v_i)\in Q} \sigma_i^{v_i}, g\Big) \\ &= \hat{e}\left(\prod_{(i,v_i)\in Q} \left(H(name \parallel i) \cdot g^{f_{\boldsymbol{\pi}_i}(\alpha)}\right)^{v_i\beta}, g\right) \\ &= \hat{e}\Big(\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big)\hat{e}\Big(g^{\beta\sum_{(i,v_i)\in Q} v_i f_{\boldsymbol{\pi}_i}(\alpha)}, g\Big) \\ &= \hat{e}\Big(\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big)\hat{e}\big(g^{f_{\boldsymbol{\mu}}(\alpha)}, g^{\beta}\big).\end{aligned}$$

$$\begin{aligned}\hat{e}(\psi, \lambda)\hat{e}(\rho, \gamma) &= \hat{e}\Big(g^{f_{\boldsymbol{\omega}}(\alpha)}, g^{\alpha\beta}\Big)\hat{e}\Big(g^{y}\psi^{-r}\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big) \\ &= \hat{e}\Big(g^{\alpha f_{\boldsymbol{\omega}}(\alpha)}, g^{\beta}\Big)\hat{e}\Big(g^{f_{\boldsymbol{\mu}}(r)}g^{-rf_{\boldsymbol{\omega}}(\alpha)}\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big) \\ &= \hat{e}\Big(g^{(\alpha-r)f_{\boldsymbol{\omega}}(\alpha)+f_{\boldsymbol{\mu}}(r)}, g^{\beta}\Big)\hat{e}\Big(\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big) \\ &= \hat{e}\big(g^{f_{\boldsymbol{\mu}}(\alpha)-f_{\boldsymbol{\mu}}(r)+f_{\boldsymbol{\mu}}(r)}, g^{\beta}\big)\hat{e}\Big(\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big) \\ &= \hat{e}\big(g^{f_{\boldsymbol{\mu}}(\alpha)}, g^{\beta}\big)\hat{e}\Big(\prod_{(i,v_i)\in Q} H(name \parallel i)^{v_i}, g^{\beta}\Big).\end{aligned}$$

4.3 Efficiency Analysis

The original Shacham-Waters PDP [8] takes many exponentiations in algorithm ProFile and algorithm Vrfy. For processing a file $M = \{M_i = (m_{i,0}, \cdots, m_{i,s-1}) : 1 \le i \le n\}$, the file owner takes 1 exponentiation and $(s+1)$ exponentiations for producing the file tag and one metadata, respectively. While the verifier takes $(|I| + s)$ exponentiations during the verification phase. In our variant of the Yuan-Yu PDP [11], since α is a secret key, we assume $\alpha^2, \cdots, \alpha^{s-1}$ have been pre-calculated by the file owner. Thus, the algorithm ProFile and algorithm Vrfy take $(2n+1)$ and $(|I|+2)$ exponentiations, respectively. We compare the computation costs as well as the communication overheads between the client and the untrusted computation server of the schemes with/without outsourcing exponentiations in Table 4. It can be seen that offloading makes both schemes much more efficient.

Table 4. Comparison of Two PDP Schemes with and without Outsourcing

	Original scheme	Outsourced scheme	
	Computation costs		Communication costs
Shacham-Waters PDP			
File tag generation	$1\mathtt{h} + 1\mathtt{E}$	$1\mathtt{h} + (12 + 1.5\log\chi)\mathtt{M} + 4\mathtt{I}$	$4\log p + 8ES_{\mathbb{G}}$
Each metadata generation	$1\mathtt{h} + s\mathtt{M} + (s+1)\mathtt{E}$	$1\mathtt{h} + 4\mathtt{I}$ $+(6s + 1.5\log\chi + 12)\mathtt{M}$	$(2s+4)\log p$ $+(4s+8)ES_{\mathbb{G}}$
Verification	$\lvert I\rvert\mathtt{h} + (\lvert I\rvert + s - 1)\mathtt{M}$ $+(\lvert I\rvert + s)\mathtt{E} + 2\mathtt{P}$	$(5\lvert I\rvert + 5s + 1.5\log\chi + 7)\mathtt{M}$ $+\lvert I\rvert\mathtt{h} + 4\mathtt{I} + 2\mathtt{P}$	$(2\lvert I\rvert + 2s + 2)\log p$ $+(4\lvert I\rvert + 4s + 4)ES_{\mathbb{G}}$
Our Variant of Yuan-Yu PDP			
File tag generation	$1\mathtt{h} + 1\mathtt{E}$	$1\mathtt{h} + (12 + 1.5\log\chi)\mathtt{M} + 4\mathtt{I}$	$4\log p + 8ES_{\mathbb{G}}$
Each metadata generation	$1\mathtt{h} + s\mathtt{M} + 2\mathtt{E}$	$1\mathtt{h} + 4\mathtt{I}$ $+(s + 1.5\log\chi + 17)\mathtt{M}$	$6\log p + 12ES_{\mathbb{G}}$
Verification	$\lvert I\rvert\mathtt{h} + (\lvert I\rvert + 2)\mathtt{M}$ $+(\lvert I\rvert + 2)\mathtt{E} + 3\mathtt{P}$	$\lvert I\rvert\mathtt{h} + 4\mathtt{I} + 3\mathtt{P}$ $+(5\lvert I\rvert + 1.5\log\chi + 18)\mathtt{M}$	$(2\lvert I\rvert + 6)\log p$ $+(4\lvert I\rvert + 12)ES_{\mathbb{G}}$

Notations: h denotes hash evaluation; M, I and E denote one multiplication, one inversion and one exponentiation, respectively; P denotes one bilinear pairing evaluation.

5 Concluding Remark

Outsourcing storage can save the cost of a client in maintaining the storage locally. Cryptographic approaches like provable data possession ensures the integrity of the outsourced file can still be verified, yet these often require modular exponentiations expensive to computationally bounded devices. We filled this gap with offloaded PDP by securely and efficiently outsourcing the most generic variable-exponent variable-base exponentiations to one untrusted computation server. Compared with the known schemes, our scheme is not only superior in its security model, but also its efficiency, interactions and privacy. Our protocol may find applications in many other cryptographic solutions which use number-theoretic cryptographic techniques.

Acknowledgements and Disclaimer. We appreciate the anonymous reviewers for their valuable suggestions. Qianhong Wu is the corresponding author. This work is supported by the National Key Basic Research Program (973 program) through project 2012CB315905, by the National Nature Science Foundation of China through projects 61272501, 61173154, 61370190 and 61003214, by a grant from the RGC of the HKSAR, China, under Project CityU 123913, by the Beijing Natural Science Foundation through project 4132056, by the Fundamental Research Funds for the Central Universities, and the Research Funds (No. 14XNLF02) of Renmin University of China and by the Open Research Fund of Beijing Key Laboratory of Trusted Computing. Sherman Chow is supported by the Early Career Scheme and the Early Career Award of the Research Grants

Council, Hong Kong SAR (CUHK 439713), and grants (4055018, 4930034) from Chinese University of Hong Kong.

References

1. Deng, H., Wu, Q., Qin, B., Chow, S.S.M., Domingo-Ferrer, J., Shi, W.: Tracing and Revoking Leaked Credentials: Accountability in Leaking Sensitive Outsourced Data. In: 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp. 425–443. ACM, New York (2014)
2. Deng, H., Wu, Q., Qin, B., Domingo-Ferrer, J., Zhang, L., Liu, J., Shi, W.: Ciphertext-Policy Hierarchical Attribute-Based Encryption with Short Ciphertexts. Information Sciences 275, 370–384 (2014)
3. Deng, H., Wu, Q., Qin, B., Mao, J., Liu, X., Zhang, L., Shi, W.: Who is touching my cloud. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8712, pp. 362–379. Springer, Heidelberg (2014)
4. Chow, S.S.M., Yiu, S.M., Hui, L.C.K., Chow, K.P.: Efficient Forward and Provably Secure ID-Based Signcryption Scheme with Public Verifiability and Public Ciphertext Authenticity. In: Lim, J.-I., Lee, D.-H. (eds.) ICISC 2003. LNCS, vol. 2971, pp. 352–369. Springer, Heidelberg (2004)
5. Qin, B., Wang, H., Wu, Q., Liu, J., Domingo-Ferrer, J.: Simultaneous Authentication and Secrecy in Identity-Based Data Upload to Cloud. Cluster Computing 16, 845–859 (2013)
6. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable Data Possession at Untrusted Stores. In: 14th ACM Conference on Computer and Communications Security (CCS), pp. 598–609. ACM, New York (2007)
7. Juels, A., Kaliski Jr., B.S.: PoRs: Proofs of Retrievability for Large Files. In: 14th ACM Conference on Computer and Communications Security (CCS), pp. 584–597. ACM, New York (2007)
8. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
9. Wang, B., Chow, S.S.M., Li, M., Li, H.: Storing Shared Data on the Cloud via Security-Mediator. In: 33rd IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 124–133 (2013)
10. Wang, C., Chow, S.S.M., Wang, Q., Ren, K., Lou, W.: Privacy-Preserving Public Auditing for Secure Cloud Storage. IEEE Transactions on Computers 62(2), 362–375 (2013)
11. Yuan, J., Yu, S.: Proofs of Retrievability with Public Verifiability and Constant Communication Cost in Cloud. In: International Workshop on Security in Cloud Computing, pp. 19–26. ACM, New York (2013)
12. Hohenberger, S., Lysyanskaya, A.: How to Securely Outsource Cryptographic Computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005)
13. Chen, X., Li, J., Ma, J., Tang, Q., Lou, W.: New Algorithms for Secure Outsourcing of Modular Exponentiations. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 541–556. Springer, Heidelberg (2012)
14. Dijk, M., Clarke, D., Gassend, B., Suh, G., Devadas, S.: Speeding up Exponentiation using an Untrusted Computational Resource. Designs, Codes and Cryptography 39(2), 253–273 (2006)

15. Wang, H., Wu, Q., Qin, B., Domingo-Ferrer, J.: Identity-Based Remote Data Possession Checking in Public Clouds. Information Security, IET 8(2), 114–121 (2014)
16. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010)
17. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable Delegation of Computation over Large Datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011)
18. Li, J., Tan, X., Chen, X., Wong, D.S.: An Efficient Proof of Retrievability with Public Auditing in Cloud Computing. In: 5th International Conference on Intelligent Networking and Collaborative Systems (INCoS), pp. 93–98 (2013)
19. Ma, X., Li, J., Zhang, F.: Outsourcing Computation of Modular Exponentiations in Cloud Computing. Cluster Computing 16(4), 787–796 (2013)
20. Tsang, P.P., Chow, S.S.M., Smith, S.W.: Batch Pairing Delegation. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) IWSEC 2007. LNCS, vol. 4752, pp. 74–90. Springer, Heidelberg (2007)
21. Canard, S., Devigne, J., Sanders, O.: Delegating a Pairing Can Be Both Secure and Efficient. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 549–565. Springer, Heidelberg (2014)
22. Xu, G., Amariucai, G., Guan, Y.: Delegation of Computation with Verification Outsourcing: Curious Verifiers. In: ACM Symposium on Principles of Distributed Computing (PODC), pp. 393–402. ACM, New York (2013)
23. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
24. Carter, H., Mood, B., Traynor, P., Butler, K.: Secure Outsourced Garbled Circuit Evaluation for Mobile Devices. In: 22nd USENIX Conference on Security, pp. 289–304. USENIX Association, Berkeley (2013)
25. Zhang, L.F., Safavi-Naini, R.: Private Outsourcing of Polynomial Evaluation and Matrix Multiplication Using Multilinear Maps. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 329–348. Springer, Heidelberg (2013)
26. Wang, B., Li, M., Chow, S.S.M., Li, H.: Computing Encrypted Cloud Data Efficiently Under Multiple Keys. In: 4th IEEE Security and Privacy in Cloud Computing, co-located with IEEE Conference on Communications and Network Security (CNS), pp. 504–513 (2013)
27. Wang, B., Li, M., Chow, S.S.M., Li, H.: A Tale of Two Servers: Efficient Privacy-Preserving Computation over Cloud Data under Multiple Keys. In: 2nd IEEE Conference on Communications and Network Security, CNS (2014)
28. Chow, S.S.M., Lee, J.H., Subramanian, L.: Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases. In: Network and Distributed System Security Symposium, NDSS (2009)
29. Nguyen, P., Shparlinski, I.E., Stern, J.: Distribution of Modular Sums and the Security of the Server Aided Exponentiation. In: Cryptography and Computational Number Theory. Progress in Computer Science and Applied Logic, vol. 20, pp. 331–342. Birkhäuser, Basel (2001)
30. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up Discrete Log and Factoring Based Schemes via Precomputations. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 221–235. Springer, Heidelberg (1998)
31. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. Journal of Cryptology 17(4), 297–319 (2004)