Multimed Tools Appl (2016) 75:6207–6235
DOI 10.1007/s11042-015-2567-8

Secret sharing approach for securing cloud-based pre-classification volume ray-casting

Manoranjan Mohanty[1] · Wei Tsang Ooi[2] · Pradeep K. Atrey[3]

Received: 25 September 2014 / Revised: 11 February 2015 / Accepted: 12 March 2015 /
Published online: 31 March 2015

Abstract With the evolution in cloud computing, cloud-based volume rendering, which outsources data rendering tasks to cloud datacenters, is attracting interest. Although this new rendering technique has many advantages, allowing third-party access to potentially sensitive volume data raises security and privacy concerns. In this paper, we address these concerns for cloud-based pre-classification volume ray-casting by using Shamir's (k, n) secret sharing and its variant (l, k, n) ramp secret sharing, which are homomorphic to addition and scalar multiplication operations, to hide color information of volume data/images in datacenters. To address the incompatibility issue of the modular prime operation used in secret sharing technique with the floating point operations of ray-casting, we consider excluding modular prime operation from secret sharing or converting the floating number operations of ray-casting to fixed point operations – the earlier technique degrades security and the later degrades image quality. Both these techniques, however, result in significant

✉ Manoranjan Mohanty
manoranjan.jnu@gmail.com

Wei Tsang Ooi
ooiwt@comp.nus.edu.sg

Pradeep K. Atrey
patrey@albany.edu

[1] Security Lab, SICS Swedish ICT, Kista, Sweden

[2] Department of Computer Science, National University of Singapore, Singapore, Singapore

[3] Department of Computer Science, University at Albany - State University of New York, Albany, NY, USA

data overhead. To lessen the overhead at the cost of high security, we propose a modified ramp secret sharing scheme that uses the three color components in one secret sharing polynomial and replaces the shares in floating point with smaller integers.

Keywords Multimedia security · Shamir's secret sharing · Cloud-based imaging · Volume ray-casting

1 Introduction

With the advances in cloud computing, organizations are outsourcing 3D volumetric data rendering tasks to third-party cloud datacenters [12, 21, 22, 25]. To this end, datacenters are being used for extracting iso-surfaces and rendering the extracted iso-surfaces in the case of indirect volume rendering [12, 25], and performing volume ray-casting in the case of the direct volume rendering [21, 22]. Compared to conventional server-side rendering, such cloud-based rendering is more scalable, more economical, offers better computing resources, and can produce lower visualization latency by performing rendering in a datacenter closer to the client location.

Security and privacy, however, become major issues in cloud-based rendering. Disclosing important data/image to a third-party cloud provider leads to the concerns of confidentiality, integrity, and availability [15]. For example, an adversary can access a datacenter that stores medical data/image of patients and misuse the information in several ways. The adversary can: (i) sell the disease information to interested parties such as insurance companies; (ii) modify a medical image to provide misleading information to doctors; (iii) leak the medical information of a prominent person to the public and media. Due to these potential threats, laws, such as the *HIPPA* act in the USA, the *PIPED* act in Canada, and the *Data Protection Directive* in European countries, are enacted to protect the information of their citizens.

One can protect these personal volumetric data from datacenters by hiding them with a cryptosystem. To render these hidden volumetric data, however, the cryptosystem must be homomorphic to the rendering operations so that the rendered images can be recovered from the hidden image(s). In other words, if the cryptosystem hides secret volume data V with an operation $H(\cdot)$, a datacenter renders the data with an operation $R(\cdot)$, and a user recovers the secret rendered image from the hidden rendered image $R(H(V))$ with an operation $H^{-1}(\cdot)$, then the condition

$$R(V) \approx H^{-1}(R(H(V)))$$

must hold. To our knowledge, there is no cryptosystem that can fulfill this requirement with acceptable overhead [19]. Somewhat homomorphic cryptosystems, which can perform certain homomorphic operations such as addition and scalar multiplication with low overhead, however, are available [2, 10]. Among the main two types of somewhat homomorphic cryptosystems: (i) secret sharing-based schemes, and (ii) public key encryption-based schemes, secret sharing-based schemes can provide better data confidentiality [1]. Therefore, secret sharing is often used to hide highly important information such as cryptographic keys [9], military data [5] etc. Furthermore, secret sharing schemes, without using an additional cryptosystem, can simultaneously provide data confidentiality, data integrity, and data availability. Among the secret sharing schemes, Shamir's secret sharing is commonly used

since it is more efficient than other secret sharing schemes such as Blakley's secret sharing and Chinese Reminder Theorem-based secret sharing schemes [16].

To address the security and privacy concerns, this paper proposes a framework for hiding color information in cloud-based volume rendering. We focus on pre-classification volume ray-casting (or its variant) as the data rendering technique (operation R) and based our cryptosystem (H and H^{-1}) on Shamir's secret sharing [24]. Our framework hides the color information of 3D volume data, typically used to annotate important information such as disease information [7]. We do not hide opacities, which can disclose shape of the 3D object. We assume that discloser of the shape divulge little confidential information in the absence of colors.

The core idea of the proposed framework is to use Shamir's secret sharing to create n shares of the volume data at the server, and send each *share volume* (i.e., share of secret volume) to a datacenter. The datacenter, upon receiving a rendering request, then renders a *share image* (which colors are hidden) using pre-classification volume ray-casting, and sends the rendered image to the client. The client then recovers the secret image from at least k share images from k datacenters. The use of Shamir's secret sharing in conjunction with volume ray-casting, however, introduces a new challenge as the modular prime operation of secret sharing is incompatible with the floating point operation of volume ray-casting.

We can address this incompatibility issue with two approaches: either (i) exclude modular prime operation from Shamir's secret sharing [8, 17], or (ii) convert the floating point operation of pre-classification volume ray-casting to fixed point operations [3]. We have previously published the former approach, called *Secure Rendering by Modification of Shamir's Secret Sharing* (SR-MSSS) [17]. SR-MSSS is less secure than Shamir's Secret Sharing, as excluding the modular prime operation means that it is not operating in a finite field, and is therefore not perfectly secure (i.e., it can lose some information about the secret color to datacenters). On the other hand, the latter approach, called *Secure Rendering by Modification of Pre-classification Volume Ray-casting* (SR-MPVR), can introduce rounding error that can lead to loss of information in the rendered image.

Both these techniques create three different color shares for the red, green, and blue color components of the 3D volumetric data. Further, representing the share of a color component by either a float or by a large integer incurs high data overhead. For applications requiring minimal overhead at the cost of security, we propose a third technique called *Secure Rendering by Ramp Secret Sharing* (SR-RSS) that improves upon SR-MSSS by first replacing modified Shamir's secret sharing with a modified (3, 4, 5) ramp secret sharing to create only one share for red, green, and blue color, and then restricting the value of a share (which is a float) to a smaller number and representing it with an integer.

Experiments and analyses show that all of our three approaches hide important color information in datacenters and incur insignificant computation overhead. The data overhead, image quality, and the level of security of these techniques, however, differ with the choice of the technique. SR-MPVR provide perfect secrecy, but incurs significant data overhead and renders lossy image. SR-MSSS renders lossless image, but loses some information about the secret to datacenters and incurs significant data overhead. Finally, SR-RSS incurs the lowest overhead among the three, but is the least secure and renders lossy image.

There are three major contributions of this paper.

1. First, we integrate Shamir's secret sharing with the pre-classification volume ray-casting such that the color information of data/image is hidden from the rendering site. Pre-classification volume ray-casting consists of a set of primitive operations such as addition, multiplication, scalar multiplication, division, exponentiation etc. Therefore, integration of Shamir's secret sharing, which is homomorphic to only addition and scalar multiplication, to the rendering pipeline is challenging. We address this challenge by pre-processing the non-homomorphic operations of the ray-casting algorithm, and by adjusting the rendering such that color rendering can be realized only by additions and scalar multiplications.
2. Second, we modify either the Shamir's secret sharing or the pre-classification volume ray-casting to make them compatible with each other. We analyze the loss in security due to the modification of secret sharing and the loss in image quality due to the modification of volume ray-casting, and show that both these losses can be acceptable.
3. Finally, we optimize our schemes, and provide a solution for applications requiring low overheads at the loss of some security.

The rest of this paper is organized as follows. In Section 2, we discuss the related work. Section 3 presents the attacker model. Section 4 provides an overview of pre-classification volume ray-casting and Shamir's secret sharing. In Section 5, we modify pre-classification volume ray-casting to make it compatible with Shamir's secret sharing. In Section 6, we propose our secure cloud-based volume rendering framework. Section 7 provides experimental results of the proposed framework, and analyzes its security and performance. Section 8 concludes.

2 Related work

In this section, we review existing cloud-based rendering techniques to highlight their growing importance, and then extend our discussion to the techniques addressing security and privacy issues in cloud-based imaging.

Recently, cloud-based rendering has drawn the attention of both researchers and enterprises. For example, using Azure cloud, Dorn et al. [6] proposed an adaptive data rendering framework that, according to the requirement, performs volume ray-casting either at a cloud datacenter or at the client. By echoing the concerns of scalability in server-side rendering and resource availability in client-side rendering, Vazhenin proposed yet another cloud-based rendering framework [28]. Similarly, enterprises such as NVIDIA Inc. [21], Sinha system [25], KDDI Inc. [12], NICE [20] etc. have started offering cloud-based 3D medical data rendering frameworks to hospitals.

Research addressing security and privacy issues in cloud-based rendering is a demanding yet little-explored area. To the best of our knowledge, we are the first to propose secure cloud-based volume rendering frameworks using pre-classification volume ray-casting [17] and post-classification volume ray-casting [18]. However, the security and privacy concerns in certain cloud-based computation, such as low-pass filtering [13], image de-noising [23] etc., have been addressed using Shamir's secret sharing as the cryptosystem. The use of Shamir's secret sharing to securely execute a new algorithm (such as pre-classification volume ray-casting) presents a new set of challenges since the algorithm's workflow must be adjusted in such a way that even after hiding additions and scalar multiplications, important information about the input and the output must be kept hidden. Similarly, the security and

privacy issues in cloud-based data/image storage, have been promptly addressed in two possible scenarios: when a single datacenter is used, and when multiple datacenters are used [1, 11, 27, 29]. In the case of the use of a single datacenter, public key encryption techniques or watermarking have been applied to protect the data/image stored in a datacenter [11, 27, 29], and in the case of the use of multiple datacenters, the secret sharing scheme has been used to distribute the secrecy among more than one datacenters [1]. For a complete list of existing cryptographic cloud storage systems, the reader can refer to AlZain et al.'s work [1], which concludes that the secret sharing-based cloud-based secure systems are more secure than the encryption-based systems. However, these cloud-based secure storage systems are designed for cloud-based archiving, and therefore use non-homomorphic cryptosystem such as watermarking, chaos-based encryption, AES, or a somewhat homomorphic cryptosystems such as Shamir's secret sharing to hide data. Thus, these systems cannot be seamlessly extended to secure cloud-based rendering.

In this paper, we extend our earlier work on securing cloud-based pre-classification volume ray-casting framework [17]. Our previous work modified Shamir's secret sharing to make it compatible with pre-classification volume ray-casting. In this paper, we instead modify pre-classification volume ray-casting, and use the modified ray-casting with unmodified secret sharing. Therefore, unlike our previous scheme, we can now offer perfect secrecy. Furthermore, we analyze the security and overheads of both these approaches, and optimize the previous scheme for applications seeking low overheads at the cost of high security.

3 Attacker model

We assume that both the server, which owns the secret volume data and outsources rendering operation to n datacenters, and the client, are secured (i.e., no adversary can access the server or the client). A datacenter, however, can be accessed by a malicious adversary, who can be either a malicious employee of the third-party cloud provider (which hosts the datacenter), or an outsider. Since the adversary is malicious, she can unlawfully read the content of a share volume data and/or share rendered image from a datacenter (confidentiality issue), tamper with the accessed share volume data or share rendered image (integrity issue), deny the share rendered image to the client (availability issue), or relate the identity of a patient to her data/image (privacy issue). We assume that the adversary, however, cannot access k (where, $k \leq n$) or more datacenters at any point of time.

Our objective is to hide the volume data V from a datacenter using Shamir's secret sharing $H(\cdot)$, and allow rendering operation $R(\cdot)$ on the hidden volume data $H(V)$ such that: (i) an adversary cannot know the confidential color information from the share volume data $H(V)$ or the share rendered image $R(H(V))$, (ii) a client can recover the secret image $R(V)$ from at least k $R(H(V))$'s, (iii) a client can detect tampering on $H(V)$ or $R(H(V))$ when $n \geq k$, and (iv) a client will able to recover a $R(V)$ even if $n - k$ datacenters cannot participate.

4 Background

In this section, we will provide an overview of pre-classification volume ray-casting and Shamir's secret sharing.

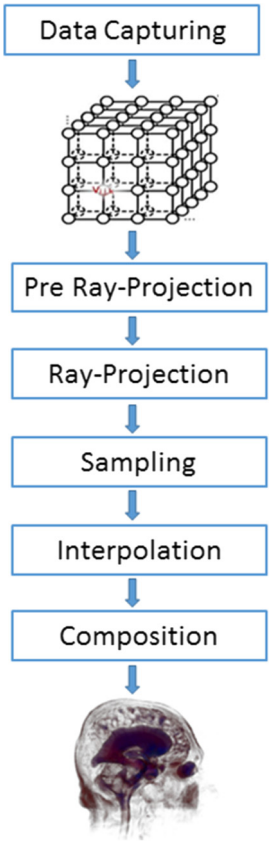

Fig. 1 Conventional Pre-Classification Volume Ray-Casting

4.1 Pre-classification volume ray-casting

Volume ray-casting is one of the preferred technique to directly render volume data [26]. The main idea behind this technique is to project rays from each pixel of the image space on a given volume V, and find the color and opacity along each ray by mapping the physical properties of the object to optical properties (i.e., color and opacity). Volume ray-casting can be classified into pre-classification volume ray-casting and post-classification volume ray-casting.

Pre-classification volume ray-casting (Fig. 1), which is the focus of this paper, consists of the following rendering components: gradient and normal estimation, classification, shading, ray-projection, sampling, interpolation, and composition [14]. These components can be categorized into two main steps: the *pre ray-projection* step and the *post ray-projection* step. The pre ray-projection step consists of gradient estimation, classification, and shading, as they are performed before the projection of rays. This step finds the shaded color and opacity of each data voxel of V, and can be pre-computed independent of the view. The computed colors and opacity are typically stored in a look-up table. The post ray-projection step, on the other hand, consists of the components: sampling, interpolation, and composition. This step finds the color and opacity along each projected ray, and is performed at

run time. In the following sections, we will discuss steps of post ray-projection rendering in detail.

Sampling: In this step, a projected ray is sampled at c sample points $s_1, s_2, ..., s_c$.

Interpolation: The color C_s and the opacity A_s of a sample point s are found by interpolating the colors (i.e., the shaded colors) and the opacities of eight neighboring voxels of s by

$$C_s = \sum_{v \in N(s)} C_v D_v, \tag{1}$$

and

$$A_s = \sum_{v \in N(s)} A_v D_v, \tag{2}$$

respectively, where $N(s)$ is the set of neighboring voxels of s, $C_v \in \mathbb{N}$ is the given color of v, $A_v \in \mathbb{R}$ is the given opacity of v, and $D_v \in \mathbb{R}$, the interpolating factor of v, is obtained from the xyz-coordinate of s and the xyz-coordinates of each data voxel $v \in N(s)$. When D_v is constant, interpolation requires only additions and scalar multiplications.

Composition: This step accumulates the colors and opacities of all the sample points to find the composite color and the composite opacity along a projected ray. Mathematically, the composite color C and the composite opacity A of the sample points $s_1, s_2, ..., s_c$ are defined as

$$C = \sum_{i=1}^{c} C_{s_i} O_i \tag{3}$$

and

$$A = \sum_{i=1}^{c} O_i \tag{4}$$

respectively, where O_i is defined as

$$O_i = A_{s_i} \prod_{j=i+1}^{c} \left(1 - A_{s_j}\right). \tag{5}$$

The composite color is then truncated to obtain the rendered color. Note that for constant O_i, composition can be performed only with additions and scalar multiplications.

In Appendix A, we have provided a simple example of pre-classification volume ray-casting.

4.2 Shamir's secret sharing

Shamir's (k, n) secret sharing is a cryptosystem that hides a secret by dividing it into n shares and recovers the secret by reconstructing it from at least k shares ($k \leq n$).

4.2.1 Share creation

Given a prime number q and a secret $S \in \mathbb{Z}$, where, $S < q$, this step creates n shares of S by first defining a $(k-1)$-degree polynomial

$$F(x) = (S + \alpha_x) \bmod q,$$

where

$$\alpha_x = \sum_{i=1}^{k-1} a_i x^i \tag{6}$$

and $a_i < q$ is a random number in GF(q), and then using this polynomial to find the p^{th} share of S by setting $x = p$.

4.2.2 Secret reconstruction

Given k distinct share numbers $\{x_0, x_1, \ldots x_{k-1}\}$ and shares $\{y_0, y_1, \ldots y_{k-1}\}$ such that $y_i = F(x_i)$, this step reconstructs the secret by first finding the $(k-1)$-degree Lagrange interpolated polynomial $L(x)$ by

$$L(x) = \sum_{i=0}^{k-1} y_i l_i(x) \bmod q,$$

where $l_i(x) = \prod_{j=0, j\neq i}^{k-1} \frac{x - x_j}{x_i - x_j}$, and then solving $L(x)$, which is equivalent to $F(x)$ by the Unisolvence theorem.

Shamir's secret sharing is homomorphic to addition and scalar multiplication [2]. In other words, if the participants hold shares of a set of secrets $S = \{S_1, S_2, ..., S_r\}$, then without communicating amongst themselves, they can compute the shares of the secret $\sum_{i=1}^{r} I_i S_i$, where I_i, for $1 \leq i \leq r$, is an integer. This property, however, cannot be used to hide operands of post ray-projection rendering operations of the pre-classification volume ray-casting, as the modular prime operation of secret sharing is incompatible with the floating point operation of ray-casting. We can address this issue either by omitting the modular prime operation from secret sharing (as we have proposed earlier [17]), or by converting a floating point to a fixed point by first rounding it off to d decimal places and then multiplying the rounded off value with K^d.

Note that even though Chor and Kushilevitz's secret sharing [4] can share a floating point number, their technique is not homomorphic to floating point number scalar multiplication (it is, however, homomorphic to addition and multiplication by integer scalar). Therefore, we cannot use this secret sharing for our framework.

5 Pre-classification volume ray-casting with fixed point operations

To make Shamir's secret sharing compatible with pre-classification volume ray casting, we perform the arithmetic operations involved over a finite field, in integer domain, instead of floating point. In this section, we outline the steps that required this change and analyze the numerical precision required to bound the error in the resulting rendered color to within one (for a detail . Note that we only need to modify the floating point operations of post ray-projection rendering of colors. As we do not hide opacities, their rendering operations will not be modified.

5.1 Modifying interpolation

The interpolation of the colors (1) involves multiplying an integer C_v with a floating point D_v. We convert this multiplication to a fixed point operation as follows. Let $x^{(d)}$ be an integer obtained by first rounding off x to d decimal places and then multiplying the rounded value by 10^d. We have

$$D_v^{(d)} = \left(D_v + \epsilon_{D_v,d}\right) \times 10^d, \tag{7}$$

where $|\epsilon_{D_v,d}| \leq 0.5 \times 10^{-d}$ is the round-off error.

By replacing D_v with $D_v^{(d)}$ in (1), we obtain the scaled interpolated color as

$$C'_s = \sum_{v \in N(s)} C_v D_v^{(d)} \tag{8}$$

$$= (C_s + \epsilon_s) \times 10^d, \tag{9}$$

where

$$\epsilon_s = \sum_{v \in N(s)} C_v \epsilon_{D_v,d}$$

is the total round-off error resulted in the interpolation step.

Since $C_v \leq 255$, $|\epsilon_{D_v,d}| \leq 0.5 \times 10^{-d}$, and $N(s) = 8$, the total error at this step is bounded:

$$|\epsilon_s| \leq 1020 \times 10^{-d}.$$

5.2 Modifying composition

The composition of colors of c sample points, which is given in (3), adds c multiplied values, where a multiplication is between two floating point operands C_{s_i} and O_i. Thus, to ensure that composition performs fixed point operations, we replace the interpolated color C_{s_i} by the scaled interpolated color C'_{s_i} from the pervious step and O_i by an integer

$$O_i^{(f)} = \left(O_i + \epsilon_{O_i,f}\right) \times 10^f, \tag{10}$$

where $\epsilon_{O_i,f}$ is the round-off error, and $|\epsilon_{O_i,f}| \leq 0.5 \times 10^{-f}$.

By replacing C_s with C'_s and O_i with $O_i^{(f)}$ in (3), we obtain the scaled composite color C' as

$$C' = \sum_{i=1}^{c} C'_{s_i} O_i^{(f)} \tag{11}$$

$$= (C + \epsilon) \times 10^{d+f}, \tag{12}$$

where

$$\epsilon = \sum_{i=1}^{c} \left(\epsilon_{s_i} O_i + C_{s_i} \epsilon_{O_i,f} + \epsilon_{s_i} \epsilon_{O_i,f}\right)$$

is the total round-off error resulted in the composition step. Since $C \leq 255$, C' satisfies

$$C' \leq (255 + \epsilon_{max}) \times 10^{d+f} \tag{13}$$

where ϵ_{max} is the upper bound of ϵ.

We know that $C_{s_i} \leq 255$, $|\epsilon_{s_i}| \leq 1020 \times 10^{-d}$, $0 \leq O_i \leq 1$, $\sum_{i=1}^{c} O_i \leq 1$, and $|\epsilon_{O_i,f}| \leq 0.5 \times 10^{-f}$. Therefore the error in scaled composition ϵ satisfies

$$\epsilon \geq 510c \times 10^{-(f+d)} - 127.5c \times 10^{-f} - 1020 \times 10^{-d} \tag{14}$$

and

$$\epsilon \leq 510c \times 10^{-(f+d)} + 127.5c \times 10^{-f} + 1020 \times 10^{-d}. \quad (15)$$

We, however, know that the color of a pixel is a natural number, and is obtained by truncating the fractional part of the composite color C. Hence, if a part of error ϵ cannot change the truncated value, then it is not effective, i.e., the effective error ϵ_{eff} due to ϵ can be obtained by

$$\epsilon_{eff} = \lfloor C + \epsilon \rfloor - \lfloor C \rfloor.$$

The lowest possible value of ϵ_{eff} is ± 1, since the smallest possible value of $|\epsilon|$ (say, $|\epsilon|$ approaches to zero) can also change the value of the rendered color. The following theorem provides the conditions to bound ϵ_{eff} by ± 1.

Theorem 1 *If the number of sample points along a ray, c, satisfies* $c \leq 7 \times 10^t$*, for any integer t, then for* $d \geq 4$ *and* $f \geq t + 3$*, the effective rounding error in rendering* ϵ_{eff} *is bounded by* ± 1.

The proof is by substitution and is straightforward.

The above analysis shows that with fixed point operation, we can limit the error when computing the composite color to less then one. Consider the case where $c < 700$, then it suffices to round off D_v to 4 decimal places and the opacity O_i to 5 decimal places.

6 Cloud-based secure rendering

Using the above modified ray-casting operations, we now describe how secret sharing is done. We first present our secure cloud-based rendering framework using standard Sharmir's secret sharing and the modified ray-casting operations (SR-MPVR). For completeness, we followed this with our previous scheme, SR-MSSS, which uses a weaken Shamir's secret sharing and standard ray-casting operations (SR-MSSS). Finally, a more efficient version that uses ramp secret sharing (SR-RSS) is presented.

6.1 Architecture

The architecture of our framework consists of three components: the server that hosts the secret volume V, n cloud datacenters, and the client who is authorized to access the secret rendered image (Fig. 2). This architecture is designed with the assumption that an adversary neither can access the server or the client nor can access more than $k - 1$ datacenters.

The framework aims to provide both a secure and practical solution. I.e, we not only aim to address data confidentiality, data integrity, and data availability issues but also wish to lessen the computation overhead, data overhead, and loss in image quality.

6.2 SR-MPVR

As shown in Fig. 3, the work flow of our framework can be divided into four steps: (i) data preparation, (ii) ray-projection, (iii) post ray-projection rendering, and (iv) image recovery.

6.2.1 Data preparation

The data preparation step creates n shares of the secret volume V. As we only hide the color information, the hidden volumes, V_1, V_2, ... maintains the shape and order of the voxels of

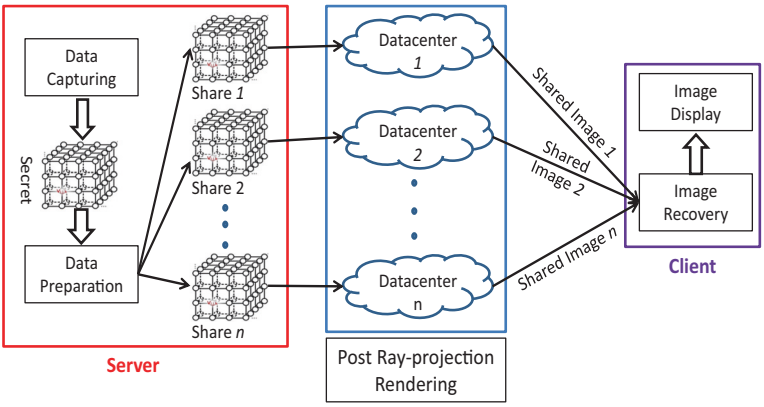

Fig. 2 Our secure cloud-based data visualization framework

V – only the color information is modified. To achieve this, the server first finds the color and the opacity of each data voxel v of a given volume V by performing gradient estimation, classification, and shading operations; and then creates n shares of V by secret sharing the color C_v of v to n shares and copying the opacity A_v of v into n times.

To create shares of C_v, we first choose a prime number q that is greater than the value of the maximum reconstructed secret $(255 + \epsilon_{max}) \times 10^{d+f}$ (which is derived in (13)). By using C_v as the secret in (6) and by setting $x = p$, we find p^{th} share of C_v as

$$C_{v,p} = (C_v + \alpha_p) \bmod q. \tag{16}$$

Note that we use the same q for all shares. Since we need to fix the value of q, which depends on ϵ_{max}, we also determine the rounding precisions d and f in this step.

Next, we create the p^{th} share volume of V, V_p. For each voxel v in V, we create a share voxel v_p with the same xyz-coordinate and opacity, but set the color of v_p to $C_{v,p}$.

6.2.2 *Ray-projection*

The shared volume V_p is stored on datacenter p. When a client requests that the volume V be rendered from a particular viewpoint by projecting a ray, the request is sent to n datacenters. The ray is projected on each of the share volume in these datacenter.

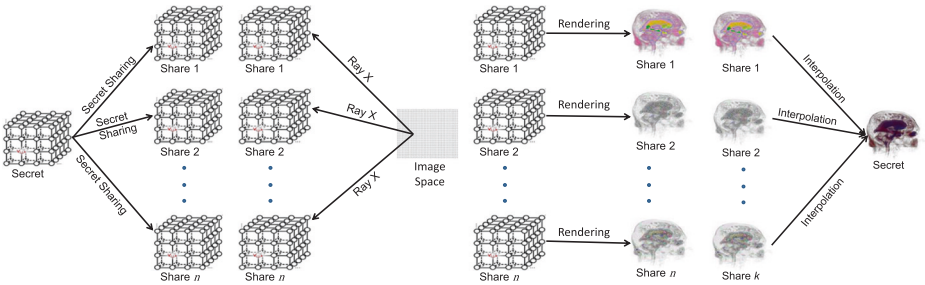

Fig. 3 Proposed secured volume ray-casting

6.2.3 Post ray-projection rendering

Sampling: Consider the projected ray on a share volume V_p. The ray is sampled at c sample points $s_{1,p}, s_{2,p}, \ldots, s_{c,p}$. Note that the xyz-coordinate $s_{i,p}$ and xyz-coordinate s_i (a sample point on the ray when it is projected on V) are the same.

Interpolation: This step finds the opacity and color of a sample point $s_p \in V_p$ by interpolating the opacities and the colors of all eight neighboring voxels of s, and is the same as the operation on the secret volume V.

Since we did not change the opacity, the interpolated opacity is the same whether it is done of a share volume V_p or the secret volume V. The voxels' color, on the other hand, has changed. By putting the color of v_p as $C_{v,p}$ in (8), we get scaled interpolated color $C'_{s,p}$ by

$$C'_{s,p} \equiv \left(C'_s + \alpha_p D_{s_p}\right) \bmod q, \tag{17}$$

where $D_{s_p} = \sum_{v \in N(s_p)} D_v^{(d)}$. The value for $D_{s_p} = D_s$ is the same for all p.

Composition: This next step composites the opacities and the colors of all the sample points $s_{1,p}, s_{2,p}, \ldots, s_{c,p}$ along the projected ray. As we do not share the opacities, they can be composited by conventional composition formula given in (4). The composition of colors, however, need to be performed by the scaled composition technique that is derived in (11).

Using $C'_{s_i,p}$ as the color of $s_{i,p}$ in (11), the scaled composite color is calculated as

$$C'_p \equiv \left(C' + K\alpha_p\right) \bmod q, \tag{18}$$

where $K = \sum_{i=1}^{c} D_{s_i} O_i^{(f)}$ has the same value for all share volume.

Using A_s as the opacity and C'_p as the color of a pixel, the p^{th} datacenter now creates the p^{th} share image, and sends it to the client.

6.2.4 Image recovery

Finally, an authorized user recovers the secret image from k share images obtained from k datacenters. As we do not hide opacities, the opacity of a pixel of a share image become the opacity of the corresponding pixel of the secret image. The color of a pixel of the secret image is recovered from k shared colors (as given in (18) by first using Lagrange interpolation to reconstruct the scaled secret color C' from k shared colors C'_p for some p in $\{1, ..n\}$, and then dividing C' by 10^{d+f}. Therefore, by (12), the recovered secret color is

$$C'' = C + \epsilon,$$

which is close to C (the color obtained by conventional ray-casting) as by Algorithm 1, the error ϵ can be bounded by ± 1 for sufficiently large value of d and f.

In Appendix A, we have provided a simple example of SR-MPVR.

6.3 SR-MSSS

We now describe the SR-MSSS method, an alternative to SR-MPVR. SR-MSSS uses floating point operations, but uses a variation of Shamir's secret sharing with a weaken security guarantee. The main work flow of SR-MSSS is similar to SR-MPVR, so we only highlight the differences below.

In the data preparation step, the server defines a polynomial

$$F'(x) = C_v + \alpha_x$$

for secret sharing, and uses this polynomial to find the p^{th} share of C_v as

$$C_{v,p} = C_v + \alpha_p,$$

without the module prime operation.

As no modular prime operation is used in secret sharing, in the post ray-projection rendering step, a datacenter uses the conventional pre-classification volume ray-casting to render its share volume, and finds the composite color of a share pixel as

$$C_p = C + K\alpha_p$$

(where K is a constant) and the composite opacity as A along a ray projected on V_p (the p^{th} share volume).

As no scaling up by 10^{d+f} operation is required for color rendering, at the image recovery step, the user does not divide the reconstructed color C by 10^{d+f}. The user recovers the secret image using the Lagrange interpolation as per normal Shamir's secret sharing scheme. As evident from this discussion, the recovered secret color is equal to the secret color rendered by conventional ray-casting.

6.4 SR-RSS

We now present another alternative, SR-RSS, that uses ramp secret sharing to reduce the size of the share images.

6.4.1 Data preparation

The objective of this step is to optimize the modified Shamir's secret sharing to create smaller share volumes.

We know that the three color components of a pixel, i.e., the red color R, the green color G, and the blue color B, are rendered by identical rendering operation, and by rendering a share, a datacenter renders all the coefficients used in the secret sharing polynomial. Therefore, we use the color component R_v, G_v, and B_v of a voxel v as three secrets in the secret sharing polynomial. Although used $(3, k, n)$ ramp secret sharing can reduce the data overhead by three times (as instead of creating three shares, ramp secret sharing creates only one share for all three color components), resulted data overhead is sill a concern as a shared color is represented by a floating point number, requiring 4 bytes on a typical systems.

If we limit the value of k and n, however, it is possible to limit the share color to 2^{16}, thus requiring only two bytes.

To use this trick to reduce the value of a color share, we choose a smaller share number at the time of secret sharing by setting the condition $k = 4$ and $n = 5$ for our ramp secret sharing. Thus the secret sharing polynomial becomes

$$F'(x) = a_0 + R_v x + G_v x^2 + B_v x^3, \tag{19}$$

where a_0 is a random number. Using this polynomial and choosing the value of x smaller than five, the server creates share volumes that contains only one share for all three color components of a voxel, and then sends the shared volume to the corresponding datacenter. As the value of color is less than 255 and the value x is less than equal to five, for $a_0 \leq 26011$, the value of $F'(x)$ cannot exceed 65536.

Note that although we can choose $n = k = 3$ to restrict the value of a color share to 3315, we do not recommend this optimization as such a scheme, by not using a random number in secret sharing polynomial, can result in complete breakdown of our framework.

Firstly, such (3, 3, 3) ramp secret sharing do not work for a gray image as it cannot hide black color (when all color components are 0) and white color (when all color components are 255) of a voxel/pixel. Secondly, due to spatial coherence in an image, it is easier for an adversary to guess color of a voxel/pixel from the known share value of the voxel/pixel and the share values of neighboring voxels/pixels of the target voxel/pixel. Similarly, we also do not recommend $n = k$ as this optimization cannot guarantee data integrity and data availability (will be discussed in Section 7.1).

6.4.2 Post ray-projection rendering

Using the post ray-projection rendering operations of SR-MSSS on the p^{th} share of colors (i.e., $F'(p)$'s that are obtained from (19)), we find the p^{th} shared composite color (for all the three color components) by

$$C_p = K + Rp + Gp^2 + Bp^3,$$

where R, G, and B are the red color, green color, and blue color composited by the conventional ray-casting, and $K \leq a_0$ is a constant for all the shares. As the value of each $F'(p)$ is less than 65536, the value of C_p is also less than 65536. We convert C_p to an integer with precision g and send $C_p^{(g)}$ to the client.

6.4.3 Image Recovery

This step finds the secret color components from k given color shares, $C_p^{(g)}$, by first using Lagrange interpolation to find the polynomial

$$L'(x) = K + Rx + Gx^2 + Bx^3 + \epsilon,$$

where ϵ is the rendering error due to rounding off C_p, and then solving $L'(x)$. As the introduced error ϵ satisfies

$$|\epsilon| \leq 0.5 \times 10^{-g} \times \sum_{i=2}^{5} \binom{5}{i},$$

we can choose $g \geq 1$ to obtain $|\epsilon| < 1$.

These optimizations, however, degrades security as both the use of ramp secret sharing scheme, and the exclusion of modular prime operation from secret sharing can disclose information about the secret. Furthermore, due to rounding error, there is a loss in information in the rendered image.

Note that we choose to optimize SR-MSSS as all of the proposed optimization tricks can be applied to SR-MSSS simultaneously. One can, however, extend the trick of using multiple secrets in a secret sharing polynomial to SR-MPVR. The trick of limiting the value of a color share by choosing a suitable share number, however, is not applicable to SR-MPVR as in the case of Shamir's secret sharing, size of a share is independent of the share number.

7 Results and analysis

We simulated the server, datacenters, and the client of our framework in a PC powered by an Intel Core 2 Quad 2.83 GHz processor and with 4GB of RAM. We implemented our

Table 1 Data sets

Name	Dimension	Bits per Voxel	Size
Head	$256 \times 256 \times 124$	8	7.8 MB
Foot	$256 \times 256 \times 256$	8	16 MB
Bucky	$32 \times 32 \times 32$	8	32.2 KB
IronProt	$68 \times 68 \times 68$	8	307.3 KB

framework by first modifying the volume ray-casting module of the open source visualization package VTK to facilitate pre-classification volume raycasting, and then integrating secret sharing into the rendering pipeline. In the case of SR-MPVR, we used (3, 5) Shamir's secret sharing in conjunction with modified pre-classification volume ray-casting; in the case of SR-MSSS, we used modified (3, 5) Shamir's secret sharing in conjunction with pre-classification volume ray-casting; and in the case of SR-RSS we used (3, 4, 5) modified ramp secret sharing in conjunction with pre-classification volume ray-casting. To validate our schemes, we used four sets of test volume data: *Head*, *Foot*, *Bucky*, and *IronProt*, which details are given in Table 1. As the number of sampling points along a ray on any of these test volume does not exceed 700, for SR-MPVR, we fixed $d = 4$ and $f = 6$ to obtain $|\epsilon| \leq 1$. For SR-RSS, we rounded off the floating point numbers by one decimal place (i.e., chosen $g = 1$) to keep the error below one.

For *Head*, Fig. 4 shows the result of SR-MPVR, Fig. 5 shows the result of SR-MSSS, and Fig. 6 shows the result of SR-RSS from single view point. As SR-RSS represents three color components of a pixel by a single value, it creates a grey share image, hiding the color information. For SR-MPVR, Fig. 7 demonstrates the secret image and first share image of *Foot*, *Bucky*, and *IronProt* from single view point, and Fig. 8 shows the share images of *Head* for multiple view points (we have provided more results as supplementary material in the file *MoreResult.pdf*). As illustrated by these figures, the color information of the secret image is hidden in the respective share images. Therefore, an adversary having access to a share image cannot perceptually infer the color coded information of the secret image.

7.1 Security analysis

In addition to perceptual security, our schemes also provide data confidentiality, data integrity, and data availability.

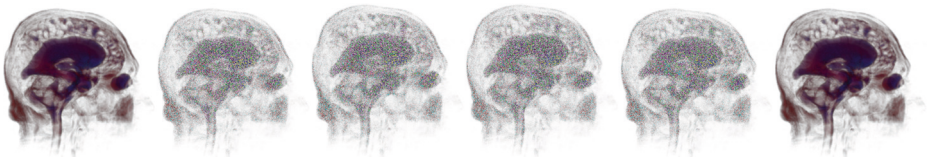

Fig. 4 Secure pre-classification volume ray-casting on *Head* for SR-MPVR

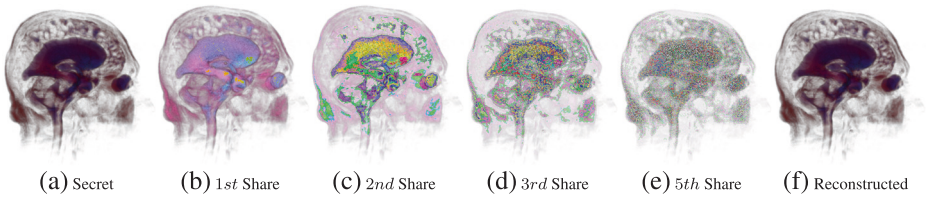

Fig. 5 Secure pre-classification volume ray-casting on *Head* for SR-MSSS

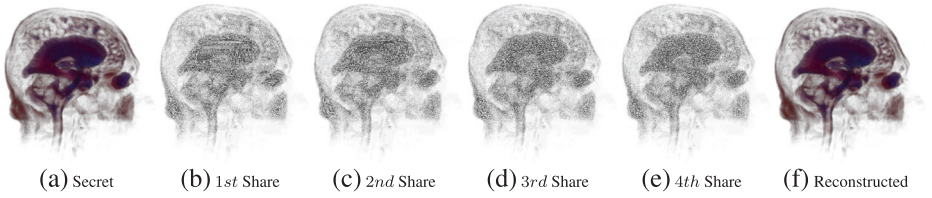

Fig. 6 Secure pre-classification volume ray-casting on *Head* for SR-RSS

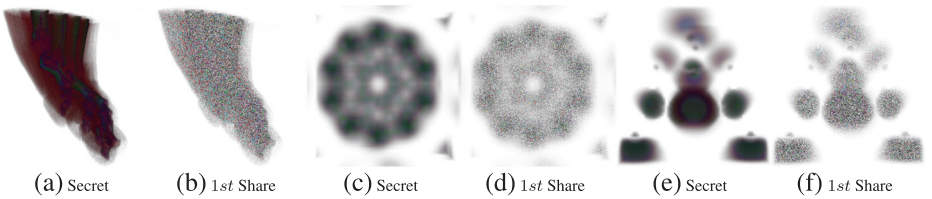

Fig. 7 Secure pre-classification volume ray-casting of *Foot* (**(a)**, **(b)**), *Bucky* (**(c)**, **(d)**), and *IronProt* (**(e)**, **(f)**) for SR-MPVR

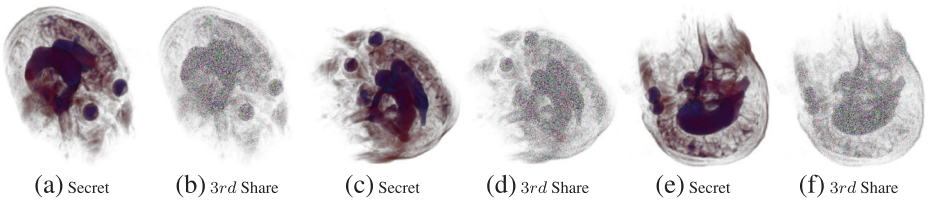

Fig. 8 Secure pre-classification volume ray-casting of *Head* in SR-MPVR from different viewpoints

7.1.1 Confidentiality

As SR-MPVR uses Shamir's secret sharing, it is perfectly secure. Thus, an adversary, irrespective of its computation power, cannot get any information about the secret color of a voxel/pixel by accessing at most $k-1$ datacenters.

In principle, the color of an object is independent of the shape of the object since a user can modify color from a color look-up table. Therefore, theoretically, the probability of guessing the secret color of a voxel/pixel is $\frac{1}{256}$: due to which, the probability of guessing a 512×512 secret image is $\frac{1}{256^{512\times512}}$. However, practically, the range of color of an object can be less than 256. In this case, the knowledge of color adds little information unless there is some color-coded abnormality in the image (such as detection of diseases in a MRI scan). Our work hides such confidential color-coded information.

By excluding modular prime operation from secret sharing, SR-MSSS looses some information about the secret color in a group of less than k datacenters. The probability of knowing a secret from a group of less than k shares increases with an increase in the number of known shares, and this probability depends on the share number x (as there is a one-to-one mapping between the share and the share number). For example, for known share number x_i and the knowledge of one $F'(x_i)$, if we let the probability of knowing the secret pixel color C be $\frac{1}{T}$, then for $k=2$,

$$T = \left\lfloor \frac{F'(x_i)}{x_i} \right\rfloor + 1,$$

For $k=3$,

$$T = \sum_{a=0}^{\left\lfloor \frac{F'(x_i)}{x_i^2} \right\rfloor} \left(\left\lfloor \frac{F'(x_i) - ax_i^2}{x_i} \right\rfloor + 1 \right). \tag{20}$$

When k increases, T increases, as the combination of $k-1$ coefficients that satisfy a $k-2$ degree polynomial $F'^{,k-2}(x, S)$ is also part of the combination of k coefficients that satisfy the $k-1$ degree polynomial $F'^{,k-2}(x, S) + a_{k-1}x^{k-1}$.

To minimize the effect of loss of information, we can choose higher valued random numbers (i.e., a_i's) as coefficients in the secret sharing polynomial to obtain higher share value for lower share number. For example, even for the $(2, n)$ modified secret sharing, if $a_i > 256$ and $x < 5$, then $T > 256$. In other words, we can provide more than 256 choices to an adversary for guessing the secret – already more than the number of color values possible.

By using (3, 4, 5) modified ramp secret sharing, SR-RSS, in addition to losing information due to the exclusion of modular prime operation, also loses information due to the use of multiple secrets in a secret sharing polynomial. Due to this information loss, an adversary, by accessing more than one datacenters, can easily guess some of the secret color by converting the (3, 4, 5) modified ramp secret sharing to (3, 3, 5) modified ramp secret sharing. Therefore, SR-RSS is insecure when an adversary can access more than one datacenter. To counter such scenario, one can easily adjust SR-RSS by introducing $(3, k+3, n)$ ramp secret sharing, where k is the maximum number of datacenters that an adversary cannot access simultaneously.

7.1.2 Integrity

By inheriting the property of (k, n) secret sharing, SR-MPVR, SR-MSSS, and SR-RSS ensures integrity of data/image. The $k < n$ condition provides $\binom{n}{k}$ different ways of reconstructing the secret image. Suppose an adversary changes the color values of the share images (either directly tampering with the rendered image or tampering the share volume) of at most $n - 1$ datacenters, then the reconstructed images will differ to each other (as shown in Fig. 9). As a result, by comparing at most $\binom{n}{n-1} + 1$ reconstructed images (the worst case happens when an adversary tampers only one share image and the client uses the tampered share image in image reconstruction after exploring all other possibilities), the client can detect tampering.

However, if the adversary is able to temper with the share images of all n datacenters by obeying the homomorphic property of secret sharing, then all the tampered reconstructed image at the client site will be the same (as shown in Fig. 9d and e). In this case, the client will not be able to detect the tampering. Furthermore, if $n = k$, then tampering with even one share image is not detectable as there can be only one possible combination to recover the secret image. Therefore, for application requiring data integrity, we recommend choosing $n > k$.

7.1.3 Availability

By inheriting the property of (k, n) secret sharing, all of SR-MPVR, SR-MSSS, and SR-RSS also ensure data availability as the client is able to reconstruct the secret image even if at most $n - k$ number of datacenters are unable to participate.

7.2 Performance analysis

The usability of the proposed cloud-based secured rendering techniques depends on its computational overhead, data overhead, and the quality of rendered image. The computational overhead includes creation of n share from the secret data voxels at the server and reconstruction of the secret image from k share images at the client. The data overhead includes extra bandwidth by the server to transmit n share volumes to n datacenters extra bandwidth to transmit k share images to the client. Computation of share volumes and their distribution to datacenters is performed by the server offline and therefore is less of a concern. The data overhead in transmitting k share images and the computational overhead to reconstruct the secret image, however, add to the latency in rendering. We discuss them below.

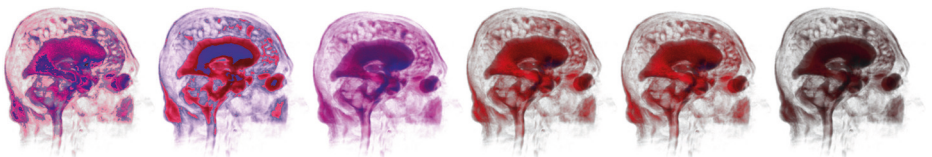

Fig. 9 **(a)**- recovered image when one share image is tampered; (**(b)**, **(c)**) - recovered images when two share images are tampered; (**(d)**, **(e)**) - recovered images when all share images are tampered; and **(f)** - recovered image when no share image is tampered

7.2.1 Computational overhead

The computation cost of our framework is equal to the computation cost required to recover colors of secret image from the share images. Therefore, computational overhead is dependent on the computation cost of Lagrange interpolation and the dimension of the image. As SR-MPVR uses modular prime operation and large integer operations, its computation cost is more than the computation cost of SR-MSSS and SR-RSS. The computation cost of SR-MSSS, however, varies with the computation cost of SR-RSS according to their requirement of the number of shares to reconstruct the secret. In our implementation, SR-MPVR and SR-MSSS takes 132 ms and 29 ms respectively to recover a 512×512 rendered image from the first, second, and third image shares. For the same image, SR-RSS takes 34 ms to recover the secret image from the first, second, third, and fourth share images. The overhead of SR-RSS is more than SR-MSSS as the constructed polynomial for SR-RSS is one degree higher than that of SR-MSSS.

7.2.2 Data overhead

For SR-MPVR and SR-MSSS, our rendering framework requires k share images to reconstruct the secret image. Thus, if b number of bits are required to represent a color component of a pixel of a share image, then a total of $3bk + 8$ number of bits are required to reconstruct the color and opacity of a pixel: due to which, the data overhead is $\frac{3bk-24}{32}$ times more than conventional server-side rendering. The value of b, however, is dependent on how we solve the incompatibility issue of secret sharing with ray-casting. In case of SR-MPVR, b is dependent on the rounding off parameters d and f as the rendered color lays between 0 to q, where $q > (255 + \epsilon_{max}) \times 10^{d+f}$. For SR-MSSS, b, however, is equivalent to the number of bits required to represent a floating point number. Therefore, in our implementation, which uses $(3, n)$ secret sharing, 32 bits for a floating point number, and sets $d = 4$ and $f = 6$, the data overhead of SR-MPVR and SR-MSSS are approximately 11 times and 8 times more than the conventional server-side volume ray-casting.

In the case of SR-RSS, our rendering framework, however, requires four color shares to recover the three color components of a pixel of the secret image. Thus, if b bits are required to represent a color share, then a total of $4b+8$ bits are required to obtain the secret color and opacity value of a pixel in the secret image. We, however, know that the value of a color share cannot exceed 65536×10^g, where g is the number of decimal places by which a share is rounded off. Therefore, our implementation of SR-RSS, which sets $g = 1$,

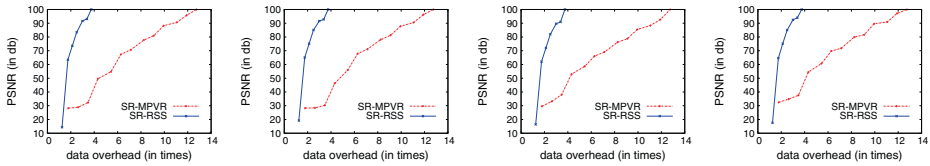

Fig. 10 For SR-MPVR and SR-RSS, this figure shows the PSNR value of the *Head*, *Foot*, *Bucky*, and *IronProt* rendered images for different data overhead. As SR-MPVR results different PSNR for a fixed overhead (as for a fixed number of rounding bits $d + f$, the PSNR can change with the change in the value of d and f), we show the maximum obtained PSNR

Table 2 Comparison among SR-MPVR, SR-MSSS, and SR-RSS

	SR-MPVR	SR-MSSS	SR-RSS
Security	Perfect security	Some information loss	More information loss than SR-MSSS
Data Overhead	Dependent on rounding bits. Typically, High	Moderate	Dependent on rounding bits. Typically, Low
Computation Cost	High	Low	Moderate
Image Quality	Lossy	Lossless	Lossy

results in approximately twice the data overhead than the conventional server-side volume ray-casting.

7.2.3 Image quality

By rounding off floating point numbers during rendering, both SR-MPVR and SR-RSS renders lossy image; but SR-MSSS, without performing rounding operations, renders lossless image.

In the case of SP-MPVR and SR-RSS, we can, however, obtain better quality image by using higher precision fixed point numbers. As discussed in Section 7.2.2, a higher number of rounding bits can increase the data overhead: resulting in a tradeoff between the quality of image and the data overhead. This claim can be verified from Fig. 10, which, for our experimental setup, shows the PSNR values of *Head*, *Foot*, *Bucky*, and *IronProt* rendered images for different overheads. As illustrated, to obtain similar quality image, SR-MPVR leads to higher overhead than SR-RSS as SR-MPVR rounds off two floating point numbers in contrast to only one in SR-RSS. Note that for any of these schemes, the PSNR values of different images are different for fixed overhead as the round-off error and its effect on final rendered color are dependent on the scalar values of the data voxels.

Table 2 shows the security level, overheads, and image quality of all our proposed schemes: SR-MPVR, SR-MSSS, and SR-RSS. As highlighted, (i) SR-MPVR is best suit for applications prioritizing security over overheads and image quality; (ii) SR-RSS is suitable for applications requiring low overhead at cost of high security and loss of information from the rendered image; and (iii) SR-MSSS is designed for applications requiring lossless rendered image, and moderate security and overhead.

8 Conclusion

Rendering important data in third party cloud datacenters presents security and privacy challenges. In this paper, we addressed these challenges by proposing a secure cloud-based rendering framework that, by integrating Shamir's secret sharing (or its variant) with pre-classification volume ray-casting (or its variant), can hide color of volume data or rendered image from datacenters. To address the incompatibility of secret sharing with ray-casting,

we modified either secret sharing (which is the case in SR-MSSS) or ray-casting (which is the case in SR-MSSS), and to decrease high data overhead of both these techniques, we optimized modified secret sharing (which is the case in SR-RSS). Experiments and analyzes showed that SR-MPVR provides high security at the cost of high data overhead and loss of some information from the rendered image; SR-MSSS renders lossless image and provides moderate security at the cost of moderate overheads; and SR-RSS incurs low overhead at the cost of high security and the loss of some information from the rendered image.

We believe that the philosophy of this work, i.e., using a somewhat homomorphic cryptosystem to hide some operations of an algorithm, and distributing the non-homomorphic operations among more than one participants such that none of the participant can get sufficient secret information, can be the basis of designing practical secure cloud based imaging frameworks for other applications in future. In the place of secret sharing, other cryptosystems also can be used to serve any particular requirement.

Acknowledgment This research was supported by Singapore Ministry of Education Academic Research Fund Tier 1 No: $T1251RES1213$ (Secure and Efficient Remote 3D Rendering). Majority of this work was done when the first author, Manoranjan Mohanty, was a PhD student in Department of Computer Science, School of Computing, National University of Singapore. Dr. Atrey's contribution was supported in parts by the NSERC Canada discovery grant number 371714 and the University at Albany grant number 640075.

Appendix A: Example of conventional pre-classification ray-casting and SR-MPVR

In this section, we will show the working of conventional ray-casting and SR-MPVR with a simple example. Since SR-MSSS and SR-RSS are similar to SR-MPVR, we do not provide examples for these two schemes.

In our example, we will first run through the conventional pre-classification ray-casting, and then will discuss SR-MPVR. We will show that the colors rendered by SR-MPVR is almost equal to the colors rendered by the conventional algorithm.

Conventional pre-classification volume ray-casting

As explained in Section 4.1 and shown in Fig. 1, the conventional volume ray-casting algorithm steps can be divided into the following steps: pre ray-projection, ray-projection, sampling, interpolation, and composition. We discuss these steps with an $8 \times 8 \times 8$ input volume data V.

Pre ray-projection

This step finds the shaded colors and opacities of the voxel values, and stores them in a look up table. For our example, let us assume that the colors and opacities of the voxels are stored in Table 3.

Ray-projection

In this step, a number of rays are projected to V from the image space. In the image space, the colors and opacity of a pixel are the colors and opacity rendered along the ray originated

Table 3 Color look up table for V

Coordinate	Red (R)	Green (G)	Blue (B)	Opacity (A)
0,0,0	120	252	3	0.7
0,0,1	5	45	9	0.1
...	...	...	...	...
3,3,3	37	44	211	0.4
3,3,4	37	45	210	0.3
3,4,3	45	45	120	0.7
3,4,4	45	46	121	0.4
4,3,3	44	46	121	0.1
4,3,3	44	46	120	0.6
4,4,3	45	46	135	0.5
4,4,4	46	245	100	0.25
...	...	...	...	...
5,5,5	105	45	150	0.1
5,5,6	110	50	100	0.4
5,6,5	111	50	98	0.5
5,6,6	110	48	98	0.1
6,5,5	100	50	100	0.5
6,5,6	50	50	100	0.3
6,6,5	50	50	101	0.4
6,6,6	150	50	50	0.3
...	...	...	...	...

from it. Since the rendering along a ray is similar to the rendering along other rays, we discuss about the rendering along one ray X.

Sampling

In our example, let us assume that the projected ray X is sampled at two sample points s_1 and s_2.

Interpolation

The colors and opacities along s_1 and s_2 are found by trilinear interpolation of the colors and opacities of the neighbouring voxels.

In our example, let us assume that the voxels $(3, 3, 3)\ldots(4, 4, 4)$ are the neighbouring voxels of s_1 and the voxels $(5, 5, 5)\ldots(6, 6, 6)$ are the neighboring voxels of s_2. If the interpolating factors of voxels $(3, 3, 3)\ldots(4, 4, 4)$ and $(5, 5, 5)\ldots(6, 6, 6)$ are given as $D_{3,3,3} = 0.125$, $D_{3,3,4} = 0.125$, $D_{3,4,3} = 0.125$, $D_{3,4,4} = 0.125$, $D_{4,3,3} = 0.125$, $D_{4,3,4} = 0.125$, $D_{4,4,3} = 0.125$, $D_{4,4,4=0.125}$ and $D_{5,5,5} = 0.125$, $D_{5,5,6} = 0.125$, $D_{5,6,5} = 0$, $D_{5,6,6} = 0.25$, $D_{6,5,5} = 0.25$, $D_{6,5,6} = 0$, $D_{6,6,5} = 0.125$, $D_{6,6,6} = 0.125$ respectively, the interpolated colors and opacities of s_1 and s_2 can be calculated as $R_{s_1} = 42.875$, $G_{s_1} = 70.375$, $B_{s_1} = 142.25$, $A_{s_1} = 0.406$ and $R_{s_2} = 104.375$, $G_{s_2} = 48.875$, $B_{s_2} = 93.25$, $A_{s_2} = 0.3$ respectively.

Composition

In this step, the colors and opacity along the ray are found by compositing the colors and opacities of the sample points. The composited colors are then truncated to get the rendered color.

Therefore, in our example, the rendered colors and opacity are calculated as $R = 43$, $G = 34$, $B = 68$, and $A = 0.5842$.

SR-MPVR

As discussed in Section 6, our scheme, such as SR-MPVR, is targeted to cloud-based pre-classification rendering. Figure 2 shows the architecture of our scheme. In our scheme, the server performs the pre ray-projection step of ray-casting and then creates shares of output color in data preparation step. A color share and a copy of opacities represents a share volume, which is sent to a datacenter. The datacenters perform post ray-projection operation, such as sampling, interpolation, and classification, on their share volumes and send the rendered share images to the client. Finally, the client recovers the secret image from the shared images by reconstructing the secret colors from the share colors.

Data preparation

In this step, the server first performs pre ray-projection operations on V, and then creates three shares of V using Shamir's secret sharing.

Table 4 Color look up table for V_1

Coordinate	Red (R)	Green (G)	Blue (B)	Opacity (A)
0,0,0	220	352	103	0.7
0,0,1	105	145	109	0.1
...	...	...	...	...
3,3,3	137	144	311	0.4
3,3,4	137	145	310	0.3
3,4,3	145	145	220	0.7
3,4,4	145	146	221	0.4
4,3,3	144	146	221	0.1
4,3,3	144	146	220	0.6
4,4,3	145	146	235	0.5
4,4,4	146	345	200	0.25
...	...	...	...	...
5,5,5	205	145	250	0.1
5,5,6	210	150	200	0.4
5,6,5	211	150	198	0.5
5,6,6	210	148	198	0.1
6,5,5	200	150	200	0.5
6,5,6	150	150	200	0.3
6,6,5	150	150	201	0.4
6,6,6	250	150	150	0.3
...	...	...	...	...

Without loss of generality, let us assume that the server uses (2, 3) Shamir's secret sharing. Therefore, three datacenters performing rendering operations, but shared rendered images from at least two datacentres are required to get the secret image.

Suppose we decide to round off the floating point number in interpolation step by 4 decimal places (i.e., $d = 4$) and the float in composition step by 6 decimal places (i.e., $f = 6$). Then, the server uses the secret sharing polynomial $F(x) = (C + 100x)$ mod 2570000000011, where C is the color. Using this polynomial, three shares of colors are created. The color shares along with copies of opacities are given in Tables 4, 5, and 6, those represent the share volumes V_1, V_2, and V_3 respectively.

Ray-projection

The client projects the same rays those it could have projected to the conventional ray-casting, to the data volumes presented by share tables in the datacenters.

Post ray-projection rendering

Post ray-projection steps are as follows.

Sampling: A ray is sampled at the same sample points where it could have been sampled in the conventional ray-casting. Thus, our discussed ray X is sampled at the sample points $s_1(3.5, 3.5, 3.5)$ and $s_2(5.5, 5, 6)$ for all three datacenters.

Table 5 Color look up table for V_2

Coordinate	Red (R)	Green (G)	Blue (B)	Opacity (A)
0,0,0	320	452	203	0.7
0,0,1	205	245	209	0.1
...	...	...	...	...
3,3,3	237	244	411	0.4
3,3,4	237	245	410	0.3
3,4,3	245	245	320	0.7
3,4,4	245	246	321	0.4
4,3,3	244	246	321	0.1
4,3,3	244	246	320	0.6
4,4,3	245	246	335	0.5
4,4,4	246	445	300	0.25
...	...	...	...	...
5,5,5	305	245	350	0.1
5,5,6	310	250	300	0.4
5,6,5	311	250	298	0.5
5,6,6	310	248	298	0.1
6,5,5	300	250	300	0.5
6,5,6	250	250	300	0.3
6,6,5	250	250	301	0.4
6,6,6	350	250	250	0.3
...	...	...	...	...

Table 6 Color look up table for V_3

Coordinate	Red (R)	Green (G)	Blue (B)	Opacity (A)
0,0,0	420	452	303	0.7
0,0,1	305	345	309	0.1
...	...	...	...	...
3,3,3	337	344	511	0.4
3,3,4	337	345	510	0.3
3,4,3	345	345	420	0.7
3,4,4	345	346	421	0.4
4,3,3	344	346	421	0.1
4,3,3	344	346	420	0.6
4,4,3	345	346	435	0.5
4,4,4	346	545	400	0.25
...	...	...	...	...
5,5,5	405	345	450	0.1
5,5,6	410	350	400	0.4
5,6,5	411	350	398	0.5
5,6,6	410	348	398	0.1
6,5,5	400	350	400	0.5
6,5,6	350	350	400	0.3
6,6,5	350	350	401	0.4
6,6,6	450	350	350	0.3
...	...	...	...	...

Interpolation: In this step, each datacenter finds the color and opacity of the sample points by interpolating the color shares and opacities given in its share table. Since we convert the interpolating factors by rounding off them by 4 decimal places, the interpolating factors of voxels (3, 3, 3)...(4, 4, 4) and (5, 5, 5)...(6, 6, 6) are calculated as $D^{(4)}_{3,3,3} = 1250$, $D^{(4)}_{3,3,4} = 1250$, $D^{(4)}_{3,4,3} = 1250$, $D^{(4)}_{3,4,4} = 1250$, $D^{(4)}_{4,3,3} = 1250$, $D^{(4)}_{4,3,4} = 1250$, $D^{(4)}_{4,4,3} = 1250$, $D^{(4)}_{4,4,4} = 1250$, and $D^{(4)}_{5,5,5} = 1250$, $D^{(4)}_{5,5,6} = 1250$, $D^{(4)}_{5,6,5} = 0$, $D^{(4)}_{5,6,6} = 2500$, $D^{(4)}_{6,5,5} = 2500$, $D^{(4)}_{6,5,6} = 0$, $D^{(4)}_{6,6,5=1250}$, $D^{(4)}_{6,6,6} = 1250$ respectively. Using these interpolating factors, the first datacenter, second datacenter, and third datacenter calculate the colors of s_1 and s_2 as ($R_{1,1} = 1428750$, $G_{1,1} = 1703750$, $B_{1,1} = 2422500$, $R_{2,1} = 2043750$, $G_{2,1} = 1488750$, $B_{2,1} = 1932500$), ($R_{1,2} = 2428750$, $G_{1,2} = 2703750$, $B_{1,2} = 3422500$, $R_{2,2} = 3043750$, $G_{2,2} = 2488750$, $B_{2,2} = 2932500$), and ($R_{1,3} = 3428750$, $G_{1,3} = 3703750$, $B_{1,3} = 4422500$, $R_{2,3} = 4043750$, $G_{2,3} = 3488750$, $B_{2,3} = 3932500$) respectively.

Interpolation: In this step, the colors and opacity along the ray are found by compositing colors and opacities of the sample points. Each datacenter composites its colors and opacities from the interpolated colors and opacities available to it. In our example, $f = 6$. Using this value, the first datacenter, the second datacenter, and the third datacenter calculate their composite colors as ($R'_1 = 1019175750000$, $G'_1 = 930830750000$, $B'_1 = 1268224500000$), ($R'_2 = 1603375750000$, $G'_2 = 1515030750000$, $B'_2 = 1852424500000$), and ($R'_3 = 2187575750000$, $G'_3 = 2099230750000$, $B'_3 = 2436624500000$) respectively. For all datacenters, the composited opacity is 0.5842.

Image recovery

In this step, the client obtains the secret colors and opacities from the share colors and opacities obtained from three datacenters. Since each datacenter has a copy of the secret rendered opacity, the opacity of a pixel is the rendered opacity along that ray at any datacenter. Therefore, for Ray X, the opacity is 0.5842, which is equal to the rendered opacity along that ray by the conventional pre-classification ray-casting.

The secret colors, however, are found by first reconstructing (using Lagrange interpolation) a value from the share colors, and then dividing the value by 10000000000. Since we use (2, 3) secret sharing, we need at least two shares to obtain the secret. Without any loss of generality, we choose the shares of first and second datacenters, and find the rendered color as $R = 43$, $G = 34$, and $B = 68$.

Appendix B: List of symbols

Table 7 Symbol table

Symbol	Meaning
a_i	i^{th} coefficient in secret sharing polynomial
c	Total number of sample points along a projected ray
d	Number of decimal places by which D_v is rounded off
f	Number of decimal places by which O_i is rounded off
g	Number of decimal places by which C' is rounded off
k	Minimum number of shares required to construct a secret in Shamir's secret sharing
$l_i(x)$	Lagrange basis function
n	Total number of shares created from a secret in Shamir's secret sharing
p	Share number
q	Prime number
s	A sample point in a secret data volume V
s_p	A sample point in the p^{th} share of a data volume V
v	A data voxel in a secret data volume V
v_p	A data voxel in the p^{th} share of a data volume V
A_i	Opacity of i^{th} voxel of i^{th} sample point
A	Opacity of a paxel composited by the conventional ray-casting or by the modified ray-casting
C_i	Color of i^{th} voxel or i^{th} sample point
C	Composited color of a pixel rendered by the conventional ray-casting
$C_{v,p}$	p^{th} share of the color of a data voxel v
C'_S	Scaled interpolated color of a sample point s rendered by the modified ray-casting
C'_p	p^{th} share of the composited color of a pixel rendered by the modified ray-casting
$C'_{s,p}$	p^{th} share of scaled interpolated color of sample point s
C'	Scaled composited color rendered by modified ray-casting
D_i	Interpolation factor of i^{th} voxel or i^{th} saple point
$X^{(d)}$	Fixed point representation of a float X
$\epsilon_{x,y}$	Roundoff error due to rounding off x by y decimal places

References

1. AlZain MA, Pardede E, Soh B, Thom JA (2012) Cloud computing security: From single to multi-clouds. In: Proceedings of the 45^{th} Hawaii International Conference on System Sciences Hawaii, pp 5490–5499
2. Benaloh JC (1987) Secret sharing homomorphisms: Keeping shares of a secret secret. In: Proceedings of the Advances in Cryptology–CRYPTO'87, Sanata Barbara, pp 31–36
3. Catrina O, Saxena A (2010) Secure computation with fixed-point numbers. In: Proceedings of the 14th international conference on financial cryptography and data security, Tenerife, pp 35–50
4. Chor B, Kushilevitz E (1993) Secret sharing over infinite domains. J Cryptol 6:87–95
5. Cooper J, Donovan D, Seberry J (1994) Secret sharing schemes arising from Latin sqaures. Bull Inst Comb Appl 12:33–43
6. Dorn K, Ukis V, Friese T (2011) A cloud-deployed 3D medical imaging system with dynamically optimized scalability and cloud costs. In: Proceedings of the 37^{th} EUROMICRO conference on software engineering and advanced applications, Oulu, pp 155–158
7. Fellgiebel A, Müller MJ, Wille P et al (2005) Color-coded diffusion-tensor-imaging of posterior cingulate fiber tracts in mild cognitive impairment. Neurobiol Aging 6:1193–1198
8. Finamore T (2012) Shamir's secret sharing scheme using floating point arithmetic. Master Thesis, Florida Atlantic University
9. Harn L, Changlu L (2010) Authenticated group key transfer protocol based on secret sharing. IEEE Trans Comput 59:842–846
10. Henry K (2008) The theory and applications of homomorphic cryptography. Master Thesis
11. Kamara S, Lauter K (2010) Cryptographic cloud storage. In: Proceedings of the 14^{th} international conference of financial cryptography and data security: Workshop on Real-Life Cryptographic Protocols and Standardization Canary Islands, pp 136–149
12. KDDI Inc. (2012) Medical real-time 3d imaging solution. Online Report. http://www.kddia.com/en/sites/default/files/file/KDDI_America_Newsletter_August_2012.pdf
13. Lathey A, Atrey PK, Joshi N (2013) Homomorphic low pass filtering on encrypted multimedia over cloud. In: Proceedings of the 7th IEEE international conference on the semantic computing, Irvine, pp 310–313
14. Levoy M (1988) Display of surfaces from volume data. IEEE Comput Graph Appl 8:29–37
15. Mather T, Kumaraswamy S, Latif S (2009) Cloud security and privacy: An enterprise perspective on risks and compliance. O'Reilly Media Inc.
16. Mohanty M (2013) Secret sharing approach for securing cloud-based image processing. PhD Thesis
17. Mohanty M, Atrey PK, Tsang Ooi W (2012) Secure cloud-based medical data visualization. In: Proceedings of the 20th ACM international conference on Multimedia, Nara, pp 1105–1108
18. Mohanty M, Tsang Ooi W, Atrey PK (2013) Secure cloud-based volume ray-casting. In: Proceedings of the IEEE international conference on cloud computing technology and services, Bristol
19. Naehrig M, Lauter K, Vaikuntanathan V (2011) Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on cloud computing security workshop, Chicago, pp 113–124
20. NICE (2011) Desktop cloud visualization. Online Report. http://www.nice-software.com/products/dcv
21. NVIDIA (2009) Realityserver 3.0 white paper. Online Report. http://www.mentalimages.com/fileadmin/user_upload/PDF/RealityServer_White_Paper1212.pdf
22. Parsonson L, Grimm S, Bajwa A, Bourn L, Bai L (2012) A cloud computing medical image analysis and collaboration platform. In: Cloud Computing and Services Science. Springer, New York, pp 207–224
23. SaghaianNejadEsfahani SM, Luo Y, Cheung SCS (2012) Privacy protected image denoising with secret shares. In: Proceedings of the 19th IEEE international conference on image processing, Orlando, pp 253–256
24. Shamir A (1979) How to share a secret. Commun ACM:612–613
25. Sinha System (2012) Cloud based medical image management and visualization platform. Online Report. http://www.shina-sys.com/assets/brochures/3Di.pdf
26. Smelyanskiy M, Holmes D, Chhugani J (2009) Mapping high-fidelity volume rendering for medical imaging to CPU, GPU and many-core architectures. IEEE Trans Vis Comput Graph 15:1563–1570
27. Tharaud J, Wohlgemuth S, Echizen I et al (2010) Privacy by data provenance with digital watermarking. In: Proceedings of the 6th international conference on intelligent information hiding and multimedia signal processing, Darmstadt, pp 510–513
28. Vazhenin D (2012) Cloud-based web-service for health 2.0. In: Proceedings on joint international conference on human-ventered computer environments, Hamamatsu, pp 240–243
29. Zissis D, Lekkas D (2012) Addressing cloud computing security issues. Future Gener Comput Syst 28:583–592

Manoranjan Mohanty is a postdoctoral fellow in the Security Lab of Swedish Institute of Computer Science (SICS Swedish ICT), where he is employed under ERCIM *Alain Bensoussan* research fellowship program. He obtained PhD from Department of Computer Science, National University of Singapore in 2014. Broadly, his research interest includes cloud-based outsourcing, hidden domain processing, applied security and privacy, and multimedia computing.

Wei Tsang Ooi received the BSc (Hon.) degree from the National University of Singapore in 1996, and the PhD degree in computer science from Cornell University, in 2001. He spent a year as postdoc at Berkeley Multimedia Research Center in U.C. Berkeley, before rejoining NUS in 2002, where he is currently an associate profes- sor in the Department of Computer Science. His research interests include interactive multimedia systems, including zoomable videos and net- worked graphics.

Pradeep K. Atrey is an Assistant Professor at the State University of New York, Albany, NY, USA. He is also an (on-leave) Associate Professor at the University of Winnipeg, Canada and an Adjunct Professor at University of Ottawa, Canada. He received his Ph.D. in Computer Science from the National University of Singapore, M.S. in Software Systems and B.Tech. in Computer Science and Engineering from India. He was a Postdoctoral Researcher at the Multimedia Communications Research Laboratory, University of Ottawa, Canada. His current research interests are in the area of Security and Privacy with a focus on multimedia surveillance and privacy, multimedia security, secure-domain cloud-based large-scale multimedia analytics, and social media. He has authored/co-authored over 90 research articles at reputed ACM, IEEE, and Springer journals and conferences. His research has been funded by Canadian Govt. agencies NSERC and DFAIT, and by Govt. of Saudi Arabia. Dr. Atrey is on the editorial board of several journals including ACM Trans. on Multimedia Computing, Communications and Applications, ETRI Journal and IEEE Communications Society Review Letters. He was also guest editor for Springer Multimedia Systems and Multimedia Tools and Applications journals. He has been associated with over 30 international conferences/workshops in various roles such as General Chair, Program Chair, Publicity Chair, Web Chair, Demo Chair and TPC Member. Dr. Atrey was a recipient of the Erica and Arnold Rogers Award for Excellence in Research and Scholarship (2014), ETRI Journal Best Editor Award (2012), ETRI Journal Best Reviewer Award (2009) and the three University of Winnipeg Merit Awards for Exceptional Performance (2010, 2012 and 2013). He was also recognized as ICME 2011 - Quality Reviewer.