CloudSafe: Securing Data Processing within Vulnerable Virtualization Environments in the Cloud

Huijun Xiong[†] Qingji Zheng[‡] Xinwen Zhang[⋆] Danfeng (Daphne) Yao[†]
[†]Department of Computer Science, Virginia Tech, VA, USA [⋆]Huawei Research Center, CA, USA
[‡]Department of Computer Science, University of Texas at San Antonio, TX, USA

***Abstract*—Data protection in public cloud remains a challenging problem. Outsourced *data processing* on vulnerable cloud platforms may suffer from cross-VM attacks, e.g. side-channel attacks that leak secrecy keys. We design and develop CloudSafe, a general and practical data-protection solution by integrating *cryptographic techniques and systematic mechanisms* seamlessly to address this issue. CloudSafe first allows a data owner to outsource encrypted data in the cloud. It then employs a cloud-based proxy to re-encrypt stored encrypted data and delivers it to authorized cloud applications upon access requests. To combat cross-VM side-channel attacks, the final data decryption key is one-time use and can be retrieved from the data owner on-demand. Any key leakage after an authorized access cannot compromise data confidentiality. For data sharing, CloudSafe allows authorized applications to efficiently access the protected data. The prototype evaluation demonstrates the efficiency of the scheme towards large-scale cloud applications.**

***Index Terms*—cloud security, outsourced computation, side-channel attack, proxy re-encryption, one-time key.**

I. Introduction

Without physical possession of outsourced data, cloud customers usually rely on cryptographic techniques to protect and control their outsourced data. Since fully homomorphic encryption (FHE) techniques are still not practical for real applications, current data processing in the cloud can only be performed on the plaintext of encrypted data. However, newly discovered vulnerabilities in cloud virtualization environment have threatened the security of using cryptographic techniques [7], [10], [20], [21], [32]. For example, researchers in [20], [32] have demonstrated the possibility of performing cross-VM (virtual machine) side-channel attacks to steal cryptographic keys on a well-known existing public commercial cloud environment. Researchers in [7], [10], [21] have found that security misconfiguration and malicious software are intentionally left in the public shared VM images. Those deliberate vulnerabilities would expose users' sensitive data once the images are reused by other cloud users. Although a security-sensitive cloud customer may build a clean virtual machine from scratch to avoid such potential security risks, it is difficult to escape from eavesdropping through hidden side-channels set up by malicious neighbors on the same cloud platform.

Previous work aiming to address cross-VM side-channel attacks against cryptographic systems focuses on a single host machine, e.g., designing patternless cryptographic operations [9], [17], [18], [23] to secure cryptographic key usage. However, due to the pervasive and stealthy characteristics of side-channels, directly blocking all side-channels of a system is difficult, especially for a system running in the public cloud. To our best knowledge, no existing solutions are effective enough to protect the usage of cryptographic keys from side-channels in the cloud. To provide complete data security in the current vulnerable cloud environment, it has been recognized that cryptography alone is not enough [24].

In this work, we introduce *CloudSafe*, a new approach that offers desirable security properties for a data owner to secure and control encrypted outsourced data in the public cloud with the presence of semi-trusted cloud service providers and malicious cloud customers. CloudSafe puts special emphasis on new security issues raised from *cross-VM side-channel attacks* against *secure data processing* on the virtualized cloud platforms.

Specifically, we observe that stealing behaviors involved in side-channel attacks are usually accomplished only after a secret, such as a data decryption key, is loaded into the victim machine's physical component, e.g., CPU cache, and computed. With this observation, our solution against these attacks is to use one-time secret keys in victim machines: each secret key is discarded once it is been used. We believe that, although this may not completely address side-channel attacks, it minimizes the damages caused by the attacks and thus enhances the security of cryptographic key usage in the vulnerable cloud. However, this approach prompts two major challenges: (1) complicated key management due to large amount of one-time data decryption keys; (2) heavy computation requirements over outsourced encrypted data by the data owner due to frequent updates for data decryption keys. Without overcoming these two challenges, it is impractical to use one-time key in the current public cloud environment.

In response, CloudSafe proposes two techniques to address these challenges. First, CloudSafe defines a flexible and efficient cryptographic scheme, which supports one-time data decryption key strategy and assures that outsourced data is not only kept confidential in the public cloud but also can be accessed with different one-time data decryption keys without imposing heavy computation on the data owner side. Particularly, CloudSafe extends CloudSeal [28], a dual-layer cryptographic scheme, to provide strong confidentiality to outsourced data via specific encryption algorithms and flexible access control over outsourced data with re-encryption technique. When an authorized entity wants to decrypt outsourced encrypted data in the cloud with a new data decryption

key, only a small part of the encrypted data needs to be modified, which significantly reduces the overall computation and makes encrypted data flexible enough to support one-time data decryption key strategy.

Second, CloudSafe leverages a novel centralized key distribution framework inside the public cloud environment to assist storing and distributing one-time data decryption keys. A data owner first distributes a key into the framework within the public data center, and then a trusted key server distributes the key from the framework to VMs through the internal network of the cloud. Through the entire process, the key management framework is responsible for two aspects: (1) the data decryption key is securely stored in public cloud; (2) only authorized computing VMs are able to access the data decryption key from anywhere in the data center. These features enable CloudSafe to protect the secure usage of cryptographic keys in the vulnerable virtualization environment and flexibly adapt to dynamic VM migration activities in the public cloud.

We have implemented a prototype of CloudSafe in a computing cluster virtualized with Xen. With extensive analysis and evaluation, we have demonstrated that CloudSafe is secure and efficient enough to protect outsourced data security and support the usage of one-time data decryption keys within the vulnerable virtualization environment.

Roadmap. We present the basic setting for our approach in Section II. We overview CloudSafe in Section III and provide detailed description for the two main components of CloudSafe in Section IV and Section V, respectively. A prototype implementation and its evaluation are described in Section VI. We discuss related work in Section VII and conclude this paper in Section VIII.

II. System Setting

A. Problem Scenario

A typical usage of public cloud infrastructure service, shown in Figure 1, involves three types of participants: infrastructure service provider, cloud customers, and virtual servers.

- *Infrastructure service provider*, also referred as the *cloud provider*, carries infrastructural resources including physical storage devices, computing machines, and network equipment, and provides virtualized resources as services to cloud customers.
- *Cloud customers* purchase infrastructure services on demand. A cloud customer who stores and processes her data in the cloud is called a *data owner*. A cloud customer who consumes the data stored in the cloud is called a *data user*, which can be a data owner at the same time. Data users might be malicious and perform cross-VM side-channel attacks, which are referred as *malicious cloud customers* or *attackers*.
- *Virtual servers* are built by a cloud customer on virtual machines leased by the *cloud provider*. It might host various cloud applications for data processing in the cloud.

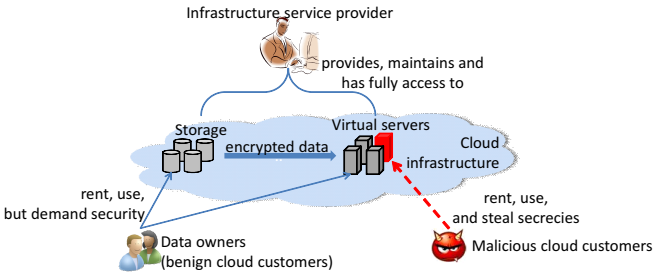

Fig. 1. The data usage scenario in public cloud.

A cloud provider usually arranges multiple virtual servers from different cloud customers on the same physical infrastructure, e.g., to maximize the utility of her resources without cloud customers' awareness. A data owner may store her data in the cloud storage and set up several virtual servers to process them in the cloud. Although it is possible that a virtual server may consume data from different origins, for simplicity we consider a single data owner in our model and regulate the direction of data flow only from the data owner to the cloud storage, and then to virtual servers.

B. Threat Model

We confine the malicious behavior of a cloud customer similar to that in [20], [32], [31]: stealing cryptographic secrets from neighbor victim cloud customers by pre-built hardware-based side-channels. They do not have the control over the victim virtual servers.

Data owners who require confidential cloud services are our victims and fully trusted. The cloud provider is semi-honest, who may attempt to peek the information that is storing in her infrastructure. In addition, we consider that the underlying infrastructure in the cloud including hypervisors and the hardware is not compromised. That is, we do not consider attacks against hypervisors and any insider threat or abuse of cloud administrative rights within a cloud. Specifically for Xen virtualization environment [6], [8], we assume malicious attackers in DomUs cannot penetrate into the Dom0 of a physical machine, i.e., a malicious cloud customer can only mount cross-VM side-channel attacks among virtual servers in different DomUs on the same physical machine.

III. Design Overview

A. Initial Attempts

Simple one-time data decryption key scheme. A straightforward solution to protect cryptographic key usage is to adopt one-time data decryption keys on original data directly. A one-time key is used to encrypt data before it is outsourced to the cloud. After being used by a virtual server, the key is revoked. However, this scheme requires the data owner to remove the used encrypted data and upload the newly ciphertext with a new key to the cloud storage each time the data decryption key is used. Obviously this solution is impractical when frequent and large-scale data decryption operations are needed.

Hypervisor-assisted key usage. In order to avoid heavy computation and large network overhead caused by the simple one-time data decryption keys, an alternative solution to protect cryptographic key usage in virtual servers is to adopt hypervisor-assisted key usage. According to our threat assumption, cross-VM side-channel attacks happen only between virtual servers on the same physical machine, and it is hard for a malicious cloud customer to build side-channels against the hypervisor and other privileged domains. Therefore, moving data decryption operation from a virtual server to the hypervisor using long-term data decryption key is secure enough to protect cryptographic key usage in the cloud virtualization environment, since the key never resides in the vulnerable virtual server environment. However, with the increasing number of virtual servers that are running on the same physical machine, usually from different cloud customers, it is discouraging to execute computation-intensive operations in the hypervisor or privileged domain for performance considerations. Furthermore, dynamically loading guest virtual server's code for data decryption purpose in the privileged domain may introduce new risks to virtualized platforms.

B. Design Strategies

CloudSafe adopts the following strategies to significantly reduce cross-VM side-channel attacks and overcome shortcomings stated above.

Combination of one-time decryption key and proxy re-encryption. The method of simply using one-time data decryption keys incurs heavy computational costs on data owners. We employ the idea of proxy re-encryption. Given a master key, the data owner generates a key pair having specific purpose. One is one-time re-encryption key. It is distributed to the cloud so that the cloud can re-encrypt the ciphertexts to a new one. The other one is one-time data decryption key. It is issued to the data user so that the data user can decrypt the new ciphertext. Note that the one-time data decryption key can be used to decrypt the ciphertext re-encrypted with designated one-time re-encryption key. This strategy enjoys the following advantages:

- The data owner can offload the heavy computational cost to the cloud by leveraging the elastic computational power provided by the cloud. In addition, without uploading new ciphertexts, it dramatically reduces the bandwidth overhead of key revocation process.
- The data owner can directly implement access control on her demands, e.g. by controlling the distribution of one-time data decryption keys and re-encryption keys.

Centralized key distribution framework in the cloud. In order to securely distribute keys to corresponding entities in public cloud, CloudSafe adopts different channels for different keys. First, we deploy a proxy in a dedicated cloud platform, which only runs a single virtual machine of the data owner. With this, the cross-VM side-channel attack is not a threat and the proxy can have a public/private key pair to build secure channel with the data owner. Therefore the re-encryption key can be distributed with this channel securely. For data decryption key, since it travels through public in-cloud network, and we cannot assume a secure channel between a virtual server and the data owner due to cross-VM side-channel attacks in the virtual server side, we develop a centralized key distribution framework by leveraging the relatively secure hypervisor and Dom0 environment on each cloud platform. With this, a secure channel can be built between the data owner and a trusted agent in Dom0, which in turn delivers data decryption keys to local virtual servers without going to public in-cloud network. This mechanism achieves high assurance of key distribution in the public cloud environment.

C. CloudSafe Overview

To embody these design strategies, CloudSafe consists of three main operations: *outsource operation*, *authorization operation*, and *distribution operation*. The first two operations are executed by a data owner with a cryptographic scheme specified for CloudSafe to protect outsourced data confidentiality, and the latter one is carried out by the key management framework of CloudSafe to guarantee secure distribution and usage of data decryption keys by virtual servers.

Outsource operation. An outsource operation provides confidentiality to the outsourced data stored in a cloud storage. During this operation, the data owner takes the original plaintext of her data into the encryption cipher with a secret key and outsources the encrypted data to the cloud storage. The data owner chooses the secret key for each outsourced data and records them locally for authorization usage. This operation is done only once for each data object (e.g., a file) before it is exported from the data owner to the cloud storage.

Authorization operation. An authorization operation prevents unauthorized access to the outsourced data in the cloud storage via one-time decryption keys and re-encryption keys. Upon a data request from an authorized virtual server, the data owner first generates a pair of one-time keys $\langle \mathsf{reencryption\_key}, \mathsf{data\_decryption\_key} \rangle$, and sends them to the proxy service and the virtual server in the cloud, respectively. Then the proxy re-encrypts the data with the received one-time $\mathsf{reencryption\_key}$ and sends the re-encrypted ciphertexts to the virtual server. Note here a pair of one-time keys is generated for each data request, and CloudSafe ensures that only the re-encrypted data can be decrypted by an authorized virtual server in the cloud, leaving the rest of the outsourced data secure in the cloud storage. With efficient outsourced data transformation, CloudSafe is able to support one-time key strategy with affordable computation overhead. Besides, the re-encryption operation offloaded to the proxy further reduces the workload at the data owner side.

Distribution operation. A distribution operation provides secure distribution of one-time key pairs with different channels. The re-encryption key is delivered through pre-built secure channel between the data owner and the proxy. The data decryption key is distributed through a novel centralized key

management framework in the cloud. First, the key is sent by the data owner to a key server through a secure channel over the public network; then the key is delivered from the key server to a requesting virtual server. This key transfer may require the hypervisor's participation.

IV. Cryptographic Scheme

A. Cryptographic Scheme for CloudSafe

Definition The cryptographic scheme in the CloudSafe consists of the following algorithms:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^\lambda)$: This is the bootstrapping algorithm run by a data owner to initialize the system. It outputs a master key mk for the data owner itself and the parameters pm which are made public.
- $\mathsf{cph} \leftarrow \mathsf{PEnc}(\mathsf{pm}, \mathsf{mk}, \mathsf{M})$: This is the data encryption algorithm run by the data owner before outsourcing M to the cloud. It outputs the encrypted version of M.
- $(\mathsf{deckey}, \mathsf{rekey}) \leftarrow \mathsf{KeyPairGen}(\mathsf{pm}, \mathsf{mk})$: This is the algorithm for generating a pair of one-time keys, including a re-encryption key rekey, issued to a cloud-based proxy, and a decryption key, issued to an authorized virtual server where a data processing application runs. It is run by the data owner in order to authorize the virtual server to access the encrypted data correctly. The pair of keys is distributed to the proxy and the virtual server via secure channels, respectively. Note that the decryption key deckey can only decrypt the re-encrypted ciphertext along with rekey.
- $\mathsf{recph} \leftarrow \mathsf{ReEnc}(\mathsf{pm}, \mathsf{cph}, \mathsf{rekey})$: This is the re-encryption algorithm run by the proxy to re-encrypt the stored ciphertext data to another ciphertext with the re-encryption key rekey. It outputs the re-encrypted ciphertext which will be delivered to the authorized virtual server.
- $\{\mathsf{M}, \perp\} \leftarrow \mathsf{Dec}(\mathsf{recph}, \mathsf{deckey}(\mathsf{cph}, \mathsf{mk}))$: This is the decryption algorithm run by an authorized virtual server with the decryption key deckey (or the data owner with the master key mk). It outputs the plaintext data M corresponding to the re-encrypted ciphertext recph, as well as the original ciphertext cph, or outputs error message $\perp$.

The security of this scheme requires that: (1) the master key cannot be leaked, (2) given one key of a key pair (a re-encryption key and a decryption key), the data user and the cloud cannot derive the other under the assumption that data users and the cloud do not collude, and (3) the security of ciphertexts is preserved, i.e., achieving indistinguishability from chosen-plaintext attacks (CPA).

B. Construction for the Cryptographic Scheme

Here we propose an efficient cryptographic scheme satisfying the desirable properties as above.

Let SE be the standard symmetric encryption scheme achieving chosen-plaintext security (CPA), such that $\mathsf{SE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen is a probabilistic key generation algorithm, Enc is a probabilistic encryption algorithm, and Dec is a deterministic decryption algorithm.

- $\mathsf{Setup}(1^\lambda)$: Given a security parameter λ, the data owner chooses a bilinear map $e : G \times G \rightarrow G_T$, where G and G_T are cyclic groups of the order p, an λ-bit prime. Let g be a generator randomly selected from G. Let H be a secure hash function, $H : \{0,1\}^* \rightarrow \{0,1\}^{\lambda_1}$, and F_k be a secure keyed hash function, $F_k : \{0,1\}^{\lambda_1+1} \stackrel{k}{\rightarrow} \mathbb{Z}_p$ where λ_1 is another security parameter and k is the key. The data owner selects a randomly from $\mathbb{Z}_p$ and sets $\mathsf{pm} = (e, g, G, G_T, p, g^a), \mathsf{mk} = (a)$.
- $\mathsf{PEnc}(\mathsf{pm}, \mathsf{mk}, \mathsf{M})$: The data owner selects s randomly from $\mathbb{Z}_p$. Let $\mathsf{M} = (\mathsf{M}_1, \cdots, \mathsf{M}_n) \in G^n$ uniquely identified by the file handler fid, and compute $x = F_{\mathsf{mk}}(H(\mathsf{fid})|0)$ and $y = F_{\mathsf{mk}}(H(\mathsf{fid})|1)$. The data owner encrypts M with SE, s.t. $\mathsf{M}' = (\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_i, \cdots, \mathsf{M}_n)$ where $\mathsf{M}'_i = \mathsf{SE.Enc}(x, \mathsf{M}_i)$ and x is the symmetric key for SE. Then she selects s randomly from $\mathbb{Z}_p$ and generates the ciphertext as $\mathsf{cph} = (g^{ys}, \{\mathsf{M}'_i e(g,g)^s\}_{i=1}^n)$.
- $\mathsf{KeyPairGen}(\mathsf{pm}, \mathsf{mk})$: Given the access request on data M specified by fid, the data owner selects t randomly from $\mathbb{Z}_p$, computes $x = F_{\mathsf{mk}}(H(\mathsf{fid})|0)$ and $y = F_{\mathsf{mk}}(H(\mathsf{fid})|1)$, and generates the decryption and re-encryption keys as $\mathsf{deckey} = (x, t), \mathsf{rekey} = (g^{t/y})$.
- $\mathsf{ReEnc}(\mathsf{pm}, \mathsf{cph}, \mathsf{rekey})$: Given the re-encryption key rekey, the cloud performs the re-encryption on $\mathsf{cph} = (g^{ys}, \{\mathsf{M}'_i e(g,g)^s\}_{i=1}^n)$ as follows: $\mathsf{recph} = (e(g,g)^{ts}, \{\mathsf{M}'_i e(g,g)^s\}_{i=1}^n)$ where $e(g,g)^{ts} = e(g^{ys}, g^{t/y})$
- $\{\mathsf{M}, \perp\} \leftarrow \mathsf{Dec}(\mathsf{recph}, \mathsf{deckey})$: Given the ciphertext $\mathsf{recph} = (e(g,g)^{ts}, \{\mathsf{M}'_i e(g,g)^s\}_{i=1}^n)$ and the decryption key deckey, the data user computes $X = (e(g,g)^{ts})^{1/t}$ and obtains the data $\mathsf{M}' = (\mathsf{M}'_1, \cdots, \mathsf{M}'_n), 1 \leq i \leq n$ by $\mathsf{M}'_i = \mathsf{M}'_i e(g,g)^s / X$. Hence, the data user can decrypt M' with the symmetric key x and gets $\mathsf{M}' = (\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_i, \cdots, \mathsf{M}_n)$ where $\mathsf{M}'_i = \mathsf{SE.Enc}(x, \mathsf{M}_i)$ and x is the symmetric key for SE. Similarly, given cph and the master key mk, the decryption can be done.

Note that in the scheme each M is encrypted with a distinct secret key, determined together by the master key and the hash value of the data handler. This simplifies the key management since only the master key mk should be kept private. On the other hand, distinct secret key for each data can minimize the attack surface if the secret key corresponding to some data was compromised.

We also note that the one-time decryption key deckey and re-encryption key rekey play vital role in enforcing access control policies for the data owner. First, only with one key from the key pair $(\mathsf{deckey}, \mathsf{rekey})$, neither the cloud nor data users can access any information from the stored data in the cloud. Second, given a key pair $(\mathsf{deckey}, \mathsf{rekey})$, deckey can only be used to decrypt the ciphertexts corresponding to rekey. This achieves the isolation among data users in the sense

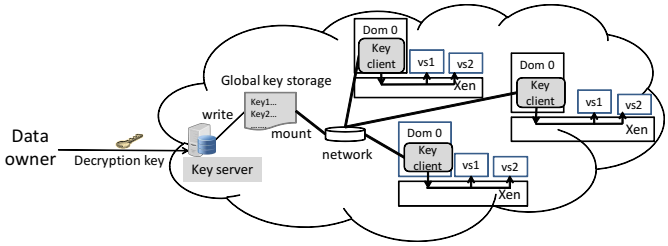

Fig. 2. The overview of key management framework.

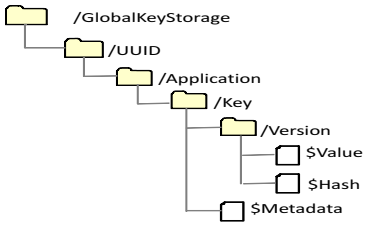

Fig. 3. Cascade structure of the global key storage.

that one data user cannot decrypt the ciphertexts which is not designated for him. Through this, this scheme enables the data owner to realize flexible access control easily by generating and distributing (deckey, rekey).

This scheme achieves chosen-plaintext security for ciphertexts generated by PEnc, given that SE is a CPA secure symmetric encryption scheme and the Decisional Bilinear Diffie-Hellman assumption is hard. The master key and the pair of one-time keys are also secure given the Decisional Bilinear Diffie-Hellman assumption. We prove these security properties in Appendix.

V. Key Management

To securely store and distribute one-time data decryption keys from data owners to virtual servers, CloudSafe relies on a centralized in-cloud key management framework. In this section, we demonstrate the design of this framework within the context of Xen virtualization environment [6], [8] as Xen is a popular hypervisor that has been used in real cloud environments. Usually, a Xen virtualized platform contains two types of domains: one Dom0 and multiple DomUs. The Dom0 is the control domain with high privileges, while DomUs are guest domains for virtual servers rented by cloud customers. As DomUs are insecure due to potential cross-VM side-channel attacks, we utilize Dom0 on each physical machine to enhance the security of data decryption key distribution and usage in DomUs.

A. Overview

Figure 2 depicts the overview of our key management framework, which consists of three main components: the key server, the global key storage, and multiple key clients.

The key server runs in a dedicated machine instead of a rented virtual machine in the cloud, e.g., a dedicate server provided by Amazon EC2 [1] or GoGrid [2], which run applications for a single cloud customer. Usually a data owner deploys one key server in the cloud, which provides the connection between the data owner (or application running as the data owner) and all virtual servers rented by the data owner. It has its own public/private key pair $\langle PK_{keyserver}, SK_{keyserver} \rangle$ to set up secure channel with the data owner. It listens on a certain port to accept one-time data decryption keys from the data owner and writes the received data decryption keys to the global key storage for future access from virtual servers.

The global key storage is a centralized data storage attached to the key server physically and remotely accessible by key clients (virtual servers) over the cloud network. It stores one-time data decryption keys on behalf of each virtual server in a cascade structure as shown in Figure 3. The `UUID` is the unique identifier of each virtual server in the cloud, which is generated when a virtual server is created by the hypervisor on a physical machine. As `UUID` is not changed during the entire lifecycle of a virtual server, we use it as the top folder name to identify the virtual server. The `Application` item is the identifier of a local program running in the virtual server that requires the data decryption key. The data decryption keys are stored in the `Key` folder with associated `Metadata` information of the encrypted data. Different versions of one-time data decryption keys are sorted in different `Version` folders in order to state the usage order of them in the application. We use `Hash` to check the integrity of the value of the key.

A key client is a daemon program running in the Dom0 of a virtualized platform. It reads the data decryption keys from the global key storage on behalf of the applications running in the virtual servers of the same physical machine. In order to use the key client, the applications that the data owner runs in a virtual server needs to call an API provided by the key client through shared library.

For security purposes, the global key storage is configured as read-only to key clients. Only the key server is able to write it. Besides, each key client is able to read the data exclusively assigned to the virtual server running on the same physical machine, e.g., a certain `UUID` folder.

B. Secure Key Distribution

To securely distribute a one-time data decryption key from the data owner to a virtual server, we follow the protocol shown in Figure 4. Specifically, there are two key distribution phases. Phase one includes Step 1 to Step 4 shown in Figure 4. It delivers the key from the data owner to the key server through public network; phase two includes Step 5 to Step 7 shown in Figure 4. It dispatches the key from the key server to a virtual server via the key client through in-cloud network. Details of each step are described as follows. For representation simplicity, we omit the timestamps and nonces for preventing re-play attacks in all protocol messages.

Step 1: The virtual server sends a `data request` packet to the data owner with three fields: `Metadata` about the requested data, $\langle IP_{keyserver}, Port_{keyserver} \rangle$ of the key server information, and its own identifier `UUID` and application identity `Application`.

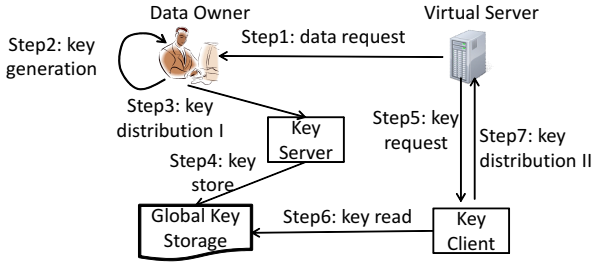

Fig. 4. Key distribution protocol with CloudSafe.

Step 2: Upon successful authorization verification based on the `UUID`, the data owner generates a pair of re-encryption and data decryption keys based on the `Metadata` she receives, according to the algorithm in Section IV-B.

Step 3: The data owner builds up a secure channel with the key server, and sends the data decryption key along with the virtual server identity, application identifier, `Metadata`, and key version to the key server through the channel.

Step 4: When the key server receives the key from the data owner, it writes the key into a directory named `UUID` of the global key storage, and sets the read permission only to the key client with `UUID`.

Step 5: The application in virtual server sends `key request` to the key client running in the same physical machine when it wants to use the data decryption key. The `key request` contains the virtual server identifier `UUID`, application identifier `Application`, and latest version information `Version`.

Step 6: The key client first verifies that the `UUID` is valid. If yes, it finds the folder according to `UUID`, `Application`, and `Version` in the global key storage, and reads the data decryption key. Note that the key client only reads the folder with `Version` that is smallest in which are greater than the received `Version`.

Step 7: The key client sends back the data decryption key to the virtual server with the same `UUID` along with `Application` and `Version` information. The `Version` information will be used by the virtual server for key retrieving next time.

With these steps, the data decryption key never appears in the space of the virtual server before it is used. Because the data decryption key is only used once, this distribution protocol ensures that the virtual server is the first one to use the key. Therefore, key compromise (e.g., through side channels) does not cause any harm to data confidentiality.

C. Virtual Server Lifecycle and Authentication

The lifecycle of a virtual server has three stages: creation, migration, and termination. To authenticate and authorize virtual servers for key usage, and sustain key availability to the virtual server through its entire lifecycle, CloudSafe executes the following steps.

Creation. Through certain interface provided by the cloud provider, the data owner creates a new virtual server instance in the cloud, and obtains a valid `UUID`. When the key server receives a response from the data owner after the new virtual server requests data from the data owner, it adds the new record to the global key storage with `UUID`. Since the `UUID` is generated by the hypervisor of the physical machine and cannot be forged by a malicious virtual server, it is used for authentication and authorization later when any application running in the virtual server requests data from the data owner, according to the key distribution protocol in Section V-B.

Migration: After the virtual server is migrated from one physical machine to another, the virtual server can continue to access the global key storage from the key client on the new physical machine, since the migration of the virtual server does not change its `UUID`.

Termination: When the data owner terminates a virtual server in the cloud, it notices the key server, which in turn cleans up the global key storage by deleting all data decryption keys associated with the `UUID`.

An alternate solution to support secure key usage is to send the one-time cryptographic secrets directly from the data owner to the virtual server in the cloud. However, the vulnerable virtual server environment makes it insecure to store secrets locally for virtual servers. CloudSafe utilizes an isolated secure storage and provides sustainable key access across the entire cloud to guarantee secure key storage and efficient key access in the public cloud.

VI. Implementation & Evaluation

A. Implementation

Cryptographic primitives. We instantiate the cryptographic primitives in Section IV-B in C language. We utilize OpenSSL library [4] for AES functionality in a CFB mode, and PBC library [5].

Key server. We implement the key server by utilizing network file system (NFS) as shown in Figure 5. With the convenient accessibility and scalability of NFS system, the key server is only responsible for communicating with the data owner and storing data decryption keys. While the NFS server is in charge of maintaining the global key storage and connecting with key clients in the cloud. In our prototype, we place both the key server and the NFS server on a dedicated physical machine, manually create key storage files for individual virtual servers in the cluster. We treat the key server as a TCP packet listener, which listens on a certain port and accepts data from the data owner and writes the data through the NFS server.

Key client. We utilize the NFS client to realize the functionality of the key client as shown in Figure 6. To obtain the key for the virtual server, the key client first reads the keys from the global key storage through the NFS client and then passes them to applications in the virtual server through the in-cloud network. The application running in a virtual server in DomU communicates to the key client via a shared library.

B. Evaluation

To confirm the practicability of CloudSafe, we demonstrate that CloudSafe brings acceptable cost to the system and the network by answering the following questions:

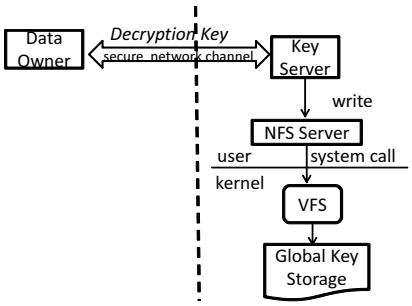

Fig. 5. The implementation of the key server with NFS.

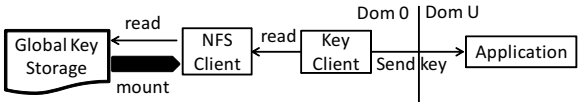

Fig. 6. The implementation of a key client with NFS client.

1) Do we have efficient proxy re-encryption operations on a dedicated server to support one-time key strategy?
2) Does the proposed key management framework cause a large network overhead?

We deployed the prototype of CloudSafe over a computing cluster with 324 individual computers, each of which is Apple Mac Pro with eight Intel Xeon CPU 2.80GHz and 8GB memory. We install Xen virtualization environment on six machines and connect them with Ethernet. A cluster-wide NFS file system is installed and a 10TB storage is mounted to the six machines in a way that the storage is only accessible in Dom0 of these machines.

1) Re-encryption Efficiency.: To validate the practicality of the strategy of combining one-time data decryption key and proxy re-encryption, we focus on the overhead caused by re-encryption operations which is executed by the proxy in CloudSafe. We test our algorithm on a single machine equipped with Intel 2 Duo CPU 2.93GHZ, 4GB memory, and Ubuntu 12.4 operating system. We run the code with different sizes of encrypted data. For each size, we run the experiment six times and record the average computation time.

Figure 7 shows the computational overhead of the re-encryption operations with different sizes of encrypted data. The cost of re-encryption operation is independent from the file sizes. The average processing time of re-encryption operation per file is around 0.005 seconds which means 200 data requests can be handled within one second. It might be true that the processing time in real applications would be larger than the experimental result. We expect that it will not significantly affect our solution's practicability in the public cloud environment.

2) Network Latency: We investigate the network latency caused by the proposed two-phase key distribution protocol in our cluster environment. Recall that CloudSafe utilizes a key server, a NFS server, NFS clients, and key clients to distribute data decryption keys from data owners to virtual servers. Hence, we calculate the network latency with the following equation, where the overall distribution latency of a single data decryption key consists of three parts: the latency

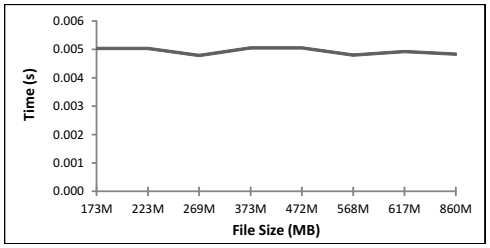

Fig. 7. Performance of proxy re-encryption operations on different sizes of encrypted files.

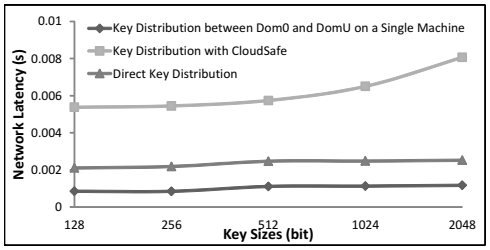

Fig. 8. Network latency of different key distribution scenarios.

between the data owner to the key server, the latency between the key server to a key client (NFS read), and the latency between the key client to a virtual server.

$$Latency_{overall} = Latency_{Dataowner \rightarrow Keyserver} + Latency_{NFSread} + Latency_{Keyclient \rightarrow Virtualserver}$$

For latency between the key server to the key client, it takes a fixed 0.003 seconds to distribute a data decryption key which is less than 2048 bits. For the other two kinds of latency, we place the data owner, the key server, the key client on different machines in the cluster and record the round-trip time of sending different sizes of keys from the data owner to the virtual server. The network setup for the virtual server is bridged which means the virtual server is at the same local area network as the other three components. For comparative purpose, we also record the round-trip latency of direct key distribution from the data owner to the virtual server through the network. We run each experiment six times and average the round-trip latency.

As shown in the Figure 8, it is obvious that CloudSafe brings extra network latency, which grows along with the increasing of the size of the keys. Note that the network latency caused by CloudSafe is exaggerated in our experiment. Because the data owner and virtual server are located in the same LAN in our experiment, the network latency of *Direct Key Distribution* is small. We expect these values would be much larger in wide area networks. Therefore, the latency gap introduced by CloudSafe in Figure 8 will be smaller in practice.

We note that the two phase key distribution of CloudSafe is not necessarily consecutive. When a virtual server needs to use the data decryption key, the latency for key distribution is caused by the NFS read and phase two key distribution as long as phase one key distribution is finished in advance. We evaluate phase two key distribution through the network latency from Dom0 and DomU on a single machine. As shown in the Figure 8, it takes less than 0.001 seconds to deliver a

2048 bits key. With fixed NFS read operation, the runtime overhead caused by CloudSafe for key distribution should be at most 0.004 seconds. Given this small overhead, we believe that CloudSafe brings acceptable network latency.

VII. Related Work

Side-channel attacks on stealing cryptographic keys are not new threats [9], [23]. Corresponding improvements on cryptographic algorithms have also been extensively discussed [9], [17], [18], [23]. Most recently, Ford seeks to solve this problem with deterministic execution and pacing queues to eliminate the sharing and concurrency in the system [14]. This solution is promising but still at its preliminary stage. CloudSafe aims to provides means to minimize the attack window of side-channel attacks with one-time keys while preserving the system efficiency with acceptable overhead.

With increasing threats towards cloud virtualization environments, researchers from both industry and academia have proposed several solutions to secure current cloud platform. Researchers in IBM corporation have proposed and realized virtual trusted platform module (vTPM) for Xen virtualization environment to explore the usage of trusted platform module (TPM) in cloud environment [3]. CloudSafe is compatible with vTPM in which CloudSafe is able to seal its one-time data decryption key with vTPM capability. However, without one-time key strategy, secure key storage cannot prevent stealing from malicious cloud customers through hidden side-channels. Song et al. proposed a general framework to build data-protection-as-service (DPaaS), which integrates various protection techniques to provide a combined multi-tier protection mechanisms for the current cloud [22]. Different from DPaaS which provides general security guidelines for the cloud, CloudSafe focuses on a specific security issue and is able to securely and efficiently protect outsourced data and cryptographic key usage against unauthorized data users and the cross-VM side channel attacks in the cloud.

Data-centric security research in the cloud focuses on cloud storage security and access control of outsourced data [11], [12], [13], [15], [16], [19], [25], [26], [27], [29], [30], [33]. For example, Kamara et al. [15] discussed a general solution for securing cloud storage by cryptographic techniques. Yu et al. [29], [30] proposed specific solutions with the help of attribute-based encryption and proxy re-encryption to secure and control the access to the outsourced data in cloud storage. Popa et al. [19] built CloudProof, a proof-based cloud storage which enables a customer to verify the integrity, write-serializability, and freshness of her data after she stores them in cloud. CloudSeal [28] leverages due-layer encryption algorithms to achieve flexible data access control and efficient content delivery in cloud environment. However it does not address secure data processing issue. Researchers pointed out that cryptography alone is not enough for privacy-preserving computing for cloud applications [24]. CloudSafe augments cryptographic solutions with a systematic approach to secure data processing in the cloud by minimizing the damage caused by side-channel attacks.

VIII. Conclusions and Future Work

We proposed CloudSafe, a framework that provides systematic protection for data accessing and processing in vulnerable public cloud environment. Cross-VM side-channel attack is a long-term and critical threat to cloud customers. CloudSafe reduces the attack surface of side-channel attacks by using one-time data decryption keys. CloudSafe leverages a dedicated in-cloud proxy and key distribution framework to achieve security and performance requirements. Our evaluation with an implemented prototype confirms that CloudSafe is a practical solution towards large-scale cloud applications. Our future work will include improving the scalability of CloudSafe's key management and building prototype applications that utilize CloudSafe.

IX. Acknowledgements

We thank anonymous referees for their valuable reviews. This work was supported in part by NSF grant CAREER CNS-0953638.

References

[1] Amazon EC2 Dedicated Instances. http://aws.amazon.com/dedicated-instances/.
[2] Gogrid Dedicated Servers, http://www.gogrid.com/products/infrastructure-dedicated-servers.
[3] IBM Virtual Trust Platform Module. http://researcher.watson.ibm.com/researcher/view_project.php?id=2850.
[4] OpenSSL Cryptography and SSL/TLS Tookit, http://www.openssl.org/.
[5] Pairing-based Cryptography Library, http://crypto.stanford.edu/pbc/.
[6] Xen. http://xen.org/.
[7] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro. A Security Analysis of Amazon's Elastic Compute Cloud Service. In *Proc. of ACM SAC*, 2012.
[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, Oct. 2003.
[9] D. J. Bernstein. Cache-timing Attacks on AES. Technical report, 2005.
[10] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider. AmazonIA: When Elasticity Snaps Back. In *Proc. of ACM CCS*, 2011.
[11] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control. In *Proceedings of CCSW*, 2009.
[12] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Efficient and Private Access to Outsourced Data. In *Proc. of ICDCS*, 2011.
[13] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of Access Control Evolution on Outsourced Data. In *Proc. of VLDB*, 2007.
[14] B. Ford. Plugging Side-channel Leaks with Timing Information Flow Control. In *Proc. of USENIX HotCloud*, 2012.
[15] S. Kamara and K. Lauter. Cryptographic Cloud Storage. In *Financial Cryptography and Data Security*, volume 6054 of *Lecture Notes in Computer Science*, pages 136–149. Springer Berlin / Heidelberg, 2010.
[16] M. Li, S. Yu, N. Cao, and W. Lou. Authorized Private Keyword Search over Encrypted Personal Health Records in Cloud Computing. In *Proc. of ICDCS*, 2011.
[17] D. Page. Theoretical Use of Cache Memory As A Cryptanalytic Side-Channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002.
[18] D. Page. Defending against Cache-based Side-channel Attacks. *Information Security Technical Report*, 8(1):30 – 44, 2003.
[19] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. Enabling Security in Cloud Storage SLAs with CloudProof. In *Proc. USENIX ATC*, 2011.

[20] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *Proc. of ACM CCS*, 2009.
[21] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono. All Your Clouds Are Belong to Us: Security Analysis of Cloud Management Interfaces. In *Proc. of ACM CCSW*, 2011.
[22] D. Song, E. Shi, I. Fischer, and U. Shankar. Cloud Data Protection for the Masses. *Computer*, 45(1):39–45, Jan. 2012.
[23] E. Tromer, D. A. Osvik, and A. Shamir. Efficient Cache Attacks on AES, and Countermeasures. *J. Cryptol.*, 23(2):37–71, Jan. 2010.
[24] M. Van Dijk and A. Juels. On The Impossibility of Cryptography Alone for Privacy-preserving Cloud Computing. In *Proc. of USENIX HotSec*, 2010.
[25] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou. Toward Secure and Dependable Storage Services in Cloud Computing. *IEEE Trans. Serv. Comput.*, 5(2):220–232, Jan. 2012.
[26] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In *IEEE INFOCOM*, 2010.
[27] W. Wang, Z. Li, R. Owens, and B. Bhargava. Secure and Efficient Access to Outsourced Data. In *Proceedings of CCSW '09*, 2009.
[28] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen. Towards End-to-End Secure Content Storage and Delivery with Public Cloud. In *Proc. of ACM CODASPY*, pages 257–266, 2012.
[29] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *IEEE INFOCOM*, 2010.
[30] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute Based Data Sharing with Attribute Revocation. In *Proc. of ACM ASIACCS*, 2010.
[31] Y. Zhang, A. Juels, A. Oprea, and M. Reiter. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *IEEE Symposium on Security and Privacy*, pages 313 –328, may 2011.
[32] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-VM Side Channels and Their Use to Extract Private Keys. In *Proc. of ACM CCS*, 2012.
[33] Q. Zheng and S. Xu. Secure and Efficient Proof of Storage with Deduplication. In *Proc. of CODASPY*, pages 1–12, 2012.

APPENDIX

Theorem 1. Given that SE is a CPA secure symmetric encryption scheme and Decisional Bilinear Diffie-Hellman assumption is hard, the scheme achieves chosen-plaintext security for ciphertexts generated by PEnc.

Proof In the threat model, the adversary $\mathcal{A}$ might have one key of the key pair $((x,t), g^{t/y})$ rather than both. In what follows, we prove that our scheme achieves CPA security for ciphertexts generated by PEnc even $\mathcal{A}$ has either the re-encryption key $g^{t/y}$ or data decryption key (x,t). Note here we model the keyed hash function F as a random oracle.

First let $\mathcal{A}$ has the re-encryption key $g^{t/y}$. We prove that if $\mathcal{A}$ can break the CPA security for ciphertexts generated by PEnc in our scheme, then we can simulate an challenger breaking the CPA security of SE.

The challenger first proceeds the security game as the standard CPA game.

In the challenge phase, $\mathcal{A}$ presents M^0 and M^1 such that $|\mathsf{M}^0| = |\mathsf{M}^1|$. The challenge encrypts M^λ where λ is randomly selected from $\{0,1\}$: Let $s = 1$, and compute $x = F_{\mathsf{mk}}(H(\mathsf{fid})|0)$ and $y = F_{\mathsf{mk}}(H(\mathsf{fid})|1)$ where M^λ is uniquely identified by the file handler fid. Then it encrypts M^λ with SE, s.t. $\mathsf{M}' = (\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_i, \cdots, \mathsf{M}_n)$ where $\mathsf{M}'_i = \mathsf{SE.Enc}(x, \mathsf{M}_i)$ and x is the symmetric key for SE, then generates the ciphertext as $\mathsf{cph} = (g^{ys}, \{\mathsf{M}'_i e(g,g)^s\}_{i=1}^n)$. The challenger returns cph to $\mathcal{A}$.

The challenger continues to proceed the security game as the standard CPA game and it is allowed to obtain $g^{t/y}$.

If $\mathcal{A}$ can output λ' such that $\lambda = \lambda'$ with non-negligible advantage, then it means that the challenger can break the CPA security of SE with no-negligible advantage.

Similarly, we can prove that our scheme achieves CPA security for ciphertexts generated by PEnc in the case that $\mathcal{A}$ has the data decryption key (x,t) with the Decisional Bilinear Diffie-Hellman assumption. Given the instance $(e, G, G_T, g, g^a, g^b, g^c, Q)$, We prove that if $\mathcal{A}$ can break the CPA security for ciphertexts generated by PEnc in our scheme, then we can simulate an challenger breaking Decisional Bilinear Diffie-Hellman assumption.

The challenger first proceeds the security game as the standard CPA game.

In the challenge phase, $\mathcal{A}$ presents M^0 and M^1 such that $|\mathsf{M}^0| = |\mathsf{M}^1|$. The challenge encrypts M^λ where λ is randomly selected from $\{0,1\}$: Compute $x = F_{\mathsf{mk}}(H(\mathsf{fid})|0)$ where M^λ is uniquely identified by the file handler fid. Then it encrypts M^λ with SE, s.t. $\mathsf{M}' = (\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_i, \cdots, \mathsf{M}_n)$ where $\mathsf{M}'_i = \mathsf{SE.Enc}(x, \mathsf{M}_i)$ and x is the symmetric key for SE, then generates the ciphertext as $\mathsf{cph} = (g^z, \{\mathsf{M}'_i Q\}_{i=1}^n)$, by selecting z random from $\mathbb{Z}_p$ and implicitly letting $y = z/abc$. The challenger returns cph to $\mathcal{A}$.

The challenger continues to proceed the security game as the standard CPA game and it is allowed to obtain (x,t).

If $\mathcal{A}$ can output λ' such that $\lambda = \lambda'$ with non-negligible advantage, then it means that the challenger can break the Decisional Bilinear Diffie-Hellman assumption with no-negligible advantage. □

Theorem 2. Given the one-way property of keyed hash function F, the master key is secure.

Proof As in the cryptographic scheme, the adversary $\mathcal{A}$ can only obtain $x, t, g^{t/y}$, which is related to the master key mk. Given the one-way property of keyed hash function $F : \{0,1\}^* \xrightarrow{k} \mathbb{Z}_p$, $\mathcal{A}$ cannot retrieve the master key mk with non-negligible probability. □

Theorem 3. Assume that the cloud and data users do not collude. Given one key from the key pair $((x,t), g^{t/y})$, the probability of inferring the other key is negligible under the hardness assumption of computing the discrete logarithm.

Proof Given the current data decryption key (x,t), let us consider the probability of inferring $g^{t/y}$. Note that y only appears in g^{ys}, which is part of ciphertexts and s is an unknown random number. Therefore, the probability of obtaining $g^{t/y}$ is negligible for the computation under the hardness assumption of computing the discrete logarithm.

Now given the re-encryption key $g^{t/y}$, the probability of obtaining t from $g^{t/y}$ is negligible for the computation under the hardness assumption of computing the discrete logarithm. In addition, as t is used only once, the probability of getting (x,t) from $g^{t/y}$ is negligible even the adversary has obtained the previous decryption key $(x,t_1), \ldots, (x,t_k)$.□