WebCloud: Web-Based Cloud Storage for Secure Data Sharing across Platforms

Shuzhou Sun, Hui Ma, Zishuai Song, Rui Zhang

Abstract—With more and more data moving to the cloud, privacy of user data have raised great concerns. Client-side encryption/decryption seems to be an attractive solution to protect data security, however, the existing solutions encountered three major challenges: low security due to encryption with low-entropy PIN, inconvenient data sharing with traditional encryption algorithms, and poor usability with dedicated software/plugins that require certain types of terminals.

This work designs and implements WebCloud, a practical browser-side encryption solution, leveraging modern Web technologies. It solves all the above three problems while achieves several additional remarkable features: robust and immediate user revocation, fast data processing with offline encryption and outsourced decryption. Notably, our solution works on any device equipped with a Web user agent, including Web browsers, mobile and PC applications. We implement WebCloud based on ownCloud for basic file management utility, and utilize WebAssembly and Web Cryptography API for complex cryptographic operations integration. Finally, comprehensive experiments are conducted with many well-known browsers, Android and PC applications, which indicates that WebCloud is cross-platform and efficient.

As an interesting by-product, the design of WebCloud naturally embodies a dedicated and practical ciphertext-policy attribute-based key encapsulation mechanism (CP-AB-KEM) scheme, which can be useful in other applications.

Index Terms—Web-Based Cloud Storage, Secure Data Sharing, Cross-Platform Encryption/Decryption Solution, Attribute-Based Encryption

1 Introduction

PUBLIC cloud storage service becomes increasingly popular due to cost reduction and good data usability for users. This trend has prompted users and corporations to store (unencrypted) data on public cloud, and share their cloud data with others. Using a cloud for high-value data requires the user to trust the server to protect the data from unauthorized disclosures. This trust is often misplaced, because there are many ways in which confidential data leakage may happen, e.g. these data breaches reported [1], [2], [3], [4], [5], [6]. To counteract data leakage, one of the most promising approaches is client-side encryption/decryption.

Concretely, client-side encryption allows senders to encrypt data before transmitting it to clouds, and decrypt the data after downloading from clouds. In this way, clouds only obtain encrypted data, thus making server-side data exposure more difficult or impossible. At the same time, as a crucial functionality of cloud storage, flexible file sharing with multiple users or a group of users must be fully supported. However, existing client-side encryption solutions suffer from more or less disadvantages in terms of security, efficiency and usability.

Known Client-Side Encryption Solutions. We review existing solutions and point out their limitations.

- **Limited support or no support.** Many cloud storage providers, including Google Drive and Dropbox, do not provide support for client-side encryption. They adopt server-side encryption for files stored, TLS for data at transit, and two-factor authentication for user authentication. Apple iCloud supports end-to-end encryption for sensitive information, e.g., iCloud Keychain, Wi-Fi passwords. For other data uploaded to iCloud, only server encryption is adopted.
- **Password-Based Solutions.** Some products [7], [8], [9] use symmetric encryption (typically AES) to encrypt users' data and then upload ciphertexts to clouds. However, in these schemes, the cryptographic keys are derived from a password/passphrase or even a 4-digit PIN. Relying on such low entropy is considered unsafe [10]. Worse still, most password-based solutions only deal with the case of single-user file encryption and decryption, and do not provide any file sharing mechanism. Notably, [7] allows users to generate a share link for each password-protected file. However, users must manually send the share link through one channel, and password to all receivers through another *secure* channel, which is inconvenient and brittle.
- **Hybrid Encryption Scheme.** The cloud adopts a key encapsulation mechanism (KEM) and a data encapsulation mechanism (DEM), so called the KEM-DEM setting. Many public cloud service providers, including Amazon [11], Tresorit [12], and Mega [13], adopt the RSA-AES paradigm. Users generate RSA key pairs and apply for certificates from the providers, who build and maintain a Public Key Infrastructures (PKI). Users encrypt data under fresh sampled AES keys, which are further encrypted under all recipients' RSA public keys. This file sharing mechanism

- *The authors are with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, and the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China. Email: {sunshuzhou,mahui,songzishuai,r-zhang}@iie.ac.cn. (Shuzhou Sun and Hui Ma contributed equally to this paper and are labeled as co-frist authors. Corresponding Author: Rui Zhang.)*

is inflexible and inefficient. A sender needs to obtain and specify the public keys of all receivers during encryption. Even worse, the size of the ciphertext and encryption workload are proportional to the number of recipients, resulting in greater bandwidth and storage costs and more user expenditure.

Limitations of the Existing Solutions. Three drawbacks exist in above-mentioned solutions: 1) comparatively poor security, 2) coarse-grained access control, inflexible and inefficient file sharing, and 3) poor usability. The first two are easy to see and we now elaborate the usability issue. Typically, users use different terminals to upload files, including desktop, Web and mobile applications [14]. However, almost all the existing solutions require additional software or plugins, thus limiting users' devices and platforms. When switching to a new device, users need to repeat the boring installation process, which greatly increases users' burden thus decreases usability.

1.1 Design and Challenges

In this work, we focus on designing and implementing a practical, secure and cross-platform public cloud storage system. The proposed solution, WebCloud, is a Web-based client-side encryption solution. Users encrypt and decrypt their data using Web agents, e.g., Web browsers. For the encryption algorithm, a secure and efficient Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme is proposed, which achieves encryption and access control simultaneously. The cloud service serves as a storage backend and does not handle users' secret keys. Specifically, WebCloud admits a uniform design and addresses all the three mentioned limitations: 1) The security of WebCloud relies on the security of the proposed CP-ABE and cryptographic keys; 2) The file sharing functionality is flexible and efficient, allowing one ciphertext to be decrypted by many receivers; and 3) WebCloud is cross-platform and works on any popular browser, without any plugin installed.

We discuss our design rationality and point out the related challenges. Recent advances of Web technology, namely Web browsers, have greatly enhanced security and usability across different platforms. On the one hand, new features such as Web Cryptography API [15] and WebAssembly [16] are introduced. On the other hand, today's personal computers and mobile devices have Web browser installed by default. Therefore, it is promising to use browser as the encryption engine. However, efficient combination of Web techniques and cryptographic schemes is completely nontrivial since each technique has its own suitable scenarios.

Certain cryptographic schemes have appealing properties to be adopted in cloud computing scenario. In particular, CP-ABE [17] has many applications in cloud data protection and sharing, since it provides fine-grained (noninteractive) access control and encryption simultaneously. In CP-ABE, an access policy is embedded into a ciphertext, thus allowing many recipients to decrypt the ciphertext. It should be promising to use CP-ABE as the encryption scheme in a client-side encryption solution. A lot of work has been done on CP-ABE, e.g., proposing new constructions [18], optimizing computation efficiency [19], [20], [21] and supporting revocation [22]. However, existing CP-ABE schemes do not combine state-of-the-art techniques, and cannot achieve efficient data encryption, robust and immediate user revocation, offline encryption and outsourced decryption simultaneously.

1.2 Our Results

We view our contribution as the uniform design, rigorous analysis and efficient implementation of WebCloud, in particular, it simultaneously achieves the following:

- **Practical Encryption Solution for Cloud Storage.** We introduce WebCloud, a practical client-side encryption solution for public cloud storage, which effectively combines modern Web techniques and cryptographic algorithms. WebCloud involves of a key management mechanism, a dedicated attribute-based encryption scheme and a high-speed implementation. More importantly, WebCloud is cross-platform (including major browsers, Android and PC) and plugin-free.
- **Fine-Grained Access Control Mechanism with ABE.** It is widely-accepted that attribute-based encryption (ABE) is promising for fine-grained access control of data. However, we find that the existing ABE schemes suffer from high computational overhead, or some vital missing functionalities, e.g., inefficient data encryption, robust and immediate user revocation, offline encryption and outsourced decryption simultaneously. To solve this problem, we propose a dedicated ciphertext-policy attribute-based access control mechanism. The proposed scheme can also be used in other scenarios.
- **Rigorous Security Analysis.** We present a security model of WebCloud, including the adversarial models for the Web and the cryptographic scheme simultaneously. The security analysis is then done in the proposed model, namely, the provable security of the proposed CP-ABE scheme and the reliability of the key storage in the browser side.
- **Efficient Operation inside Browsers.** We implement WebCloud based on ownCloud [23]. The functionalities and performances are evaluated in major browsers on many devices, and applications on PC and Android devices. The benchmark result indicates that WebCloud is a practical solution. Most remarkably, in the Chrome browser on a 4-core 2.2 GHz Macbook machine, encrypting a 1 GB file takes 3.1 seconds, while decryption costs 3.9 seconds.

We describe a scenario of WebCloud. When used inside a corporation, the company itself generates and distributes secret keys, and any public cloud service can be adopted as the cloud server. When an employee joins into the company, he/she registers with a set of attributes, e.g., ("IT Department", "Man"), and obtains a secret key. The employee shares files with others by encrypting the files under a specific access policy, e.g., "IT Department and Manager", using CP-ABE on a Web browser. The ciphertexts are then uploaded to the cloud. Employees who satisfy the access policy can download the ciphertext and decrypt on their

browsers. The company has no need to build its own cloud infrastructure and each employee can use personal computers or mobile devices to access the cloud. An employee is revoked from the system when he/she resigns from the company.

Organization. The rest of the paper is organized as follows. Section 2 recaps modern Web technologies and primitives. Section 3 elaborates the design of WebCloud. Section 4 give detailed descriptions of the algorithms in WebCloud. Section 5 gives rigorous security analysis and highlights nice features of WebCloud. Section 6 demonstrates benchmarks of WebCloud on many devices. Section 7 gives the conclusion.

1.3 Related Work

In-Browser Cryptography. Both the Web community and security researchers understand the importance and usefulness of in-browser cryptography and have made remarkable efforts in this area.

JavaScript cryptographic libraries were developed for ease of use of cryptography on browsers, for instance [24], [25], [26]. Many of these libraries have a large number of downloads, e.g., 423,368 for OpenPGP.js [24] in total. The World Wide Web Consortium (W3C) noticed this trend of using in-browser cryptography and as a solution proposed a standard called Web Cryptography API [27], [15]. The standard supports a few widely adopted standard algorithms, e.g., AES and ECDSA, which is convenient for building several secure Web applications [28] including authenticated video services and encrypted communications via Web mail.

Meanwhile, there are researches in the literature having explored the idea of running cryptographic algorithms on Web browsers. [29] focused on using Identity-Based Cryptography for client side security in Web applications and presented a JavaScript implementation of their scheme. They selected Combined Public Key cryptosystem as the encryption scheme to avoid complex computations involved in bilinear pairing and elliptic curve. ShadowCrypt [30] allows users to transparently switch to encrypted input/output for text-based Web applications. It requires a browser extension, replacing input elements in a page with secure, isolated shadow inputs and encrypted text with secure, isolated cleartext. [26] implemented several Lattice-based encryption schemes and showed the speed performance on four common Web browsers on PC. Their results demonstrated that some of today's Lattice-based cryptosystems can already have efficient JavaScript implementations. Recently, [31] constructed an efficient two-level homomorphic public-key encryption in prime-order bilinear groups and presented a high-performance implementation using WebAssembly that allows their scheme to be run very fast on any popular Web browser, without any plugins required.

Attribute-Based Encryption. Attribute based encryption (ABE) was first introduced by Sahai and Waters under the name fuzzy identity-based encryption [32]. Goyal et al. [33] extended fuzzy IBE to ABE. Up to now, there are two forms of ABE: key-policy ABE (KP- ABE) [33], [34], [35], [36], where the key is assigned to an access policy and the ciphertext to a set of attributes, and ciphertext-policy ABE (CP-ABE) [17], [37], [38], where the ciphertext is assigned to an access policy and the key to a set of attributes. A user can decrypt a ciphertext if the set of attributes satisfies the access policy. In this work, CP-ABE is adopted as a building block of WebCloud: each file has an access policy to indicate the allowed receivers.

The complex pairing and exponentiation operations in ABE are migrated by many works. Green et al. [19] introduced outsourced decryption into ABE systems such that the complex operations of decryption can be outsourced to a cloud server, only leaving one exponentiation operation for a user to recover the plaintext. Further, online/offline ABE [20] was proposed by Hohenberger and Waters, which splits the original algorithm into two phases: an offline phase which does the majority of encryption computations before knowing the attributes/access control policy and generates an intermediate ciphertext, and an online phase which rapidly assembles an ABE ciphertext with the intermediate ciphertext after the attributes/access control policy is fixed. Meanwhile, [20] proposed two scenarios about the offline phase: 1) the user does the offline work on his smartphone. 2) A high-end trusted server helps the user with low-end device do the offline work.

2 Preliminary

2.1 Modern Web Technologies

Web Cryptography API. The Web Cryptography API [15] is a standard API for accessing cryptographic primitives in JavaScript-based environments (e.g., browsers and Node.js). The API is only available within HTTPS. Actually, major Web browsers and operating systems already contain well-verified and reviewed cryptographic algorithms. The API simply exposes these algorithms to Web application developers through a standardized interface. It contains a cryptographically strong pseudo-random number generator (CSPRNG) and supports a few widely used standards, e.g., AES and PBKDF2. Unfortunately, these primitives are still insufficient for building some novel cryptographic schemes, e.g., pairing-based cryptography including Attribute-Based Encryption. Even worse, this API does not specify storage mechanisms for cryptographic keys.

WebAssembly. WebAssembly [16] is a binary instruction format for stack-based virtual machine. It is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the Web for client and server applications. It can be invoked from JavaScript codes. WebAssembly aims at executing applications at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including mobile and IoT. The WebAssembly codes are shipped to users' browsers along with other Web files (e.g., stylesheets), compiled to binary codes on users' browsers and optionally cached in IndexedDB.

Web Storage. Browsers usually use Web Storage API or IndexedDB to store data. The Web Storage API provides mechanisms to store key/value pairs, in a much more intuitive fashion than using cookies. The API includes temporary storage sessionStorage (available for the duration of the page session) and persistent storage localStorage (can be persisted even when the browser is closed and reopened). It contains megabyte storage, usually 5 MB for

mobile browsers and 10 MB for desktop browsers. For storing larger amounts of structured data, IndexedDB [39] provides a solution, which is a low-level API for client-side storage of significant amounts of structured data.

Web Workers. Typically, browsers create a single main thread for a Web page. If there are costly operations in the main thread that take significant time, the user interface may be frozen, e.g., clicking and typing have no response. A Web Worker is a JavaScript script that runs in background threads, which is more efficient to utilize multi-core CPUs. The worker can perform time-consuming computations without interfering the user interface. The Web Worker API provides methods for initializing and destroying workers. Once created, a worker can send messages to the main thread or store computed results in IndexedDB (other types of Web storages are not available in Web workers).

2.2 Multi-Factor Authenticated Key Exchange

Multi-Factor Authenticated key exchange (MFAKE) is a useful tool where a user interacts with a server to set up a session key where pre-registered information (aka. authentication factor), such as a password or biometrics, of the user is stored. So far, many practical MFAKE schemes have been proposed, e.g., [40], [41], [42], [43]. We review [43] briefly, which is adopted as one of building component in WebCloud.

In [43], an efficient MFAKE scheme was constructed under the Decisional Diffie-Hellman (DDH) assumption, where a user is authenticated with different factors simultaneously, such as passwords, biometrics (e.g., fingerprint) and hardware devices. It consists two phrases: registration and login-Authentication.

Registration: The registration phase takes place in a secure and reliable environment. The client generates a password α, a good biometrics template W, and a random group element $\gamma \in \mathbb{Z}_{F_p}^*$. The biometrics template W is sent to the registration center (RC) and derived to a random β using a fuzzy extractor algorithm. RC computes $Z = H^{\alpha+\beta+\gamma}$, uses a public key encryption scheme to encrypt Z and other user information and stores the encrypted result into database. The client should store γ safely on his/her devices.

Login-Authentication: A client with userid uses a registered device and sends an authentication request to server. The server computes an authentication challenge and sends it to client. The user inputs password α and biometrics W'. Meanwhile, the device reads γ and computes β from W' using fuzzy extractor algorithm. Then client computes authentication response and sends it to server. Both sides compute authentication confirmation, and determine whether this mutual authentication and key agreement is completed.

WebCloud can use MFAKE [43] for secure key distribution. The registration center is the private key generator and the registration phase happens in the key generation phase of WebCloud. The device secret γ is stored on users' computers and can be accessed by browsers with users' interactions. While [43] adopted fingerprint as the biometric data and required special fingerprint scanner, WebCloud uses facial features [44] that can be extracted from cameras.

3 Overview of WebCloud

3.1 System Model

As shown in Fig. 1, WebCloud adopts the browser and server (B/S) architecture. There are four entities involved: a private key generator (PKG), a public cloud server, data owners and data consumers. The roles of each entity are as follows:

- PKG generates and distributes system parameters and keys to other entities, and instructs the cloud to revoke a user. PKG maintains a Public Key Infrastructure (PKI) and plays as the root Certificate Authority (CA). We stress that this only increases PKG's workload marginally since certificate issuance and key distribution are completed at the same time.
- The public cloud server provides a website for accessing and storing data in Web user agents. Moreover, it runs a few services:
  - Storage Service (SS) stores transformation keys and encrypted data reliably.
  - Outsourced Decryption Service (DS) checks whether a data consumer has been revoked and preprocesses ciphertext to ease computation overhead of decryption for users.
  - Key Update Service (KUS) updates cloud secret key CSK periodically or when current CSK is leaked.
  - Ciphertext Update Service (CUS) updates ciphertexts with new CSK.
- Data owners decide access policies and encrypt data under these policies before uploading to the public cloud.
- Data consumers download encrypted data from public cloud server and decrypt the data locally.

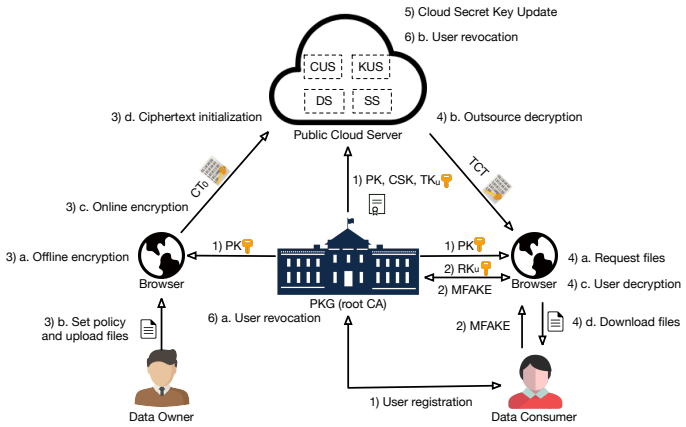
We remark that such a trusted party (serving as PKG) is not hard to find, for instance, e.g., government organizations or major banks.

3.2 Security Notions

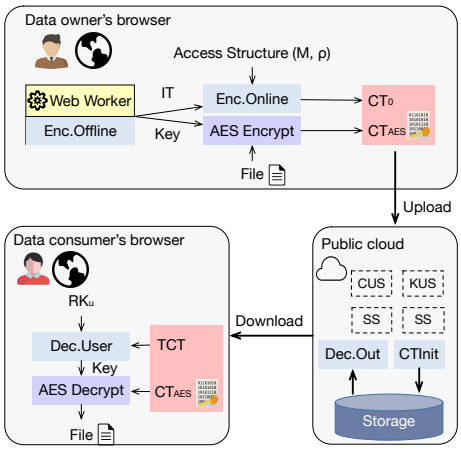
The security goal of WebCloud is to protect users' data from disclosure on the server side for cloud storage systems. We assume the cloud is *honest-but-curious* [45]. The cloud honestly follows the protocol, e.g., provides storage service and outsource decryption service. It does not adversarily modify users' data. Most of the data consumers are honest, but few of them may be corrupt and share their secret keys in the collusion. On the contrast, PKG and data owners are assumed to be totally trusted. All the communications are secured by TLS [46]. Following adversary models are considered:

Passive Man-in-the-Middle. The adversary reads all network traffic passively, but does not perform any active attacks, e.g., altering network packets.

Web Attacker Model. This model is the standard security model of Web applications [47]. An adversary in this model can access any open Web application, learn its client-side code, send emails and other messages, and can set up their own (malicious) Web applications. The adversary is unable



(a) System Model of WebCloud



(b) Outline of WebCloud

Fig. 1: System Design of WebCloud

to forge Web origins [48], since this undermines the security of any Web application.

Data Security against User Collusion. In this model, some users and the cloud can collude in arbitrary manner, e.g., they can obtain some cryptographic keys $\mathsf{RK}_u, \mathsf{TK}_u$ and cloud secret key $\mathsf{CSK}_{\mathsf{ctr}}$. They try to decrypt files that beyond their authorized access rights. We formalize it with Definition 1 in Appendix B.

Data Security against Cloud Server. The public cloud, who can obtain $\mathsf{CSK}_{\mathsf{ctr}}$, the conversion key TK_u of all the users, and all the ciphertexts, cannot decrypt the ciphertexts. We formalize it with Definition 2 in Appendix B.

User Revocation Validity. In this model, a revoked user, who can obtain his/her secret keys $(\mathsf{SK}_u, \mathsf{TK}_u, \mathsf{RK}_u)$, cannot decrypt files within its authorized access rights. We formalize it with Definition 3 in Appendix B.

3.3 Deployment Architecture

As shown in Fig. 2, WebCloud consists of four functional modules (**M1** to **M4**).

M1. WebCloud Core. This module works in a user's browser and contains crypto and storage modules. The crypto module further implements following submodules.

- CP-AB-KEM1 submodule includes offline and online encryption, i.e., algorithms *Enc.Offline* and *Enc.Online*. Meanwhile, the submodule implements LSSS and converter from access policy string to access structure. The converter can convert an access policy string, e.g., "(Employee and IT department) or Manager" to an access structure (M, ρ). To the best of our knowledge, it is the first time that LSSS and the converter are implemented in JavaScript environment. This submodule takes advantage of WebAssembly by adopting MCL-WASM [49].
- Utilization submodule packs some useful routines, e.g., encrypting users' retrieval key RK_u, deriving AES key using PBKDF2, AES encryption and decryption etc. These routines invoke the Web Cryptography API.
- MFAKE submodule helps to establish a secure channel for retrieval key distribution. We implement the MFAKE protocol proposed in [43].

The storage module implements a cache storage layer in IndexedDB, which allows to store and obtain users' encrypted retrieval keys RK_u. Meanwhile, it also maintains a cache in sessionStorage, which contains intermediate ciphertexts (generated by offline encryption).

M2. WebCloud Storage. This module also works in users' browsers and provides access and query routines of Web storage, which is used by the storage management submodule of **M1**. It provides storage in browsers for the WebCloud system, including data consumers' retrieval keys in IndexedDB and intermediate ciphertexts in sessionStorage.

M3. Cloud Crypto Module. This module implements cryptographic routines at the server side, which includes following submodules.

- CP-AB-KEM2 submodule implements the CUS, KUS, DS services in the cloud. It includes ciphertext initialization and outsourced decryption routines, i.e., algorithms *CTInit*, *CSKUpdate*, *CTUpdate*, and *Dec.Out*. This submodule relies on MCL-C [49].
- Utility submodule contains useful functionalities, including file processing, logging functionality.
- Serialization submodule converts bytes to ABE ciphertext CT_0 and converts transformed ciphertext TCT to bytes.
- Revocation submodule maintains the revocation list $\mathbb{L}$. It provides add, delete and query functions of the list $\mathbb{L}$ to other submodules.

M4. Key Management. This module works in PKG and contains two modules. The authentication module authenticates users' identities via the multi-factor authenticated key exchange (MFAKE) protocol [43]. The key storage module is responsible to store users' retrieval key securely and return a user's key on receiving an authenticated user's request.

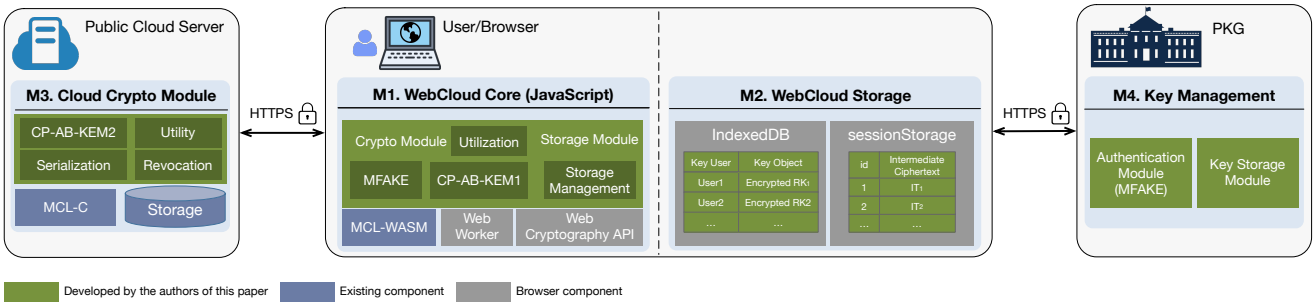


Fig. 2: System Architecture of WebCloud

4 Algorithms in WebCloud

4.1 Description of Algorithms

We now elaborate the algorithms of WebCloud (cf. Fig. 1). Some acronyms are listed in Table 1.

TABLE 1: Acronyms Used in This Paper

Acronym	Description	Acronym	Description
PKG	private key generator	CUS	ciphertext update service
KUS	key update service	DS	outsourced decryption service
SS	storage service	MSK	master secret key
PK	public key	ctr	current time counter
SK_u	a user's secret key	RK_u	a user's retrieval key
CSK	cloud secret key	TCT	transformed ciphertext
TK_u	a user's transformation key	IT	intermediate ciphertext

1) System Initialization. PKG runs the algorithm *Setup*() to generate a public key PK, a master secret key MSK and a cloud secret key CSK_1. All the data consumers register themselves to PKG: 1) run the registration phase of MFAKE protocol [43] where PKG serves as the registration center; and 2) state a set of properties to indicate their identities. Then, PKG runs the algorithms *KG*() and *KG.Random*() to generate TK_u and RK_u for each data consumer. Further, PKG generates a certificate $\mathcal{T}_{\mathsf{cloud}}$ for the public cloud server, which is used to establish secure communication between the cloud server and users. Finally, PKG distributes PK to all the entities, CSK_1 and TK_u along with $\mathcal{T}_{\mathsf{cloud}}$ to the cloud, and keeps RK_u for future distributions.

The above-mentioned algorithms are as follows:

Setup(λ, U). On input a security parameter λ and an attribute universe U, PKG chooses a bilinear map $D = (\mathbb{G}, \mathbb{G}_T, e, p)$, where $p \in \Theta(2^\lambda)$ is the prime order of groups $\mathbb{G}$ and $\mathbb{G}_T$. The attribute universe U consists of elements in $\mathbb{Z}_p^*$. It chooses random generators $g, h, u, v, w \in \mathbb{G}$, picks two random elements $\alpha, \beta_1 \in \mathbb{Z}_p^*$. It sets a counter $\mathsf{ctr} = 1$. Finally, PKG outputs: $\mathsf{PK} = (D, g, h, u, v, w, e(g,g)^\alpha)$, $\mathsf{MSK} = (\mathsf{PK}, \alpha)$ and $\mathsf{CSK}_1 = (\mathsf{csk}_1 = \beta_1, \mathsf{ctr})$.

KG(S, MSK). On input a master secret key MSK and an attribute set $S = (A_1, A_2, \ldots, A_k) \subseteq \mathbb{Z}_p^*$, PKG picks a random element $r \in \mathbb{Z}_p^*$ and computes $K_0 = g^\alpha w^r, K_1 = g^r$. For j from 1 to k, it picks random $r_j \in \mathbb{Z}_p^*$ and computes $K_{j,2} = g^{r_j}, K_{j,3} = (u^{A_j} h)^{r_j} \cdot v^{-r}$. PKG outputs a secret key $\mathsf{SK}_u = (S, K_0, K_1, \{K_{j,2}, K_{j,3}\}_{j\in[1,k]})$.

KG.Random$(\mathsf{PK}, \mathsf{SK}_u)$. On input a public key PK and a secret key SK_u, PKG picks a random element $\tau \in \mathbb{Z}_p^*$, then it computes $K_0' = (K_0)^{1/\tau}, K_1' = (K_1)^{1/\tau}$. For $j = 1$ to k, it computes: $K_{j,2}' = (K_{j,2})^{1/\tau}, K_{j,3}' = (K_{j,3})^{1/\tau}$. PKG outputs: $\mathsf{RK}_u = \tau, \mathsf{TK}_u = (S, K_0', K_1', \{K_{j,2}', K_{j,3}'\}_{j\in[1,k]})$.

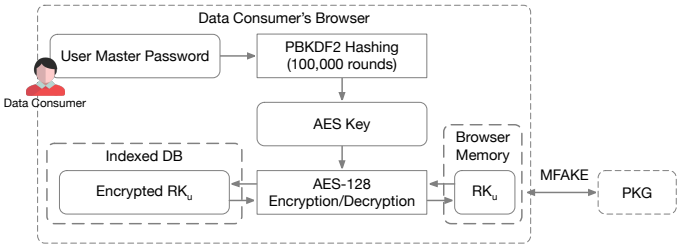


Fig. 3: Key Management of WebCloud

2) Key Management. To decrypt data in browsers, data consumers obtain their retrieval keys from PKG as shown in Fig. 3. To this end, the login-authentication phase of the MFAKE protocol [43], is run between a user's browser and PKG to establish a secure communication channel. The consumer's retrieval key RK_u is transmitted to the browser through the secure channel and later be used in the browser's memory. If the user remains idle for a specified period of time (e.g., 30 minutes), RK_u is automatically erased from the memory and later use of RK_u requires running the MFAKE protocol again.

The login-authentication phase of [43] requires a user to enter a few authentication factors, which may cause poor usability. For ease of use, another option is provided. We derive a 128-bit AES key with PBKDF2 from a user master password. RK_u is then encrypted with AES and stored locally in IndexedDB. When necessary, the consumer is required to provide the user master password (with salt if necessary) to decrypt the locally encrypted RK_u. Following the NIST standard [50], we require that the user master password must satisfy three requirements: 1) at least 8 characters in length as a memorized secret, 2) not appear in known dictionaries, and 3) be updated periodically. We remark that in this manner the usage of password does not lead to low-level security as existing password-based solutions. The analysis is postponed to Section 5.1.

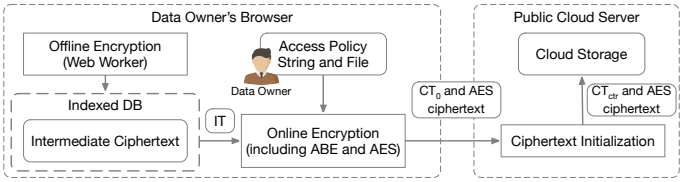


Fig. 4: Data Encryption Procedure of WebCloud

3) Data Encryption. To improve performance, data encryp-

tion procedure is divided into three parts as depicted in Fig. 4. The encryption is in the KEM/DEM setting.

Offline encryption in browser (before an access policy is known): This algorithm processes almost all the costly operations in the encryption algorithm of CP-AB-KEM. On opening the WebCloud website, a Web worker (cf. Section 2.1) is created in background. During idle time, the worker runs the algorithm *Enc.Offline*() to generate a few intermediate ciphertexts IT and keys Key. Idle time is defined as: a) no online encryption part is running, b) no user decryption part is running, and c) no AES encryption or decryption is running. We store $(\mathsf{IT}, \mathsf{Key})$ in sessionStorage, which is erased automatically by browsers after the Web page is closed.

Online encryption in browser (after an access policy string and a file are given): The data owner uploads a file and specifics an access policy string, e.g., "(Employee and IT department) or Manager". Note that the policy string is a flexible logic expression, which supports "and", "or" and "()" operations. The policy string is converted to an access structure (M, ρ). Meanwhile, a pair of $(\mathsf{IT}, \mathsf{Key})$ is obtained from sessionStorage. On input the access policy (M, ρ) and the intermediate ciphertext IT, the algorithm *Enc.Online*() generates ABE ciphertext CT_0. The input file is encrypted with AES, using a 128-bit key derived from Key and a random initialization vector (IV). All necessary data including ABE ciphertext CT_0 and AES ciphertext, are packed together and forms a new file before uploading to the cloud server.

Ciphertext initialization in cloud: On receiving the uploaded file, the cloud parses ABE ciphertext CT_0 from the file. It processes CT_0 to $\mathsf{CT}_{\mathsf{ctr}}$ by calling the algorithm *CTInit*() and replaces CT_0 with $\mathsf{CT}_{\mathsf{ctr}}$ in the file. Finally, the updated file is stored in the cloud.

The above-mentioned algorithms are as follows:
Enc.Offline(PK, N'). On input a PK and a maximum bound of N' rows in any LSSS access structure, the Web worker in data owner's browser picks $3N'+1$ random elements $s, \{\lambda'_i, x_i, t_i\}_{i\in[1,N']} \in \mathbb{Z}_p^*$, and computes $\mathsf{Key} = e(g,g)^{\alpha s}$, $C_0 = g^s$. For $i = 1$ to N', it then computes: $C_{i,1} = w^{\lambda'_i} v^{t_i}, C_{i,2} = (u^{x_i} h)^{-t_i}, C_{i,3} = g^{t_i}$. The Web worker outputs: $\mathsf{IT} = (s, \mathsf{Key}, C_0, \{\lambda'_i, x_i, t_i, C_{i,1}, C_{i,2}, C_{i,3}\}_{i\in[1,N']})$.
Enc.Online$(\mathsf{PK}, (M,\rho), \mathsf{IT})$. On input a public key PK, an LSSS access structure (M,ρ) [51], where M is an $l\times n$ matrix, an intermediate ciphertext IT, data owner's browser picks $n-1$ random elements $(y_2,\ldots,y_n) \in \mathbb{Z}_p^*$ and constructs a vector $\overrightarrow{y} = (s, y_2, \ldots, y_n)^T$. It then computes a vector of shares of s as $(\lambda_1,\ldots,\lambda_l)^T = M\overrightarrow{y}$. For $i=1$ to k, compute $C_{i,4} = \lambda_i - \lambda'_i, C_{i,5} = t_i(x_i - \rho(i))$. The browser outputs: $\mathsf{CT}_0 = ((M,\rho), C_0, \{C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}, C_{i,5}\}_{i\in[1,l]})$.
CTInit$(\mathsf{CT}_0, \mathsf{CSK}_{\mathsf{ctr}})$. On input a ciphertext CT_0 and a cloud secret key $\mathsf{CSK}_{\mathsf{ctr}} = (\mathsf{csk}_{\mathsf{ctr}}, \mathsf{ctr})$, the public cloud computes $C'_0 = C_0^{1/\mathsf{csk}_{\mathsf{ctr}}}$. For $i=1$ to l, it computes: $C'_{i,1} = C_{i,1}^{1/\mathsf{csk}_{\mathsf{ctr}}}$, $C'_{i,2} = C_{i,2}^{1/\mathsf{csk}_{\mathsf{ctr}}}$, $C'_{i,3} = C_{i,3}^{1/\mathsf{csk}_{\mathsf{ctr}}}$, $C'_{i,4} = C_{i,4}/\mathsf{csk}_{\mathsf{ctr}}$, $C'_{i,5} = C_{i,5}/\mathsf{csk}_{\mathsf{ctr}}$. The cloud outputs: $\mathsf{CT}_{\mathsf{ctr}} = ((M,\rho), C'_0, \{C'_{i,1}, C'_{i,2}, C'_{i,3}, C'_{i,4}, C'_{i,5}\}_{i\in[1,l]})$.

4) Data Decryption. As depicted in Fig. 5, file decryption is divided into two parts where the cloud server performs almost all heavy computation.

Outsourced decryption in cloud: On receiving a file download request, the cloud server checks that whether the user has been revoked. If revoked, the cloud rejects the request. Otherwise, it finds the requested file from the cloud storage, parses $\mathsf{CT}_{\mathsf{ctr}}$ from the file and transforms $\mathsf{CT}_{\mathsf{ctr}}$ to TCT by calling the algorithm *Dec.Out*(). The cloud packs TCT and AES ciphertext as a new file and returns the new file as a response to the request.

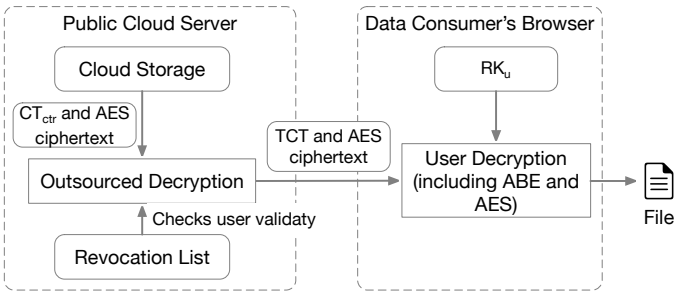


Fig. 5: Data Decryption Procedure of WebCloud

User decryption in browser: In the browser side, TCT is parsed from the response. If the retrieval key RK_u already exists in the browser memory, the key is used directly. Otherwise, the retrieval key RK_u is obtained as in **2) Key Management**. The algorithm *Dec.User*() is invoked to decrypt TCT to obtain encapsulated key Key. We derive the same AES key as in the encryption procedure. Finally, we decrypt the file with AES.

The above-mentioned algorithms are as follows:
Dec.Out$(\mathsf{PK}, \mathsf{CSK}_{\mathsf{ctr}}, \mathsf{TK}_u, \mathsf{CT}_{\mathsf{ctr}})$. On input a public key PK, a cloud secret key $\mathsf{CSK}_{\mathsf{ctr}} = (\mathsf{csk}_{\mathsf{ctr}}, \mathsf{ctr})$, a conversion key $\mathsf{TK}_u = (S, K'_0, K'_1, \{K'_{i,2}, K'_{i,3}\}_{i\in[1,k]})$ for an attribute set S and a ciphertext $\mathsf{CT}_{\mathsf{ctr}}$ for access structure (M,ρ), if S does not satisfy the access structure, the cloud server outputs $\perp$. Otherwise, it calculates a set $I = \{i : \rho(i) \in S\}$ and computers the constants $\{\omega_i \in Z_p\}_{i\in I}$ such that $\sum_{i\in I}\omega_i \cdot M_i = (1,0,\ldots,0)$, where M_i is the i-th row of the matrix M. It then computes:

$$B = e(w^{\sum_{i\in I} C'_{i,4}\omega_i}, K'_1) \cdot \prod_{i\in I} e(C'_{i,1}, K'_1)^{\omega_i} \cdot \prod_{i\in I}(e(C'_{i,2}\cdot u^{C'_{i,5}}, K'_{j,2}) \cdot e(C'_{i,3}, K'_{j,3}))^{\omega_i}$$

The cloud outputs $\mathsf{TCT} = (e(C'_0, K''_0)/B)^{\mathsf{csk}_{\mathsf{ctr}}} = e(g,g)^{\alpha s/\tau}$.
Dec.User$(\mathsf{TCT}, \mathsf{RK}_u)$. On input a TCT and a retrieval key $\mathsf{RK}_u = \tau$, data consumer's browser outputs the encapsulated key $\mathsf{Key} = \mathsf{TCT}^{\mathsf{RK}_u} = e(g,g)^{\alpha s}$.

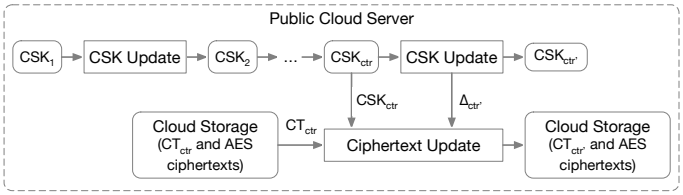


Fig. 6: Cloud Secret Key Update Procedure of WebCloud

5) Cloud Secret Key Update. The cloud secret key CSK is of great importance to the revocation mechanism. If CSK leaks, the user revocation functionality is in vain. Therefore,

we update CSK periodically or in emergency situations (e.g., CSK is leaked or stolen).

Cloud Secret Key Update: This procedure is depicted in Fig. 6. Assume current cloud secret key is $\mathsf{CSK}_{\mathsf{ctr}}$. When $\mathsf{CSK}_{\mathsf{ctr}}$ is required to be updated, the public cloud server invokes the algorithm *CSKUpdate*() to generate a new key $\mathsf{CSK}_{\mathsf{ctr}'}$ and an increment $\Delta_{ctr'}$. The cloud then updates all stored ABE ciphertexts $\mathsf{CT}_{\mathsf{ctr}}$ to $\mathsf{CT}_{\mathsf{ctr}'}$ by calling the algorithm *CTUpdate*(). Once all ciphertexts have been updated, the cloud sets current secret key to $\mathsf{CSK}_{\mathsf{ctr}'}$, and deletes $\mathsf{CSK}_{\mathsf{ctr}}$ and all old ABE ciphertexts $\mathsf{CT}_{\mathsf{ctr}}$ from its storage.

The above-mentioned algorithms are as follows:

CSKUpdate$(\mathsf{PK}, \mathsf{CSK}_{\mathsf{ctr}})$. On input a public key PK and a cloud secret key $\mathsf{CSK}_{\mathsf{ctr}} = (\mathsf{csk}_{\mathsf{ctr}}, \mathsf{ctr})$ where $\mathsf{ctr} \in \{1, 2, \ldots\}$, the cloud server updates the counter $\mathsf{ctr}' = \mathsf{ctr}+1$ and picks a random element $\beta_{\mathsf{ctr}'} \in \mathbb{Z}_p^*$, computes $\mathsf{csk}_{\mathsf{ctr}'} = \mathsf{csk}_{\mathsf{ctr}} \cdot \beta_{\mathsf{ctr}'} = \prod_{i=1}^{\mathsf{ctr}'} \beta_i$. The cloud outputs the updated cloud secret key $\mathsf{CSK}_{\mathsf{ctr}'} = (\mathsf{csk}_{\mathsf{ctr}'}, \mathsf{ctr}')$ and the increment $\Delta_{\mathsf{ctr}'} = \beta_{\mathsf{ctr}'}$.

CTUpdate$(\mathsf{ctr}', \mathsf{CT}_{\mathsf{ctr}'-1}, \Delta_{\mathsf{ctr}'})$. On input an updated counter $\mathsf{ctr}' \in \{2, 3, \ldots\}$, a ciphertext $\mathsf{CT}_{\mathsf{ctr}'-1} = ((M, \rho), C_0, \{C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}, C_{i,5}\}_{i\in[1,l]})$ and an increment $\Delta_{\mathsf{ctr}'} = \beta_{\mathsf{ctr}'}$, the cloud server computes $C_0' = C_0^{1/\beta_{\mathsf{ctr}'}}$. For $j = 1$ to l, the cloud computes $C_{i,1}' = C_{i,1}^{1/\beta_{\mathsf{ctr}'}}, C_{i,2}' = C_{i,2}^{1/\beta_{\mathsf{ctr}'}}, C_{i,3}' = C_{i,3}^{1/\beta_{\mathsf{ctr}'}}, C_{i,4}' = C_{i,4}/\beta_{\mathsf{ctr}'}, C_{i,5}' = C_{i,5}/\beta_{\mathsf{ctr}'}$. It outputs: $\mathsf{CT}_{\mathsf{ctr}} = ((M, \rho), C_0', \{C_{i,1}', C_{i,2}', C_{i,3}', C_{i,4}', C_{i,5}'\}_{i\in[1,l]})$.

6) Key and User Revocation. WebCloud supports both key and user revocation, and does not support attribute revocation. To revoke a user key, PKG runs *KG*(S, MSK) to generate a new secret key SK_u' and *KG.Random*$(\mathsf{PK}, \mathsf{SK}_u')$ to obtain RK_u' and TK_u'. PKG replaces RK_u with RK_u' and distributes TK_u' to the public cloud, who deletes previous transformation key TK_u directly. The key revocation is taken effect immediately after the cloud updates its transformation key. To revoke a user from the system, PKG sends a revocation request to the public cloud server to revoke a data consumer, where the cloud inserts an entry to the revocation list $\mathbb{L}$ by calling the algorithm *Revoke*(). The revocation is taken effect immediately after the insertion. On receiving a user's file download request, the cloud compares the user identity u against the list $\mathbb{L}$ and rejects the request if a match is found. Without the help of the cloud, data consumers cannot decrypt files individually. The size of the list $\mathbb{L}$ is the same as the number of revoked user in the system. Many efficient algorithms exist for finding an element from a(n) (ordered) list e.g., binary search or hash table.

The above-mentioned algorithms are as follows:

Revoke$(u, \mathbb{L})$. On input a user identity u, and a revocation list $\mathbb{L} = \{(id)\}$ where id is the user identity, the cloud server adds an entry (u) to the list $\mathbb{L}$, i.e., $\mathbb{L}' = \mathbb{L} \cup \{(u)\}$.

4.2 A Tailored CP-AB-KEM for WebCloud

Ciphertext-policy attribute-based key encapsulation mechanism (CP-AB-KEM) is an important component for WebCloud (Section 4.1). It simultaneously achieves offline encryption and outsourced decryption, robust and immediate user revocation, while only a small number of computations are left to the user. The proposed CP-AB-KEM derives from the offline encryption and outsourced decryption techniques in [19], [20], [52], and combines the immediate user revocation mechanism in [53]. For completeness, we give the syntax in Appendix A and elaborate the scheme in the supplementary material. We emphasize that this CP-AB-KEM is useful in many scenarios.

Correctness: We require the standard correctness property: for an attribute universe U, a user identity U and $\lambda, N, N' \in \mathbb{N}$, for all $(\mathsf{PK}, \mathsf{MSK}, \mathsf{CSK}_1) \in Setup(\lambda, U)$, all $\mathsf{SK}_u \in KG(S, \mathsf{MSK})$, all $(\mathsf{RK}_u, \mathsf{TK}_u) \in KG.Random(\mathsf{SK}_u)$, all $\mathsf{IT} \in Enc.Offline(\mathsf{PK}, N')$, all $\mathsf{CT}_0 \in Enc.Online(\mathsf{PK}, (M, \rho), \mathsf{IT})$, all $\mathsf{CT}_{ctr} \in CTInit(\mathsf{CT}_0, \mathsf{CSK}_{\mathsf{ctr}})$, all $(\mathsf{CSK}_{\mathsf{ctr}'}, \Delta_{\mathsf{ctr}'}) \in CSKUpdate(\mathsf{PK}, \mathsf{CSK}_{\mathsf{ctr}})$, all $\mathsf{CT}_{\mathsf{ctr}'} \in CTUpdate(\mathsf{ctr}', \mathsf{CT}_{\mathsf{ctr}'-1}, \Delta_{\mathsf{ctr}'})$, all $\mathsf{TCT} \in Dec.Out(\mathsf{PK}, \mathsf{CSK}_{\mathsf{ctr}}, \mathsf{TK}_u, \mathsf{CT}_{\mathsf{ctr}})$, if S satisfies (M, ρ) and the user u was not revoked, $Dec.User(\mathsf{TCT}, \mathsf{RK}_u)$ outputs the encapsulated Key.

Security: The security proof of the proposed CP-AB-KEM scheme is given in Appendices B and C.

4.3 Security Enhancement of PKG

In the WebCloud system, all users' secret key are derived from the master secret key MSK, which is stored in the trusted PKG. In reality, a single point of failure, e.g., loss of MSK, will immediately lead to system failure. It is of great importance to provide simple mechanisms to enhance the security of MSK and the system.

An effective way is secret sharing, i.e., splitting the MSK into multiple pieces. Without loss of generality, we consider a (t, n) threshold scheme.

As shown in Fig. 7, there are 1 root PKG, n child PKG_i $(1 \le i \le n)$ and a combiner PKG_c. PKG is responsible for generating PK, MSK and n shares of α, and distributes the i-th share MSK_i to PKG_i. When generating a user's secret key SK_u, PKG_i generates a partial secret key $\mathsf{SK}_{u,i}$ using its share. The combiner PKG_c combines any t partial keys to SK_u, and invokes *KG.Random* to obtain TK_u and RK_u.

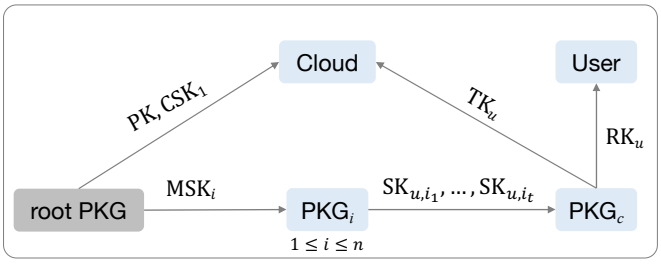


Fig. 7: Secret Sharing of Master Secret Key

† The root PKG exits the system after setup. Any t child PKG_i, the combiner PKG_c, the cloud and optionally the user, should stay online.

Note that we only change system setup and user secret key generation phases. Other phases are processed as in Section. 4.1. After the setup phase, the root PKG is removed from the system. To make the system work, any t out of n child PKG_i and also the combiner PKG_c should stay online. This makes the combiner PKG_c an attractive attack target. In Section 5.1, we show that compromise of PKG_c does not reveal MSK. However, user secret keys generated after the compromise may be leaked. Offline of PKG_c or great than $n-t$ child PKG_i will affect the usability, but not the security

of the system, i.e., the user secret key generation service will be temporarily unavailable.

In this work, we use Shamir's Secret Sharing [54]. Let $\ell_i(x)$ denote the Lagrange polynomial: $\ell_i(x) = \prod_{\substack{m=1 \\ m \neq i}}^{t} (\frac{x - x_m}{x_i - x_m})$.

Following algorithms are defined:

Setup$'(\lambda, U)$. On input a security parameter λ and an attribute universe U, PKG performs following actions:

1) Chooses a bilinear map $D = (\mathbb{G}, \mathbb{G}_T, e, p)$, where $p \in \Theta(2^\lambda)$ is the prime order of groups $\mathbb{G}$ and $\mathbb{G}_T$. The attribute universe U consists of elements in $\mathbb{Z}_p^*$.
2) Chooses random generators $g, h, u, v, w \in \mathbb{G}$, picks two random elements $\alpha, \beta_1 \in \mathbb{Z}_p^*$.
3) Determines the parameters n (number of child PKGs) and t (number of minimal PKGs to construct a user secret key).
4) Selects $t-1$ random elements $a_1, \ldots, a_{t-1}$ from $\mathbb{Z}_p^*$, and constructs $f(x) = \alpha + a_1 x + \cdots + a_{t-1} x^{t-1}$. It then generates n random points (x_i, y_i) ($i \in \{1, \ldots, n\}$) where x_i is randomly selected from $\mathbb{Z}_p^*$ and $y_i = f(x_i)$.
5) Erases α and also the polynomial $f(x)$ completely.
6) Sets a counter $\mathsf{ctr} = 1$. Finally, PKG outputs: $\mathsf{PK} = (D, g, h, u, v, w, e(g,g)^\alpha)$, n partial master secret key $\mathsf{MSK}_i = (x_i, y_i)$ and $\mathsf{CSK}_1 = (\mathsf{csk}_1 = \beta_1, \mathsf{ctr})$.

KG$'(S, \mathsf{MSK}_i)$. On input an attribute set $S = (A_1, \ldots, A_k)$ and a point $\mathsf{MSK}_i = (x_i, y_i)$, PKG_i picks a random element $r_i \in \mathbb{Z}_p^*$ and computes $K_{i,0} = g^{y_i} w^{r_i}$, $K_{i,1} = g^{r_i}$. For j from 1 to k, it picks random $r_{i,j} \in \mathbb{Z}_p^*$ and computes $K_{i,j,2} = g^{r_{i,j}}, K_{i,j,3} = (u^{A_j} h)^{r_{i,j}} \cdot v^{-r_i}$. PKG outputs a partial secret key $\mathsf{SK}_{u,i} = (S, x_i, K_{i,0}, K_{i,1}, \{K_{i,j,2}, K_{i,j,3}\}_{j \in [1,k]})$.

Combine$(\mathsf{SK}_{u,i_1}, \ldots, \mathsf{SK}_{u,i_t})$. On input t partial user secret keys SK_{u,i_m} where $m \in [1,t]$, $i_m \in [1,n]$ and all i_m are different, the combiner PKG_c first computes t quantities $\ell_{i_m}(0)$. Let $r = \sum_{m=1}^{t} r_{i_m} \cdot \ell_{i_m}(0)$. Let $r_j = \sum_{m=1}^{t} r_{i_m,j} \cdot \ell_{i_m}(0)$. It then computes $K_0 = \prod_{m=1}^{t} (K_{i_m,0})^{\ell_{i_m}(0)} = g^\alpha \cdot w^r$, $K_1 = \prod_{m=1}^{t} (K_{i_m,1})^{\ell_{i_m}(0)} = g^r$. For j from 1 to k, it computes: $K_{j,2} = \prod_{m=1}^{t} (K_{i_m,j,2})^{\ell_{i_m}(0)} = g^{r_j}$ and $K_{j,3} = \prod_{m=1}^{t} (K_{i_m,j,3})^{\ell_{i_m}(0)} = (u^{A_j} h)^{r_j} \cdot v^{-r}$. The user secret key is $\mathsf{SK}_u = (S, K_0, K_1, \{K_{j,2}, K_{j,3}\}_{j \in [1,k]})$. The combiner invokes *KG.Random*$(\mathsf{PK}, \mathsf{SK}_u)$ to generate RK_u and TK_u.

5 Analysis of WebCloud

5.1 Security Analysis

In this section, we analyze WebCloud in the security models defined in Section 3.2.

Passive Man-in-the-Middle. In this attack model, the attacker observes all network traffic, thus he may be able to find out the access policy of each file and notice that a data consumer is uploading or downloading files. But due to the IND-CPA property of used encryption schemes (CP-AB-KEM and AES-GCM/AES-CBC), the adversary cannot learn the content of encrypted files.

Web Attacker Model. It is typically impossible to prove the security of a complex Web application in the Web attacker model – even Google's websites suffer from vulnerabilities under this attacker model. However, in WebCloud, the main attack target would be the cryptographic keys. We highlight the protection mechanisms of keys in WebCloud:

Security of the Master Secret Key. We discuss the security of the secret sharing in Section 4.3. PKG invokes the algorithm *Setup*$'$ to generate and distribute shares of α. After that, the quantity α, all n points and the polynomial $f(x)$ are erased completely. In the *Combine* algorithm, the combiner takes t partial user secret key where each key contains a share. Note that each share is given in the form $(x_i, g^{y_i} w^{r_i})$ and the Lagrange polynomials are computed in the exponent. Since r_i are randomly selected by PKG_i, $g^{y_i} w^{r_i}$ is indistinguishable from a random group element. The finally computed $K_0 = g^\alpha w^r$ is also indistinguishable from a random element. This is actually an Elgamal ciphertext. Thus, the combiner cannot extract α from its inputs. Certainly, t PKG_i can corrupt to construct the polynomial $f(x)$ to obtain α. This can be migrated by increasing the value of t and letting each PKG_i be a different company.

Security of retrieval keys. The retrieval key RK_u is obtained from PKG after the user is authenticated with the MFAKE protocol [43] and only used in the browser memory. Meanwhile, if RK_u is not used for a period of time (e.g., 30 minutes), it is erased from the memory.

Optionally, RK_u can be encrypted with a user master password and stored in IndexedDB. When necessary, i.e., on decrypting files, the user is required to input the user master password to decrypt RK_u. Note that our usage of password is different from the existing password-based solutions as mentioned in Section 1, where the password-protected file is transmitted over a public channel and the password cannot be changed once the file is sent, where an offline brute-force search is possible in that scenario. In WebCloud, we only store the encrypted retrieval keys locally and require the password to be updated periodically, e.g., 7 days. So, obtaining the ciphertexts should be difficult, and a short update period with possible usage of salt will make offline brute-force search attach useless.

Security of the encapsulated key. During offline encryption phase, several intermediate ciphertexts IT and encapsulated key Key are generated. When encrypting a file, an AES key is derived from the encapsulated key Key. If an adversary obtains Key, he can decrypt the file. Thus, we store IT and Key in sessionStorage, which will be erased after the Web page is closed, to avoid being persistent.

Security of AES keys. The AES algorithm is used in WebCloud for many times. The AES keys are derived using the Web Cryptography API with the field extractable set to false, meaning that the keys cannot be obtained by JavaScript codes and exported outside of the browser.

Security of the proposed CP-ABE. The security of the proposed CP-ABE is given in Theorem 1 and the the security proof is postponed to Appendix B.

Theorem 1. *The proposed ciphertext-policy attribute-based encryption mechanism can achieve Data Security Against User Collusion in Definition 1, Data Security Against Cloud Server in Definition 2, User Revocation Validity in Definition 3, if the CP-ABE scheme in [55] is selective Chosen Plaintext Attack (CPA)-secure.*

5.2 Features Analysis

We highlight a few nice properties of WebCloud.

TABLE 2: Comparison of WebCloud with Related Works.

Scheme	Security Level	Cross-Platform	Revocation		File Sharing	Performance	
			Mechanism	Immediate		Single Receiver	N Receivers
Password-Based solutions	Low	×	×	×	Inflexible	Fast	Depends on N
RSA-AES paradigm	High	×	CRL, OCSP	×	Inflexible	Fast	Depends on N
Ours	High	√	Server-Aided	√	Flexible	Fast	Fast

Data Privacy. In WebCloud, all files are encrypted and decrypted locally, i.e., in browsers. The cloud only sees ciphertext and deals with ciphertext.

Flexible File Sharing. By assigning an access policy to each file, an encrypted file can be decrypted by multiple data consumers as long as their attributes satisfy the access policy. The user only encrypts a file one time and the cloud only stores one copy of each encrypted file. In a corporation scenario, an employee can share a file with all managers of the sales department by setting the access policy to "Sales Department and Manager", without the need to find out who are the concrete receivers or their public keys as in the password-based solutions and RSA-AES paradigm. If a new employee is hired, he can decrypt all the ciphertexts that match his/her attributes immediately. In existing schemes, manually encryption and sharing to the employee are required.

User and Key Revocation. The revocation mechanism is efficient and immediately effective. The cloud server revokes a data consumer by adding the consumer to the revocation list $\mathbb{L}$. On receiving a consumer's download request, the cloud checks the list $\mathbb{L}$ and rejects revoked consumers' requests. The key revocation is achieved by requiring PKG to regenerate a new SK_u and related $(\mathsf{RK}_u, \mathsf{TK}_u)$, and distributes RK_u to the user and TK_u to the cloud server.

Usability and Efficiency. WebCloud only requires a Web user agent and does not require any additional software, Java applet or browser plugin. WebCloud is fully optimized in two aspects: a) The proposed CP-AB-KEM scheme is very suitable for browser side cryptography, which moves almost all costly computations offline and outsourced, and b) The implementation fully utilize power of modern Web techniques, including WebAssembly, Web Cryptography API, Web Workers and Web storage. The functionalities and performance are tested in major browsers on different devices, including both laptops and mobiles (cf. Section 6.2).

Cloud Server Key-Exposure Resistance. The cloud secret key CSK is important to the revocation mechanism. If CSK leaks, the revocation functionality is useless. Thus, WebCloud introduces key-exposure resistance property for CSK. Concretely, the cloud updates CSK periodically (or when key is leaked). Meanwhile, it updates all stored ABE ciphertexts and deletes old ciphertexts when the CSK is updated.

5.3 Comparisons with Related Work

We compare WebCloud with existing client-side cloud data protection solutions in terms of security, usability, revocation, and performance (cf. Table 2). The comparison of the proposed CP-ABE scheme and other CP-ABE schemes are given in Section 6.

Comparing with existing solutions, WebCloud has two main innovations:

1) We propose and adopt a dedicated CP-AB-KEM scheme as the encryption algorithm. This provides WebCloud with high-level security and flexible file sharing.
2) We use Web context as the encryption environment. This provides WebCloud with cross-platform.

This provides many practical advantages for WebCloud.

Security. The security of password-based solutions relies on low-entropy passwords, thus the security level is typically low, i.e., tens of bits [10]. In contrast, the security of RSA-AES paradigm and WebCloud bases on the cryptographic keys. When adopting appropriate parameters and key lengths, i.e., RSA-3072 and AES-128, the security level achieves at least 128-bit. In above analysis, we assume the adopted cryptographic primitives are provably secure.

Usability. Both the existing client-side encryption solutions and WebCloud allow users to encrypt and decrypt on their devices. However, for above mentioned client-side solutions, they require addition software, browser plugins for each operating system and browser. In contrast, WebCloud doesn't require any additional software or plugins. It effectively combines state-of-the-art Web techniques, including Web Cryptography API, WebAssembly, thus can be accessed in any Web context, e.g., Web browsers in desktop computers and mobile phones, WebView in Android applications, native application using Electron framework. Usually, Web browsers are installed by default on today's devices.

Revocation. Password-based encryption schemes have no user management or revocation mechanisms. Relying on PKI, RSA-AES paradigm can adopt Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP) to achieve certificate revocation, i.e., user revocation. WebCloud adopts a server-aided revocation mechanism, which supports both user and key revocation. The time required for the revocation to take effect depends on the response of the cloud server, which should take effect immediately in most cases.

File Sharing. Password-based solutions have no file sharing or access control mechanisms. Users need to manually share the data to all receivers. For RSA-AES paradigm, the sender must collect all receivers' public keys and encrypt under those keys, resulting in large storage space and increased workload. Leveraging CP-ABE, WebCloud can encrypt a file under a possibly complicated access policy of receivers. Even a future-joined user can decrypt the ciphertexts without any interaction and workload of the data sender.

Most of research on ABE focused on theoretical aspects, including proposing new constructions, reducing the computation complexity, etc. In this work, we aim at proposing a practical cloud storage solution and implementing

TABLE 3: Theoretical Comparison of the Proposed CP-AB-KEM Scheme.

Scheme	Enc.Online	Dec.User	Transfer Size (KG)	Transfer Size (Enc)	Transfer Size (Dec)	Revocable
[55]	$(5l+2)$Exp	$(\|I\|)$Exp + $(3\|I\|+1)$P	0	0	$\|\mathbb{A}\|+(3l+1)\|\mathbb{G}\|+1\|\mathbb{G}_T\|$	×
[20]	0	$(\|I\|+1)$Exp+$(3\|I\|+2)$P	0	0	$\|\mathbb{A}\|+(3l+1)\|\mathbb{G}\|+2l\|\mathbb{Z}_p\|$	×
[52]	3Exp	3Exp	0	$(3+3l)\|\mathbb{G}\|+3l\|\mathbb{Z}_p\|$	$\|\mathbb{A}\|+1\|\mathbb{G}\|+1\|\mathbb{G}_T\|$	×
[21]	1Exp	1Exp	$2((2+2y)\|\mathbb{G}\|+(2+2y)\|\mathbb{Z}_p\|)$	$(4+6l)\|\mathbb{G}\|+4l\|\mathbb{Z}_p\|$	$\|\mathbb{A}\|+1\|\mathbb{G}_T\|$	×
Ours	0	1Exp	0	0	$\|\mathbb{A}\|+1\|\mathbb{G}_T\|$	√

‡ Exp and P denote a module exponentiation and a pairing computation, respectively. y, l and I indicate the number of attributes, the access policy size, and the set that satisfies decryption requirement, respectively. $|\mathbb{A}|$, $|\mathbb{G}|$ and $|\mathbb{G}_T|$ denote the size of an access structure, an element in $\mathbb{G}$ and $\mathbb{G}_T$, respectively.

a prototype to prove the effectiveness of the solution. We believe that after some serious engineering work, the solution and the prototype can be turned into a real application. Therefore, we require that the underlying CP-ABE is secure and efficient, and supports user revocation. For this reason, we combine state-of-the-art techniques to obtain a custom scheme, which has a few attractive properties, including offline encryption, outsource decryption, and user (and key) revocation.

Performance. The existing solutions and WebCloud have a fast encryption and decryption speed when encrypting to a single receiver. Password-based solutions is fast and only performs Key Derivation Function and symmetric encryption. RSA-AES paradigm includes RSA encryption and AES encryption. However, when encrypting to a set of users, the workload is proportional to the number of receivers. In WebCloud, encryption (*Enc.Online()*) and decryption (*Dec.User()*) are fast even under hundreds of attributes (tens of milliseconds, cf. Section 6.2) , and irrelevant to the number of receivers.

6 Evaluations

In this section, we evaluate the performance of WebCloud.

6.1 Theoretical Analysis for CP-AB-KEM

As a core component of WebCloud, the operation of CP-ABE-KEM is the most resource-consuming part of WebCloud. Here we evaluate its complexity and make comparisons with similar schemes.

The online computation cost of the PKG, the data owner and the data consumer refers to the execution time of *KG.Online*, *Enc.Online* and *Dec.User*. Table 3 compares the numbers of modular exponentiations and pairing operations in our construction with those in an efficient ABE scheme [55], online/offline ABE [20] and the outsourced ABE schemes [52], [21].

Our scheme is the first to achieve offline encryption, outsourced decryption and user revocation simultaneously. The PKG and the users only need to perform a small number of online computations, which highly improves the overall performance. During encryption, the online computation for users needs no exponentiations, which is less than [55], [52], [21]. The work [20] also do not need modular exponentiation for encryption, but has large computational cost in decryption. During decryption, the computation is only 1 exponentiation, which is less than [55], [20], [52] and equal to [21]. Compared with above schemes, our scheme has a considerable advantage in efficiency.

6.2 Performance of WebCloud

General Remarks. WebCloud has no specific requirement for the underlying cloud storage systems. It can be applied to all cloud storage systems. We implement WebCloud based on ownCloud (version 10.0.10), which is an open source cloud collaboration platform. We carefully review the codes of ownCloud on its PHP framework, file process, user management and debug method. Meanwhile, we also investigate a few third party open source projects that used in ownCloud, mainly jQuery file upload plugin [56] and sabre/dav framework [57].

We extended ownCloud in a few ways: We implement a C++ module and a JavaScript module. The C++ codes are compiled by clang on macOS or g++ on Ubuntu with flags "-O3". The Web server is Apache 2.4.37 with MySql 8.0.13 and PHP 7.2.13. The C++ module is embedded into both the server side (as a PHP module) and browser side (as a WebAssembly module). The WebAssembly codes are compiled by emcc with flags "-s WASM=1 -O3". Further, we modify the user interface, file upload and download process of ownCloud. As a Web application, WebCloud gives visual feedback to users according to the actual running stage of encryption and decryption. Furthermore, we pack WebCloud into desktop applications, with the Electron framework [58]. The packed software supports Mac, Windows and Linux. We also implement WebCloud on Android applications using the WebView component.

Benchmarks. We evaluate the performances of WebCloud on various devices and platforms. Following the standard testing requirements, we turned off hardware-related options, such as Turbo Boost and hyperthreading. Table 4 lists detailed information of used devices.

Benchmark of WebCloud: The complexity of access policy affects computational and communicational cost. Thus, we generate access policy in the form of (S_1 and ... and S_y) to simulate the worst situation, where S_i is an attribute. We set 10 distinct access policies in this from with y increasing from 10 to 100, and repeat each instance 100 times and take the average running time. Each instance is generated randomly. All presented time is in milliseconds (ms).

Fig. 8 presents the benchmark result of WebCloud. Fig 8(a) presents running time of *Enc.Offline* in browsers. Though algorithm *Enc.Offline* is executed offline, its performance is also efficient in most browsers. For laptop browsers, the running time is about 41∼632 ms. For mobiles browsers, the running time is about 65∼2,661 ms. Actually, except Chrome on Huawei Honor 10, other mobile browsers have running time within 600 ms. Fig 8(b) presents running time of *Enc.Online* in browsers. For laptop browsers, the run-

TABLE 4: Devices and Platforms in Our Experiments.

	Client Device (64-bit)	Browser
1	Macbook Pro Intel Core i7@2.2 GHz 4-core	Safari/605.1.15, Google Chrome 71.0.3578.98 and Firefox 64.0
2	Dell Intel Core i5-6440HQ@2.6 GHz 4-core	Microsoft EdgeHTML 17.17134
3	iPhone 6s Plus ARMv8-A@1.85 GHz 2-core	Safari/604.1, Google Chrome 71.0.3578.89 and Firefox 14.0 (12646)
4	Huawei Honor 10 4×Cortex A73@2.36 GHz+4×Cortex A53@1.8 GHz	Chrome 70.0.3538.110
	Server Device (64-bit)	**Operating System**
5	Intel Xeon CPU E5-2609 v4@1.7 GHz 8-core	Ubuntu 16.04.5 LTS
6	Macbook Pro Intel Core i7@2.2 GHz 4-core	macOS Mojave 10.14.2

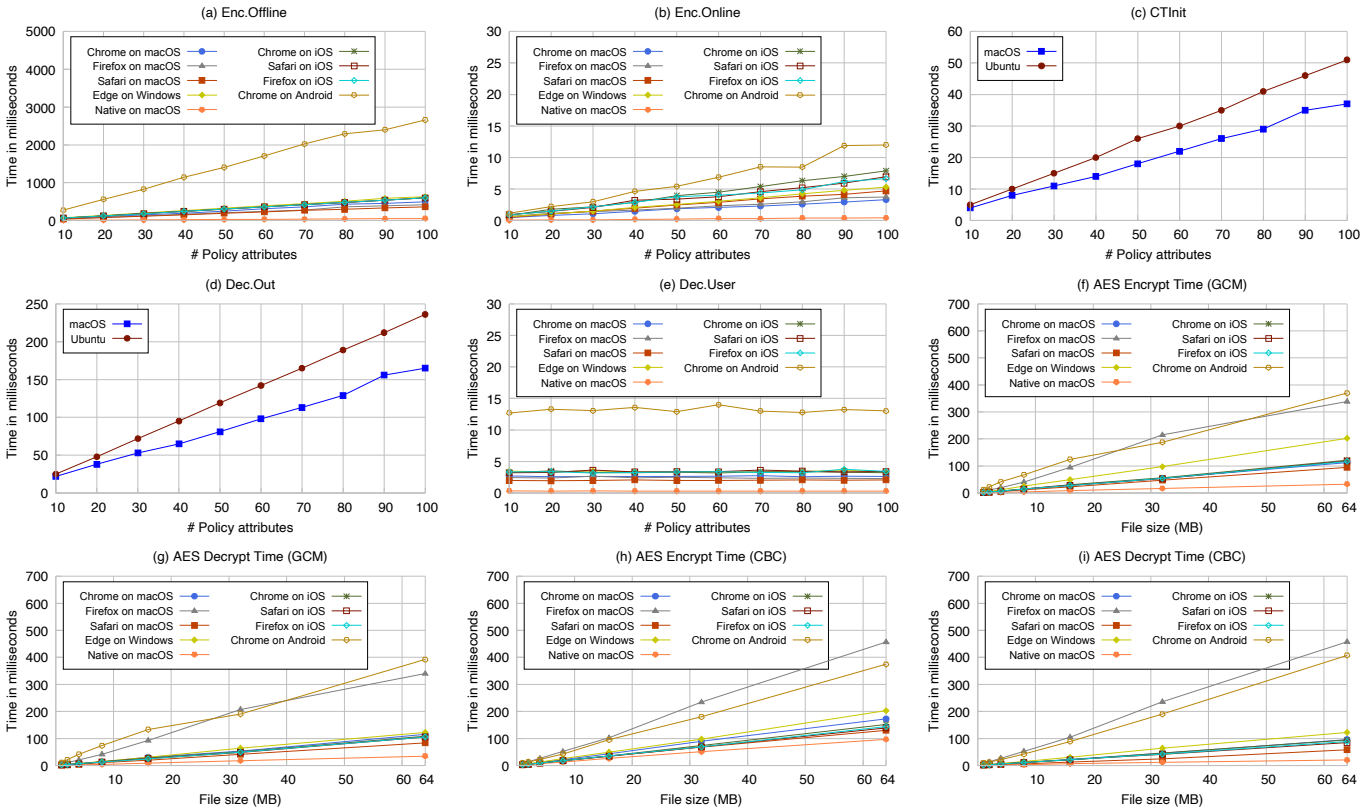


Fig. 8: Experiment Results of WebCloud

ning time is about 0.46∼5.32 ms. Most notably, even when the size of access structure increases to 100, the online encryption part only requires 3.31 ms in Chrome. For mobiles browsers, the running time is about 0.81∼11.99 ms. Fig 8(c) presents running time of *CTInit* on the cloud server. On macOS, the running time is about 4∼37 ms. On Ubuntu, the running time is about 5∼52 ms. Fig 8(d) presents running time of *Dec.Out* on the cloud server. On macOS, the running time is about 22∼165 ms. On Ubuntu, the running time is about 25∼236 ms. Fig 8(e) presents running time of *Dec.User* in browsers. For laptop browsers, the running time is about 1.97∼3.35 ms. For mobiles browsers, the running time is about 3.20∼13.99 ms. Actually, except Chrome on Huawei Honor 10, other browsers have running time within 4 ms.

Benchmark of Data Encryption/Decryption: We also benchmarked the data encryption/decryption routines, namely key deriving, encryption and decryption of AES. We omit the key deriving routine in Fig 8, which is small and stable in different browsers. Its running time is about 0.73∼2.15 ms for most browsers except Microsoft Edge, which does not support PBKDF2 key deriving mechanism. We instead use the SJCL library [25], resulting 151 ms latency for Edge. The AES encryption and decryption performance is tested with different file sizes (1, 2, 4, 8 and 16 megabyte, respectively). All files are generated randomly. Fig 8(f) - (i) present running time of AES encryption and decryption in the GCM and the CBC modes, respectively. As for the GCM mode, encrypting 64 MB data in Safari costs 95 ms, with only 84 ms for decryption.

Benchmark for Large File Operations: We further tested files with sizes 128, 256, 512 and 1,024 MB in the Chrome browser on macOS with an Apache server running on macOS. Each instance is executed 100 times and the average result is taken. Table 5 presents the total execution time of encryption and decryption. Note that we only computes the actually time that will be noticed by users. The file transmission time is omitted since it mainly depends on the network. For encryption, only *Enc.Online*, AES key deriving

TABLE 5: Benchmarks for Large Files Operations.

	Encryption (ms)						Decryption (ms)					
Size (MB)	Enc.Online	AES Key Derving	AES-GCM	Total	AES-CBC	Total	Dec.Out	Dec.User	AES-GCM	Total	AES-CBC	Total
128	1.06	0.73	407.59	409.38	506.44	508.23	1.08	2.61	399.68	403.37	346.99	350.68
256	0.98	0.78	740.03	741.79	1,017.46	1,019.22	1.05	2.68	724.40	728.13	702.13	705.86
512	1.10	0.90	1,474.53	1,476.53	2,010.03	2,012.03	1.10	2.64	1,437.93	1,441.67	1,380.45	1,384.19
1,024	1.00	0.79	3,081.71	3,083.50	4,064.27	4,066.06	1.07	2.60	3,854.04	3,857.71	2854.58	2,858.25

‡ Results obtained in Chrome on macOS with an Apache server running on macOS. The access policy contains 30 attributes, connected by "AND" ($S_1 \wedge S_2 \wedge \ldots \wedge S_{30}$).

and AES encryption are counted. For decryption, *Dec.Out* and *Dec.User* are counted. As shown in Table 5, the proposed CP-AB-KEM scheme only adds less than 1% overhead during encryption and decryption. The AES encryption and decryption occupy most of the execution time. Remarkably, encrypt a 1 GB file uses about 3.1/4.1 seconds, while decryption costs 3.9/2.8 seconds, in the GCM/CBC mode.

Applications on Android and PCs: We pack WebCloud into a desktop application on macOS and a mobile application on Android. We perform the benchmark of CP-AB-KEM and AES as above. The benchmark results have no notable differences with the results in the browser on the same platform. This is because that the browsers use the same engine, typically WebKit or Blink. The benchmark results are omitted.

Comparisons with Native Libraries: We implement all algorithms in C++ using MCL-C library and OpenSSL library. In Fig 8 (a)(b)(e)(f)(g)(h)(i), we present the running time of CP-AB-KEM and AES in C++ on macOS. Running algorithms natively is faster. Algorithms in CP-AB-KEM are about 7∼10 times faster than running that in browsers on macOS. Native AES encryption is about 1.78 times faster and decryption is 4.75 times faster. Overall, the algorithms run faster with native library, which is reasonable.

7 Conclusion

We propose WebCloud, a practical client-side encryption solution for public cloud storage in the Web setting, where users do cryptography with only browsers. We analyze the security of WebCloud and implement WebCloud based on ownCloud and conduct a comprehensive performance evaluation. The experimental results show that our solution is practical. As an interesting by-product, the design of WebCloud naturally embodies a dedicated CP-AB-KEM scheme, which is useful in many other applications.

References

[1] "Vulnearability and threat in 2018," Skybox Security, Tech. Rep., 2018. [Online]. Available: https://lp.skyboxsecurity.com/WICD-2018-02-Report-Vulnerability-Threat-18_Asset.html

[2] D. Lewis, "icloud data breach: Hacking and celebrity photos," Duo Security, Tech. Rep., September 2014. [Online]. Available: https://www.forbes.com/sites/davelewis/2014/09/02/icloud-data-breach-hacking-and-nude-celebrity-photos

[3] T. Hunt, "Hacked dropbox login data of 68 million users is now for sale on the dark web," Tech. Rep., September 2016. [Online]. Available: https://www.troyhunt.com/the-dropbox-hack-is-real/

[4] "Amazon data leak," ElevenPaths, Tech. Rep., November 2018. [Online]. Available: https://www.elevenpaths.com/amazon-data-leak/index.html

[5] K. Korosec, "Data breach exposes trade secrets of carmakers gm, ford, tesla, toyota," TechCrunch, Tech. Rep., July 2018. [Online]. Available: https://techcrunch.com/2018/07/20/data-breach-level-one-automakers/

[6] M. Grant, "$93m class-action lawsuit filed against city of calgary for privacy breach," Tech. Rep., October 2017. [Online]. Available: http://www.cbc.ca/news/canada/calgary/city-calgary-class-action-93-million-privacy-breach-1.4321257

[7] (2020, April) Secure file transfer — whisply. [Online]. Available: https://whisp.ly/en

[8] (2020, April) Cryptomator: Free cloud encryption for dropbox and others. [Online]. Available: https://cryptomator.org/

[9] (2020, April) Whitepapers from spideroak. [Online]. Available: https://spideroak.com/whitepapers/

[10] W. Ma, J. Campbell, D. Tran, and D. Kleeman, "Password entropy and password quality," in Fourth International Conference on Network and System Security, NSS 2010, Melbourne, Victoria, Australia, September 1-3, 2010, Y. Xiang, P. Samarati, J. Hu, W. Zhou, and A. Sadeghi, Eds. IEEE Computer Society, 2010, pp. 583–587. [Online]. Available: https://doi.org/10.1109/NSS.2010.18

[11] (2020, April) Aws sdk support for amazon s3 client-side encryption. [Online]. Available: https://docs.aws.amazon.com/general/latest/gr/aws_sdk_cryptography.html

[12] (2020, April) Cloud storage security - secure cloud storage from tresorit. [Online]. Available: https://tresorit.com/security

[13] (2020, April) Mega - secure cloud storage and communication. [Online]. Available: https://mega.nz/

[14] E. Bocchi, I. Drago, and M. Mellia, "Personal cloud storage: Usage, performance and impact of terminals," in 4th IEEE International Conference on Cloud Networking, CloudNet 2015, Niagara Falls, ON, Canada, October 5-7, 2015. IEEE, 2015, pp. 106–111. [Online]. Available: https://doi.org/10.1109/CloudNet.2015.7335291

[15] "Web cryptography api," the Web Cryptography WG of the W3C, Tech. Rep., January 2017. [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/

[16] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with webassembly," in ACM SIGPLAN Notices, vol. 52, no. 6. ACM, 2017, pp. 185–200.

[17] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in International Workshop on Public Key Cryptography. Springer, 2011, pp. 53–70.

[18] W. Zhu, J. Yu, T. Wang, P. Zhang, and W. Xie, "Efficient attribute-based encryption from r-lwe," Chin. J. Electron, vol. 23, no. 4, pp. 778–782, 2014.

[19] M. Green, S. Hohenberger, B. Waters et al., "Outsourcing the decryption of abe ciphertexts." in USENIX Security Symposium, vol. 2011, no. 3, 2011.

[20] S. Hohenberger and B. Waters, "Online/offline attribute-based encryption," in International Workshop on Public Key Cryptography. Springer, 2014, pp. 293–310.

[21] R. Zhang, H. Ma, and Y. Lu, "Fine-grained access control system based on fully outsourced attribute-based encryption," Journal of Systems and Software, vol. 125, pp. 344–353, 2017.

[22] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in Proceedings of the 5th ACM symposium on information, computer and communications security, 2010, pp. 261–270.

[23] (2020, April) owncloud - the leading opensource cloud collaboration platform. [Online]. Available: https://owncloud.org/

[24] (2020, April) Openpgp implementation for javascript. [Online]. Available: https://github.com/openpgpjs/openpgpjs

[25] E. Stark, M. Hamburg, and D. Boneh, "Symmetric cryptography in javascript," in Computer Security Applications Conference, 2009. ACSAC'09. Annual. IEEE, 2009, pp. 373–381.

[26] Y. Yuan, C.-M. Cheng, S. Kiyomoto, Y. Miyake, and T. Takagi, "Portable implementation of lattice-based cryptography using javascript," International journal of networking and computing, vol. 6, no. 2, pp. 309–327, 2016.
[27] H. Halpin, "The w3c web cryptography api: motivation and overview," in Proceedings of the 23rd International Conference on World Wide Web. ACM, 2014, pp. 959–964.
[28] A. Rangathan, "Web cryptography use-cases. working draft, w3c, 2013."
[29] Z. Guan, Z. Cao, X. Zhao, R. Chen, Z. Chen, and X. Nan, "Webibc: Identity based cryptography for client side security in web applications," in Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on. IEEE, 2008, pp. 689–696.
[30] W. He, D. Akhawe, S. Jain, E. Shi, and D. Song, "Shadowcrypt: Encrypted web applications for everyone," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 1028–1039.
[31] N. Attrapadung, G. Hanaoka, S. Mitsunari, Y. Sakai, K. Shimizu, and T. Teruya, "Efficient two-level homomorphic encryption in prime-order bilinear groups and a fast implementation in webassembly," in Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 2018, pp. 685–697.
[32] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2005, pp. 457–473.
[33] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in Proceedings of the 13th ACM conference on Computer and communications security. Acm, 2006, pp. 89–98.
[34] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in Proceedings of the 14th ACM conference on Computer and communications security. ACM, 2007, pp. 195–203.
[35] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2010, pp. 62–91.
[36] T. Okamoto and K. Takashima, "Fully secure functional encryption with general relations from the decisional linear assumption," in Annual Cryptology Conference. Springer, 2010, pp. 191–208.
[37] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in 2007 IEEE symposium on security and privacy (SP'07). IEEE, 2007, pp. 321–334.
[38] L. Cheung and C. Newport, "Provably secure ciphertext policy abe," in Proceedings of the 14th ACM conference on Computer and communications security, 2007, pp. 456–465.
[39] (2020, April) Indexeddb api. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
[40] A. Bhargav-Spantzel, A. C. Squicciarini, S. K. Modi, M. Young, E. Bertino, and S. J. Elliott, "Privacy preserving multi-factor authentication with biometrics," Journal of Computer Security, vol. 15, no. 5, pp. 529–560, 2007. [Online]. Available: http://content.iospress.com/articles/journal-of-computer-security/jcs292
[41] A. P. Sabzevar and A. Stavrou, "Universal multi-factor authentication using graphical passwords," in 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems. IEEE, 2008, pp. 625–632.
[42] X. Huang, Y. Xiang, E. Bertino, J. Zhou, and L. Xu, "Robust multi-factor authentication for fragile communications," IEEE Transactions on Dependable and Secure Computing, vol. 11, no. 6, pp. 568–581, 2014.
[43] R. Zhang, Y. Xiao, S. Sun, and H. Ma, "Efficient multi-factor authenticated key exchange scheme for mobile communications," IEEE Transactions on Dependable and Secure Computing, 2017.
[44] Y. Sutcu, Q. Li, and N. Memon, "Design and analysis of fuzzy extractors for faces," Proceedings of SPIE - The International Society for Optical Engineering, vol. 7306, 05 2009.
[45] J. Li, C. Jia, J. Li, and X. Chen, "Outsourcing encryption of attribute-based encryption with mapreduce," in International Conference on Information and Communications Security. Springer, 2012, pp. 191–201.
[46] T. Dierks and E. Rescorla, "Rfc 5246," The transport layer security (TLS) protocol version, vol. 1, 2008.
[47] A. Barth, C. Jackson, and J. C. Mitchell, "Securing frame communication in browsers," Communications of the ACM, vol. 52, no. 6, pp. 83–91, 2009.
[48] A. Barth, "The web origin concept," Tech. Rep., 2011.
[49] M. Shigeo. (2020, April) A portable and fast pairing-based cryptography library. [Online]. Available: https://github.com/herumi/mcl
[50] P. A. Grassi, J. L. Fenton, E. Newton, R. Perlner, A. Regenscheid, W. Burr, J. Richer, N. Lefkovitz, J. Danker, Y.-Y. Choong et al., "Nist special publication 800-63b: Digital identity guidelines," Enrollment and Identity Proofing Requirements. url: https://pages. nist. gov/800-63-3/sp800-63a. html, 2017.
[51] A. Beimel, "Secure schemes for secret sharing and key distribution," Ph.D. dissertation, Technion-Israel Institute of technology, Faculty of computer science, 1996.
[52] H. Ma, R. Zhang, Z. Wan, Y. Lu, and S. Lin, "Verifiable and exculpable outsourced attribute-based encryption for access control in cloud computing," IEEE Transactions on Dependable and Secure Computing, vol. 14, no. 6, pp. 679–692, 2017.
[53] H. Ma, R. Zhang, S. Sun, Z. Song, and G. Tan, "Server-aided fine-grained access control mechanism with robust revocation in cloud computing," IEEE Transactions on Services Computing, pp. 1–1, 2019.
[54] A. Shamir, "How to share a secret," Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979.
[55] Y. Rouselakis and B. Waters, "Practical constructions and new proof methods for large universe attribute-based encryption," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013, pp. 463–474.
[56] (2020, April) jquery file upload plugin. [Online]. Available: https://github.com/blueimp/jQuery-File-Upload
[57] (2020, April) Sabre/dav: the most popular webdav framework for php. [Online]. Available: https://github.com/sabre-io/dav
[58] (2020, April) Electron: Build cross platform desktop apps with javascript, html, and css. [Online]. Available: https://www.electronjs.org/

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments. This work was partially supported by National Natural Science Foundation of China (Nos. 61772520, 61802392, 61972094, 61472416, 61632020), Key Research and Development Project of Zhejiang Province (Nos. 2017C01062, 2020C01078), Beijing Municipal Science & Technology Commission (Project Number. Z191100007119007, Z191100007119002). Shuzhou Sun and Hui Ma contributed equally to this paper and are labeled as co-frist authors. Corresponding author is Rui Zhang.

Shuzhou Sun received his B.E. degree in software engineering from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2015. He is currently a PhD student in information security with the the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. His research interests include applied cryptography and information security.

Hui Ma received his B.E. degree in information security from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2012. He received his Ph.D. degree in information security from the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, in 2017. Now he is with Institute of Information Engineering, Chinese Academy of Sciences as an associate professor. His research interest includes applied cryptography and the security of cloud computing.

Zishuai Song received his B.E. degree in information security from the Xidian University, China, in 2017. He is currently pursuing the Ph.D. degree in information security with the Institute of Information Engineering, Chinese Academy of Sciences. He is currently involved in the security mechanisms in cloud computing.

Rui Zhang received his B.E. degree from Tsinghua University, and M.S./PhD. degrees from the University of Tokyo, respectively. He was a JSPS research fellow before he joined AIST, Japan as a research scientist. Now he is with Institute of Information Engineering (IIE), Chinese Academy of Sciences as a research professor. His research interests include applied cryptography, network security and information theory.