POSTER: WESHARE: A Coercion-Resistant and Scalable Storage Cloud

Hoon Wei Lim, Loi Luu, Shruti Tople, Prateek Saxena
School of Computing, National University of Singapore
{hoonwei,loiluu,shruti90,prateeks}@comp.nus.edu.sg

***Abstract*—Several cloud providers encrypt user data as best practices to protect against internal data theft. However, recent incidents have shown that cloud providers can be forced or coerced by governments, legal authorities or overtly malicious external attackers to decrypt user data when necessary. We define a property called coercion-resistance that frees cloud providers from the liability to decrypting client data under such external coercion.**

We study the challenges in building a coercion-resistant storage cloud that gracefully scales with large number of users and massive data. We discuss why existing solutions fail to achieve the desirable properties of a scalable, coercion-resistant cloud. We propose a new cryptographic primitive that achieves near-ideal storage and computation costs when scaled to millions of users sharing large datasets. Using these techniques, we build a cloud storage system called WESHARE that securely handles cryptographic access control and efficient ciphertext re-encryption under the assumptions of the coercion-resistance property. Our prototype integrates seamlessly with a commercial cloud service (Box) and with existing key directory services. We demonstrate that WESHARE achieves strong scalability as data sizes or users increase.

I. INTRODUCTION

Coercion-resistance is a desirable property for cloud provider as it frees them from being liable to decrypting content under legal subpoena or other forms of coercion from external agencies. In order to be coercion-resistant, cloud storage providers have recently resorted to encrypting all the client-side data before saving it on cloud [1], where the cloud does not participate in any private key generation or key storage, leaving key management entirely to users.

One of the important unaddressed challenges in present techniques is supporting encrypted file sharing at a *large scale*, in presence of a coercion-resistant cloud server. Here, by scale we mean three things—the number of users, the number of files (or distinct keys), and the size of ciphertexts. The demand for such scalable and dynamically controlled sharing is imminent. In enterprise email and other social applications, users often share sensitive text and attachments with a large number of users, mailing lists or groups.

To this end, we present our solution WESHARE that guarantees a coercion-resistant cloud-based file sharing system which is scalable with a practically acceptable performance.

II. PROBLEM & CHALLENGES

A. Problem definition

We consider a scenario where a file owner encrypts and stores her files on a remote platform operated by a cloud storage service provider who is assumed to be honest initially but can get coerced later at some point t_c. Moreover, the file owner may share some of her encrypted files with a group of intended recipients. Figure 1 describes our problem setting.

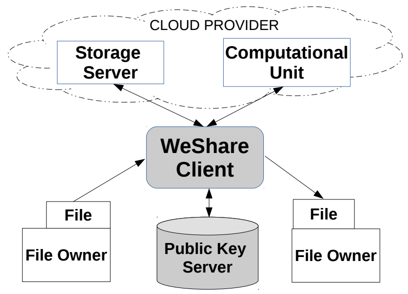

Fig. 1: WESHARE system model. The light shaded entities are publicly verifiable and trusted components. The cloud provider (indicated by dotted line) is untrusted.

Security Properties

- P1–Confidentiality & integrity: The confidentiality and integrity of any encrypted content should be protected against any party who is not an intended recipient, including the cloud provider before the coercion-point.
- P2–Coercion-resistance: The file key should be known only to the file owner and the intended recipients. The cloud provider should not be able to decrypt any content until the coercion-point t_c occurs. Figure 2 explains this property in our system.

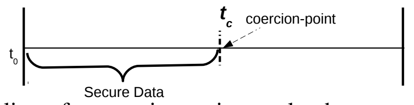

Fig. 2: Time line of a coercion-resistant cloud storage provider. All the data stored on the cloud is secure until time t_c when the provider gets coerced.

Efficiency Properties

- P3–Minimal upload cost
- P4–Minimal download cost
- P5–Minimal key revocation/refresh cost
- P6–Minimal key storage

Definition 1 (Ideal EFS): We regard an encrypted file system (EFS) to be ideal if it satisfies all properties **P1–P6**.

B. Existing Solutions & Challenges

Basic Approach. The majority of current commercial tools rely on a simple combination of symmetric and asymmetric encryption techniques (henceforth, we refer to this as the basic

	#. recipients	10					100					1000				
	File size	1KB	10KB	1MB	10MB	100MB	1KB	10KB	1MB	10MB	100MB	1KB	10KB	1MB	10MB	100MB
Basic approach	Time (ms)	1.6	1.8	16.5	146.6	1559.4	11.5	11.7	26.4	156.5	1569.3	102.6	102.8	117.5	247.6	1660.4
	Download / Upload size (KB)	2.3	11.2	1039.6	10283.8	102827.0	13.8	22.8	1051.1	10295.3	102838.5	129	138	1166.3	10410.5	102953.7
WeShare	Time(ms)	45.5														
	Download / Upload size (KB)	0.64														

TABLE I: Computational and upload costs during key refreshing on the client-side (for both basic scheme and WeShare); and the server-side (for WeShare only) with different file sizes and numbers of recipients. The computation time for sharing a file of size 100 MB to 1000 users is $36\times$ more for basic approach as compared to WeShare.

approach). That is, one encrypts a file with a symmetric file key k (e.g., AES) and then places an asymmetric encryption (e.g., RSA) of k, under the recipient's public key, in the file header. If the file is shared among n recipients, then k needs to be encrypted under the public keys of all the n recipients, which is undesirable particularly for large numbers of recipients. Moreover, to revoke any of the recipient, i.e., preventing the recipient from reading any future update to the file, the file owner essentially has to repeat the same steps as before.

A natural solution to circumvent the above problem is to use broadcast encryption (BE) [2], [3] which allows a ciphertext to be decryptable by multiple recipients. However, two major challenges remain.

Challenge 1: Removing Key Escrow. BE requires a trusted key authority possessing a master secret key to derive and distribute private keys to all system users.

Challenge 2: Re-encryption. We also note that re-encryption is costly (as costly as that of the basic approach) in existing file systems supporting cryptographic key revocation.

III. Our Approach

Using Broadcast Encryption without key escrow. One conceptually simple way to address the limitation of BE techniques is to let the user be the key issuer. We let the cloud pick a fixed public parameter set, and each client derives the required private BE keys without the server learning the derived keys.

Symmetric-Key Double Encryption technique. Intuitively, one could use a key homomorphic pseudo-random function (KH-PRF) [4] for file encryption, but it turns out to be computationally expensive.

We propose a symmetric-key double-encryption technique supporting efficient ciphertext update that guarantees security in our coercion-resistant model where cloud is assumed to be honest / benign until coercion-point. Note that, our double encryption scheme is not secure with a fully malicious cloud who is under coercion all the time. Each encryption comprises of two steps: (i) inner-layer encryption under a key known between the file owner and all recipients, and (ii) outer-layer encryption under a key shared between the file owner and the latest non-revoked recipient set. During revocation, the file owner provides key update material to the cloud, which in turn, performs re-encryption of the outer-layer of the encrypted file. All remaining non-revoked recipients have access to the keys required to perform double-decryption.

IV. Experimental Results

We have implemented WeShare to run transparently on top of a commercial service called Box. Users access the encrypted files without trusting any third-party server via our Chrome browser extension that asks for a small number of permissions, thereby being easily verifiable and isolated from the cloud provider's web page.

We benchmark WeShare with up to $10,000$ simulated users on Box and report the results in Table I and Table II. In comparison with the basic approach used by most commercial systems (as discussed in Section II-B), WeShare offers constant header size, as well as constant computational and storage overhead during file sharing and key revocation/refresh. Our advantage is more significant for large file sharing and/or large numbers of recipients. For example, when sharing a file with $1,000$ users, WeShare requires a file header which is $200\times$ smaller than the approach used by most commercial systems; on the other hand, the computational overhead for ciphertext update for a group size of $1,000$ users for sharing a 100 MB file is roughly $36\times$ less than that of the basic approach. The overall system latency matches our theoretical expectation. Our system, which is based on modifications to existing techniques, is secure.

	#. shared users	10	100	1,000	10,000
Basic approach	Time (ms)	1.63	11.52	102.63	1043.09
	Header size (KB)	1.2	12.8	128	1280
BoxCryptor	Header size (KB) [1]	7.4	74	740	7400
WeShare	Time (ms)	47.45	49.22	71.94	320.97
	Header size (KB)	0.64			

TABLE II: Computation time and size of ciphertext header in WeShare, the basic approach and BoxCryptor. The header size for basic approach is $200\times$ more than that of WeShare for 10,000 recipients.

V. Acknowledgement

This research is supported in part by the Ministry of Education, Singapore under Grant No. R-252-000-495-133 and in part by the National Research Foundation, Prime Minister's Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2014NCR-NCR001-21) and administered by the National Cybersecurity R&D Directorate.

References

[1] Boxcryptor. https://www.boxcryptor.com/.

[2] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Proc. of CRYPTO*, pages 258–275, 2005.

[3] A.B. Lewko, A. Sahai, and B. Waters. Revocation systems with very small private keys. In *Proc. of IEEE S&P*, pages 273–285, 2010.

[4] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In *Proc. of EUROCRYPT*, 1999.

[1] We only consider the file key encrypted with users' private key.