Access Control in Untrusted Cloud Storage using Unidirectional Re-encryption

Zach Kissel, Jie Wang
University of Massachusetts Lowell

The Cloud

Cloud storage makes many promises:

- Data can be accessed anywhere at any time
- No end-user cost for maintenance or infrastructure
- Platform independence

Cloud Security

Cloud storage is inherently insecure

- Data for different parties coexist on the same hardware, segregated by the service provider
- Data not necessarily stored in an encrypted form
- Must implicitly trust the service provider

Honest but Curious Model

- Assume that cloud is honest but curious
- Users of cloud storage should have complete control over whom they can trust to access their data
- Encryption is needed
  - Should only store one encrypted copy of the file
  - Sending keys directly to users would make it cumbersome to change keys

Current Methods

- Current cloud-based storage security revolves around heavy weight cryptographic primitives
- Attribute Based Encryption (ABE) is so far the most popular method, which provides fine grained access control over the data

ABE 10,000 Foot View

- Encryption primitive devised by Bethencourt, Sahai, and Waters (2006)
- In ABE attributes are arranged into a Boolian formula. When this formula is satisfied, decryption can occur
- Formula satisfaction is part of the cryptography

Inefficient Bilinear Pairings

- All known implementations of ABE use bilinear pairings:
  - $e(g^a, g^b) = g^{ab}$, for unknown a, b
  - Can be done over bilinear groups in time of a high order polynomial
  - Computationally inefficient

Other Methods

- mediated cryptography
  - using a mediated server
- proxy re-encryption

Proxy Re-Encryption

- A primitive that allows messages encrypted with Alice's public key to be transformed to messages under Bob's public key without Bob knowing Alice's private key
  - The name of the primitive derives from the fact that in the above scenario, Bob can serve as a proxy for Alice
  - Traditionally, proxy will perform re-encryption that takes as input the encrypted message and re-encryption key

Re-Encryption

- We modify the proxy re-encryption primitive to make it applicable in our system:
  - We do not use proxy to perform re-encryption; instead, Bob, in our scenario, will run the re-encryption algorithm himself
  - The re-encryption keys are stored in the cloud that are publicly accessible

Heavy Use of Proxy

- Mediated cryptography typically uses a form of secret sharing for the key between the user and proxy
- Both use proxy to enforce access control
- Proxy becomes single point of failure for all operations

Our Views

Why do you care what other people think?

- Current methods are inefficient and overly complicated
- We'd like a simple, efficient, and secure scheme

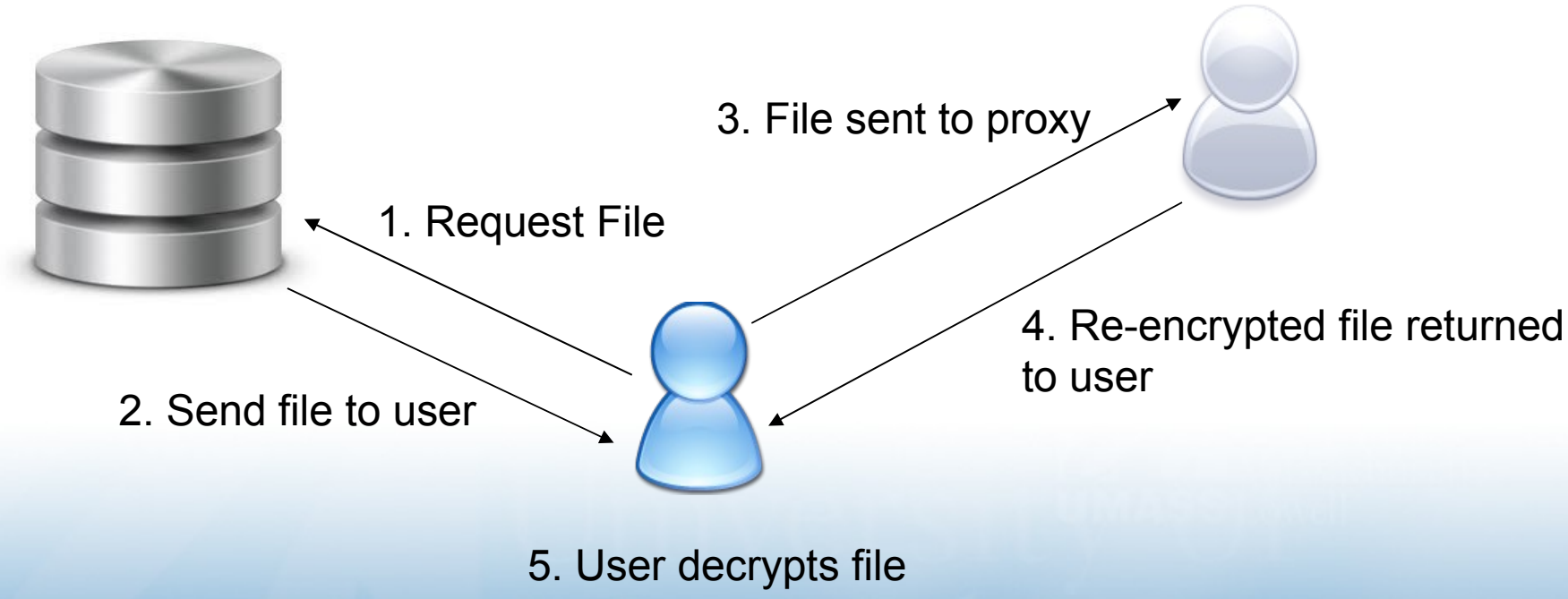
Richard Feynman

An Early Result

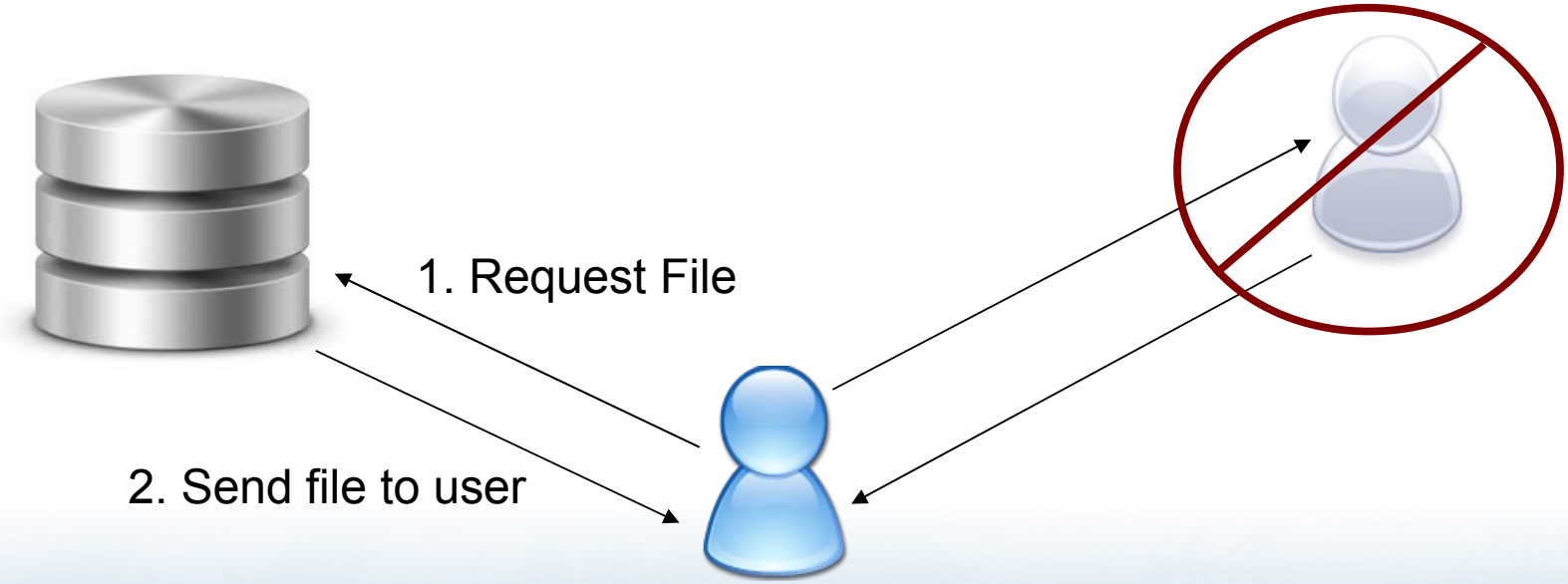
- Based on "Improved Proxy Re-Encryption with Applications to Secure Distributed Storage" (Atienese, Fu, Green and Hohenberger 2005)
- The paper presented a system that uses a collusion free unidirectional proxy re-encryption (UPRE) to secure distributed storage
  - Their UPRE scheme uses bilinear pairings
  - They use a proxy to do the re-encryption

A High Level View

- Alice requests a file from storage, this file is encrypted with a symmetric key (the symmetric key is encrypted with a public key in the UPRE system)
- Alice forwards this file to the proxy. The proxy then re-encrypts a wrapped shared key that forms the header of the file. The file with the re-encrypted header is returned to Alice
- Alice can then decrypt the file



Goal One: Remove Proxy



Goal One Cont.

- We satisfy goal one by removing the proxy and having the user do their own re-encryption
- This requires that the PRE system be unidirectional and collusion free
  - Collusion free means that given a re-encryption key K, between users A and B, private keys SA and SB for users A and B respectively, there does not exist a function f(K, SB,I) that yields any information that allows the proxy and B to perform an operation one of them wouldn't be able to do on their own.

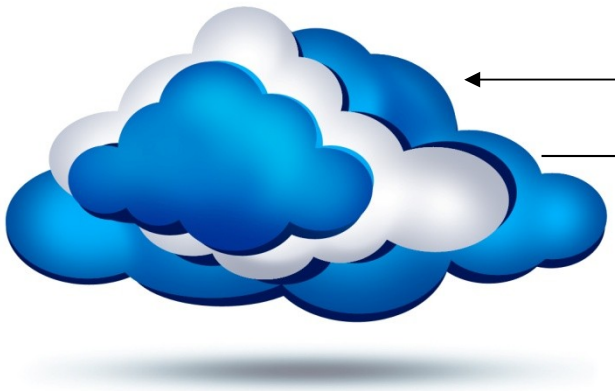
Goal Two: Remove Bilinear Pairing Operations

- We can remove the proxy and use Atienese, Fu, Greene and Hohenberger's PRE system as it's collusion free, but it uses undesirable bilinear pairings
- Another recent work: "Efficient Unidirectional Proxy Re-Encryption" by Chow, Weng, Yang and Deng
  - Pairing free unidirectional PRE scheme
  - But NOT Collusion free!

Goal Two Cont.

- We fixed the system to prevent the collusion attack
- We simplified the system via the removal of four hashes used in [CWYD]
- We showed this new scheme to be CPA secure in the IND-PRE-CPA game.

Final View in The Cloud



1. Request File

2. Send file to user

3. Users re-encrypts file

4. Users decrypts file

Secure Unidirectional Re-Encryption (SURE)

- We develop a Secure Unidirectional Re-Encryption (SURE) scheme (Details will be given later)
- **Theorem**. If Decisional Diffie-Hellman is secure, then SURE is secure in the IND-URE-CPA game
- SURE is Semantically Secure

Secure Cloud Storage over SUPRE

- Three major types of operations in our Secure Cloud Storage System (CSS)
  - Authentication
  - Group Operations – akin to POSIX (UNIX) access control groups
  - File Operations

Authentication

Uses the concept of tickets from Kerberos

Protocol 1. (Authentication Protocol)

(M1) $A \rightarrow C$: "I'm A"

(M2) $C \rightarrow A$: $\{K_{AC}, N_A\}_{P_A}, \{A, K_{AC}, \tau\}_{P_C}$

(M3) $A \rightarrow C$: $\{N_A + 1\}_{K_{AC}}, \{A, K_{AC}, \tau\}_{P_C}$

Creating Groups

Suppose Alice wants to create a group name n with a public-private key pair (P_n, s_n)

Protocol 2. (Group Create)

(M1) $A \rightarrow C : \{\text{GCREATE}, \mathsf{P}_n, n\}_{\mathsf{K_{AC}}}, \{A, \mathsf{K_{AC}}, \tau\}_{\mathsf{P_C}}$

(M2) $C \rightarrow S : A, \mathsf{P}_n, n, \{H(A, \mathsf{P}_n, n)\}_{\mathsf{s_C}}$

Add Users to A Group

Protocol 3. (Add User)

$$(\mathrm{M1})\quad A \rightarrow C : \{\mathrm{AUSER}, B, \mathrm{K}_{RE\text{-}B}, n\}_{\mathrm{K_{AC}}}, \{A, \mathrm{K_{AC}}, \tau\}_{\mathrm{P_C}}$$

$$(\mathrm{M2})\quad C \rightarrow S : A, B, \mathrm{K}_{RE\text{-}B}, n, \{H(A, B, \mathrm{K}_{RE\text{-}B}, n)\}_{\mathrm{s_C}}$$

where $\mathrm{K}_{RE\text{-}B}$ is the re-encryption key for Bob (using the group's private key and Bob's public key

File Operations

- To store a file F in the cloud for a group x of users, Alice generates a symmetric key K and uses it to encrypt F to get F'
- Alice retrieves from the cloud the certificate of the group she wants to share F with
- Alice verifies the certificate using Charlie's public key also stored in the cloud
- If verified, she uploads $(x, E(P_x, K))$ to the header of F'

File Operations cont.

- To retrieve a file, Bob of group x downloads the file with the appropriate header
- Looks at the group name and retrieves his re-encryption key for the group
- Run ReEncrypt on the encrypted K with the group's public key to generate the transformed ciphertext
- Use his private key to decrypt the transformed cipertext and retrieve K

SURE Components

- KeyGen – Generates a pair of public and private keys (encryption key)
- ReKeyGen – Generates a re-encryption key
- Encrypt – Encrypts a message with the encryption key
- ReEncrypt – Re-encrypts a ciphertext with the re-encryption key
- Decrypt – Decrypt the encrypted cipher text.

Parameters

- Prime p, q such that $q|p-1$
- $G = <g>$ and a subgroup of $(Z/pZ)^*$
- G has order q

KeyGen

- Alice selects $a \in Z/qZ$ at random
  - a: private (secret)
  - g^a: public
- Bob's key pair: (b, g^b)

Encrypt

- Alice encrypts a message $m \in G$:
  - Choose a random ephemeral key $t \in Z/qZ$
  - Compute the ciphertext:

$$(C_1, C_2) = (mg^t, (g^a)^t)$$

ReKeyGen

- Alice generates a re-encryption key for Bob (unidirectional) using Alice's private key a and Bob's public key g^b:
  - Choose at random h, y, v from Z/qZ
  - $K = h/a + y/a^2$
  - $V_B = (g^b)^v$, $W_B = g^v(h + y/a)$
  - Re-encryption key: (K, V_B, W_B)

ReEncrypt

- Bob re-encrypts ciphertext (C_1, C_2) as

$$(C_1, \ C_2{}^K) = (mg^t, ((g^a)^t)^K)$$

Decrypt

- In the case of original encryption, the input is (C_1, C_2), then

$$m = C_1 / C_2^{1/a}$$

- In the case of re-encryption, the input is (C_1, C_2^K), then

$$m = C_1 / C_2^L,$$

$$L = V^{1/b} / W$$

The IND-URE-CPA Game

1. *C* (the challenger) informs *A* (the adversary) the SUPRE parameters

1. *A* asks *C* to generate a public key or a public-private key pair; may do so for a fixed polynomial number of times

1. *A* selects two users *i* & *j* from the public key pool, encrypts a message using *i*'s public key, and asks *C* to re-encrypt it using *j*'s re-encryption key; may do so for a fixed polynomial number of times

IND-URE-CPA Game cont.

1. *A* generates messages m_0 & m_1 ($|m_0| = |m_1|$), selects users *i* whose private key is not known to *A*, and sends them to *C* with *i*'s public key. *C* flips a random coin $c \in \{0,1\}$, encrypts m_c using *i*'s public key, and sends it to *A*

1. *A* guesses $c' \in \{0,1\}$ (from information obtained from previous phases) and wins the game if $c' = c$

   A's advantage is defined to be $\mathbf{Pr}[c' = c] - ½$

Implementation

- We implemented SUPRE in a source group induced by the prime p = 2q – 1, where q is a prime, with GNU's GMP library
- 128-Bit AES was used as the symmetric cryptography system
- Signature system was implemented using RSA with SHA-1 Hashes
- All cryptographic operations were provided by OpenSSI's libcrypto

Settings

- The cloud was implemented as a web server on a Linux 2.6.35-28 (AMD dual core x64)
- The server and the client machines were placed in different cities
- Each test was performed 1,000 times, each on a 1 kilobyte file
- Public keys were about 5K bit long

Reading Test

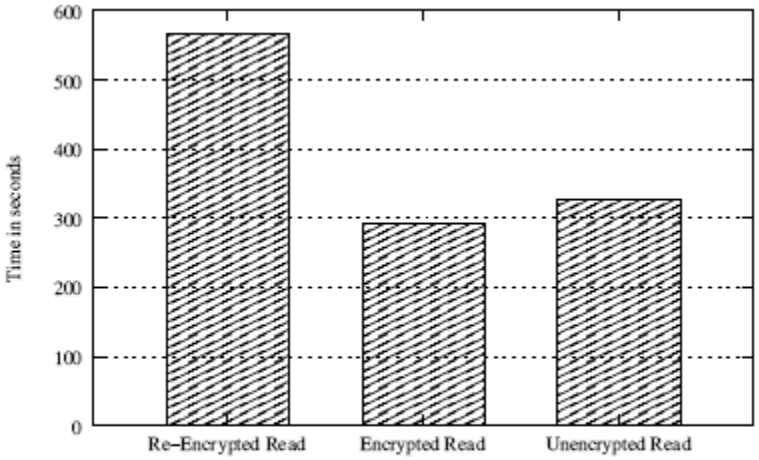


Fig. 1. Remote reading of 1000 1KB files

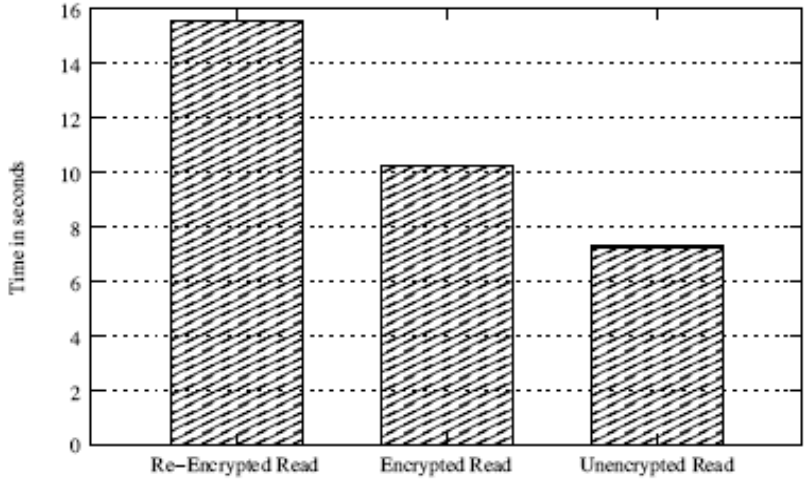


Fig. 2. Local reading of 1000 1KB files

Writing Test

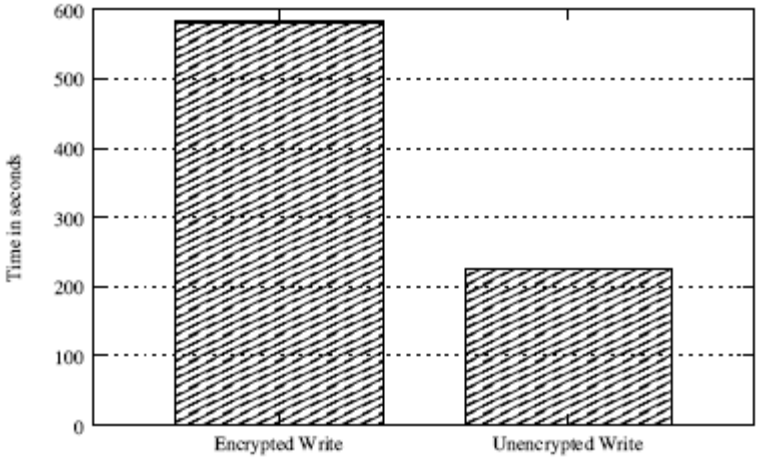


Fig. 3. Remote writing of 1000 1KB files

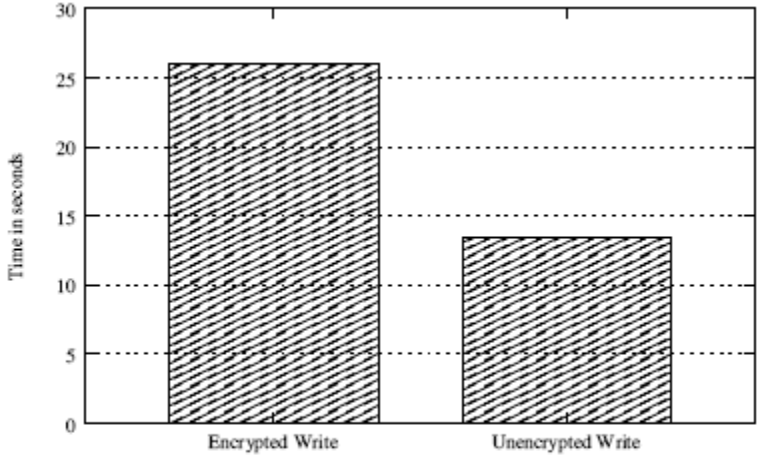


Fig. 4. Local writing of 1000 1KB files

Result Summary

TABLE I
AVERAGE READ AND WRITE TIMES, IN SECONDS, BASED ON 1000 OPERATIONS ON A 1KB FILE.

Operation	Average Time	Standard Deviation
Unencrypted Read File	0.326578	1.069543
Proxy Read File	0.567046	1.240770
Encrypted Read File	0.292369	0.909916
Unencrypted Write File	0.224174	0.947876
Write Encrypted File	0.582125	1.255204

Conclusion

- We presented a simple and efficient cloud storage protocol based on a secure unidirectional re-encryption scheme
- Our protocol removes bottlenecks common in other systems
- Garner some protection from the single-hop nature of the ciphertexts