DupLESS

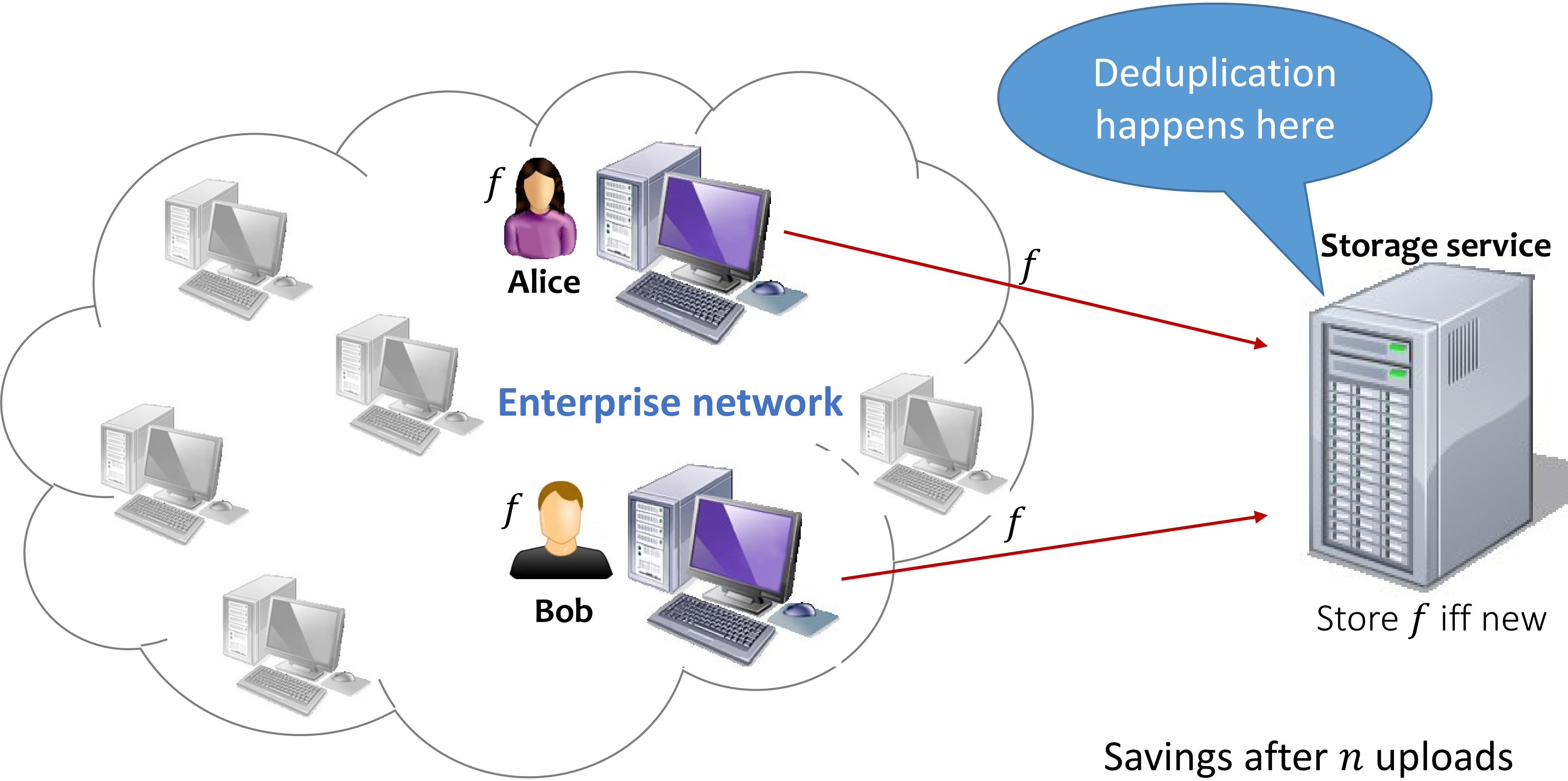
Server-Aided Encryption for Deduplicated Storage

Mihir Bellare[1] Sriram Keelveedhi[1] Thomas Ristenpart[2]

[1]University of California, San Diego [2]University of Wisconsin-Madison

Deduplication

Avoid storing multiple copies of the same data



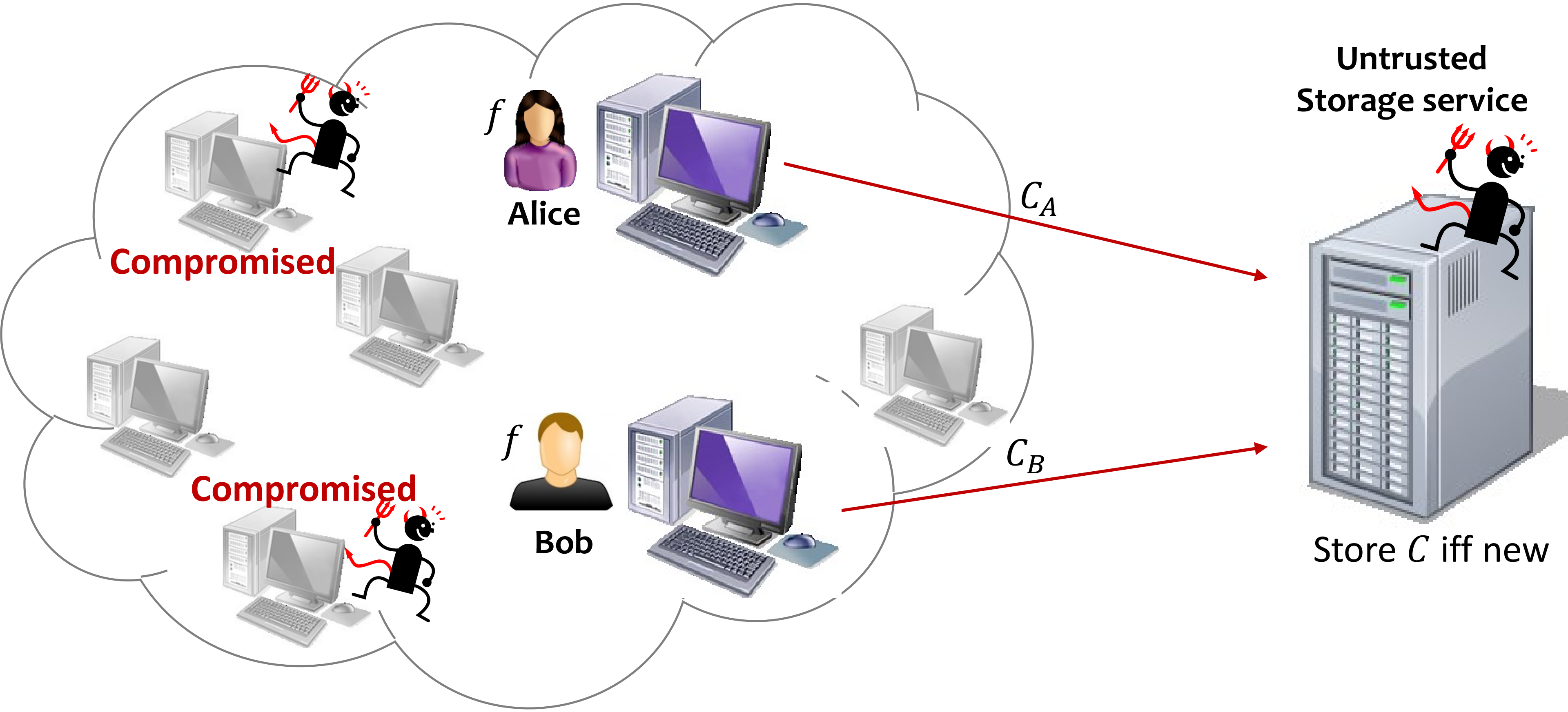
Used in outsourced storage services Dropbox Google Drive

Savings after n uploads

No dedup	$n \cdot$ size of f
Dedup	$1 \cdot$ size of f

Savings of 50% in enterprise networks [MB11]

Our goals



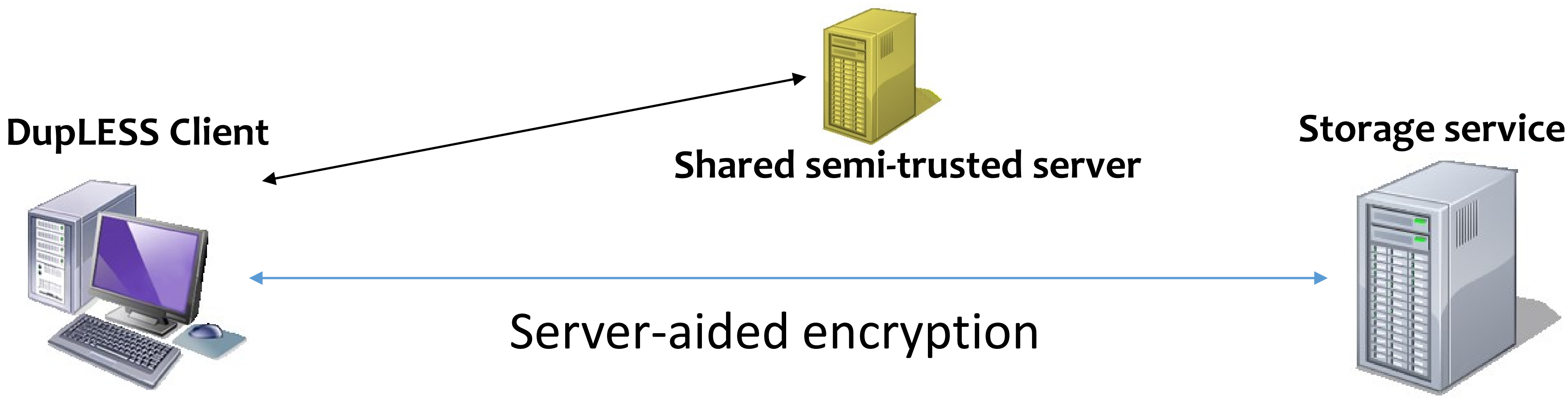
1. Secure deduplication: Dedup + Strong security against untrusted storage

2. Compromise resilience: Meaningful security under client compromise

Overview

DupLESS (DuplicateLess Encryption for simple storage)

First solution to achieve secure deduplication with compromise resilience

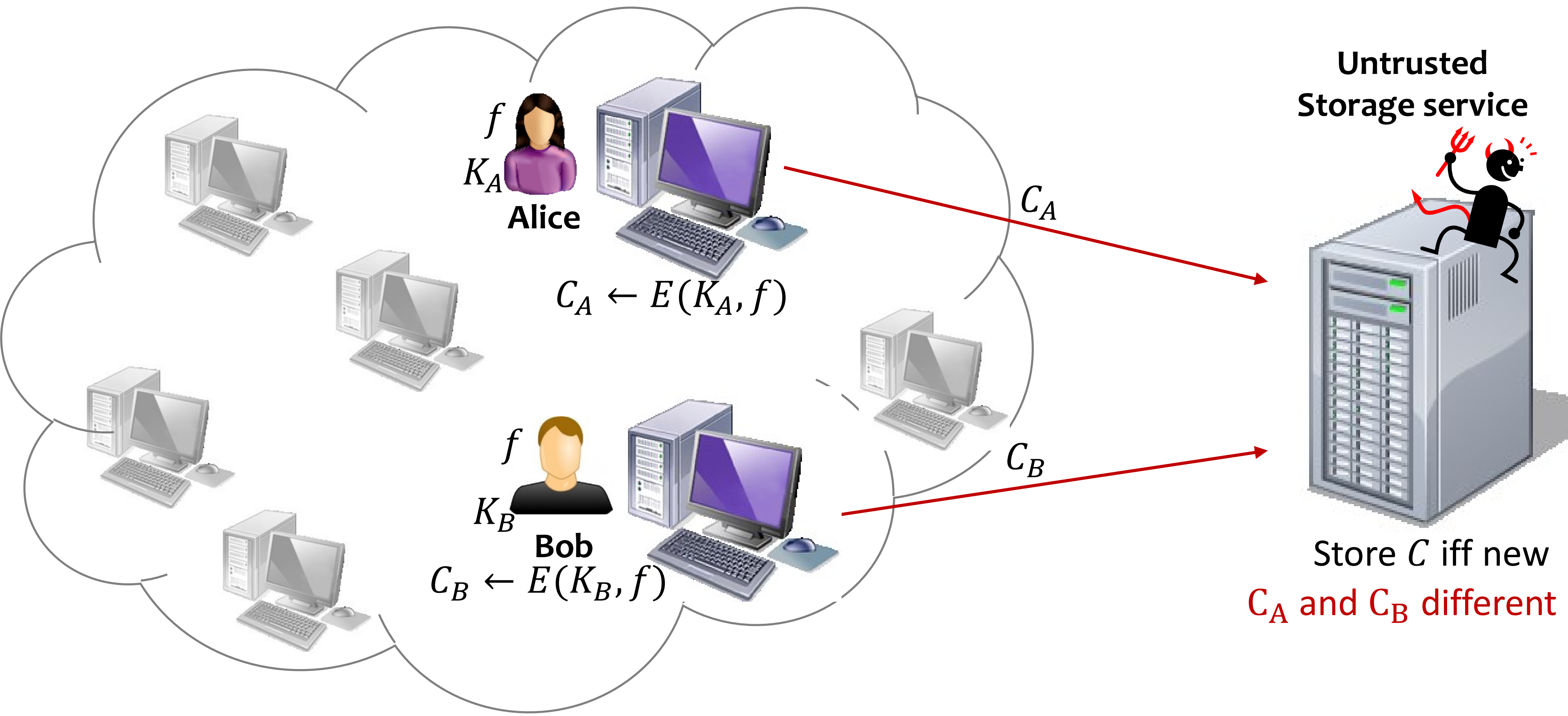


- Can be deployed **transparently** over existing systems
  - Implementations over Dropbox, Google Drive
- **Modest performance overhead** over plaintext dedup
- Storage savings match plaintext dedup

Current approaches

Attempt 1: Client specific keys

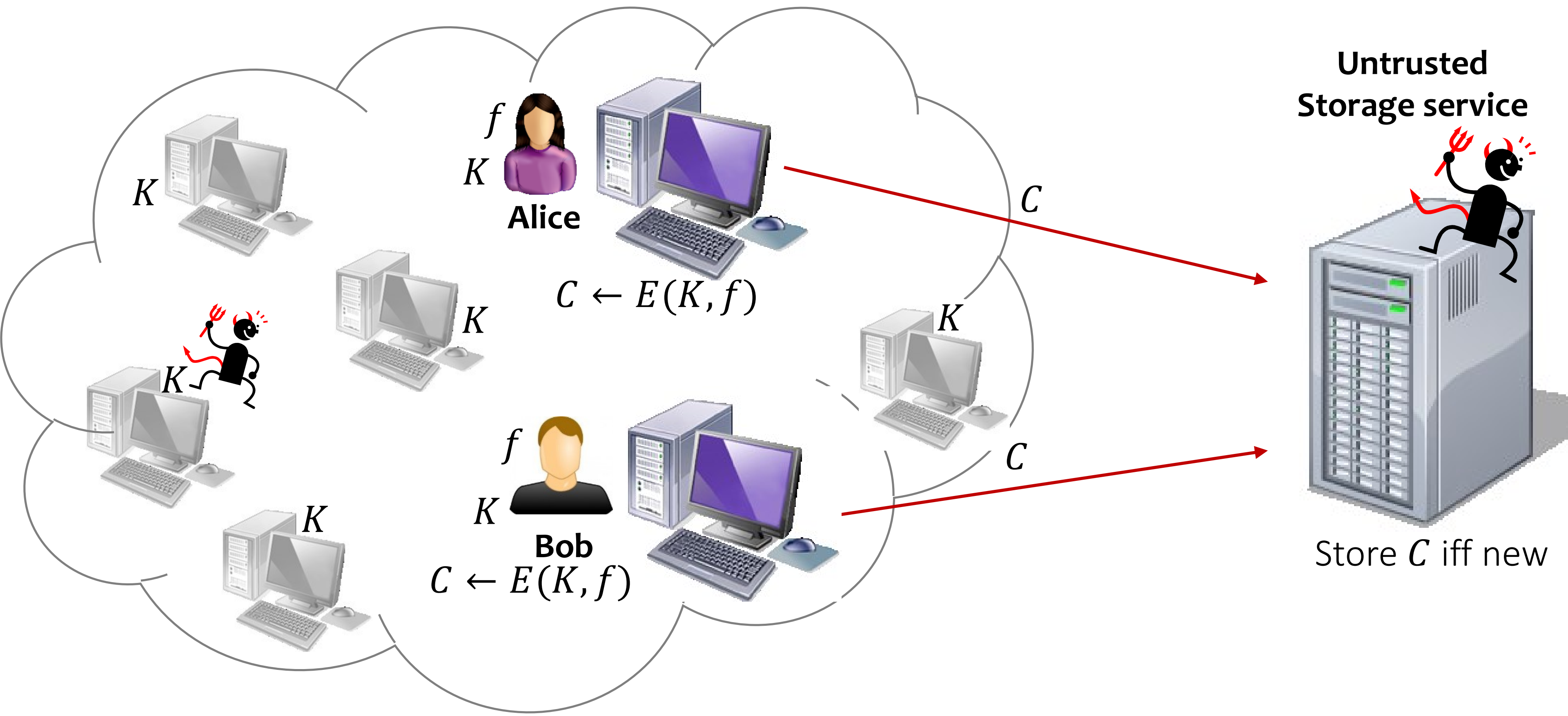
[GSMB03], [KRS*03], [KPR11]



Deduplication cannot work

Attempt 2: Network-wide key

[BBO07], [RS06]



No compromise resilience

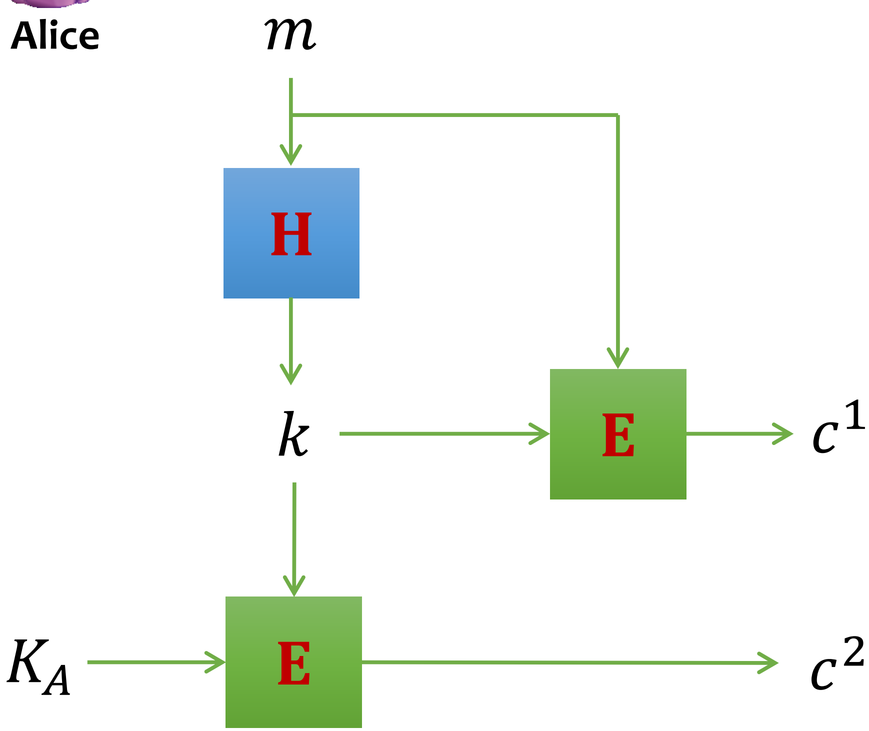
All data is insecure even if one client is compromised

Attempt 3: Convergent Encryption

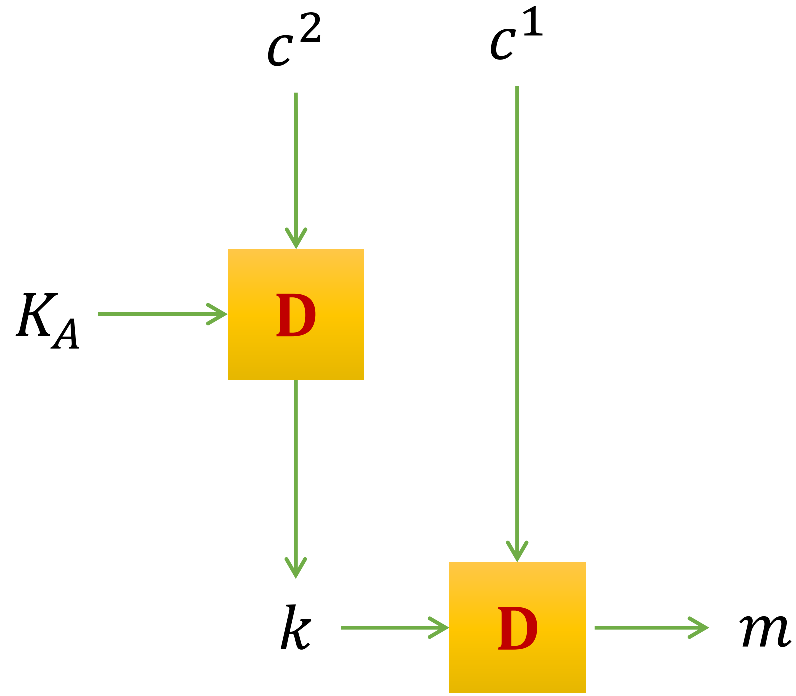
[DCS02]

Alice

To encrypt message m:



To decrypt ciphertext c^1, c^2

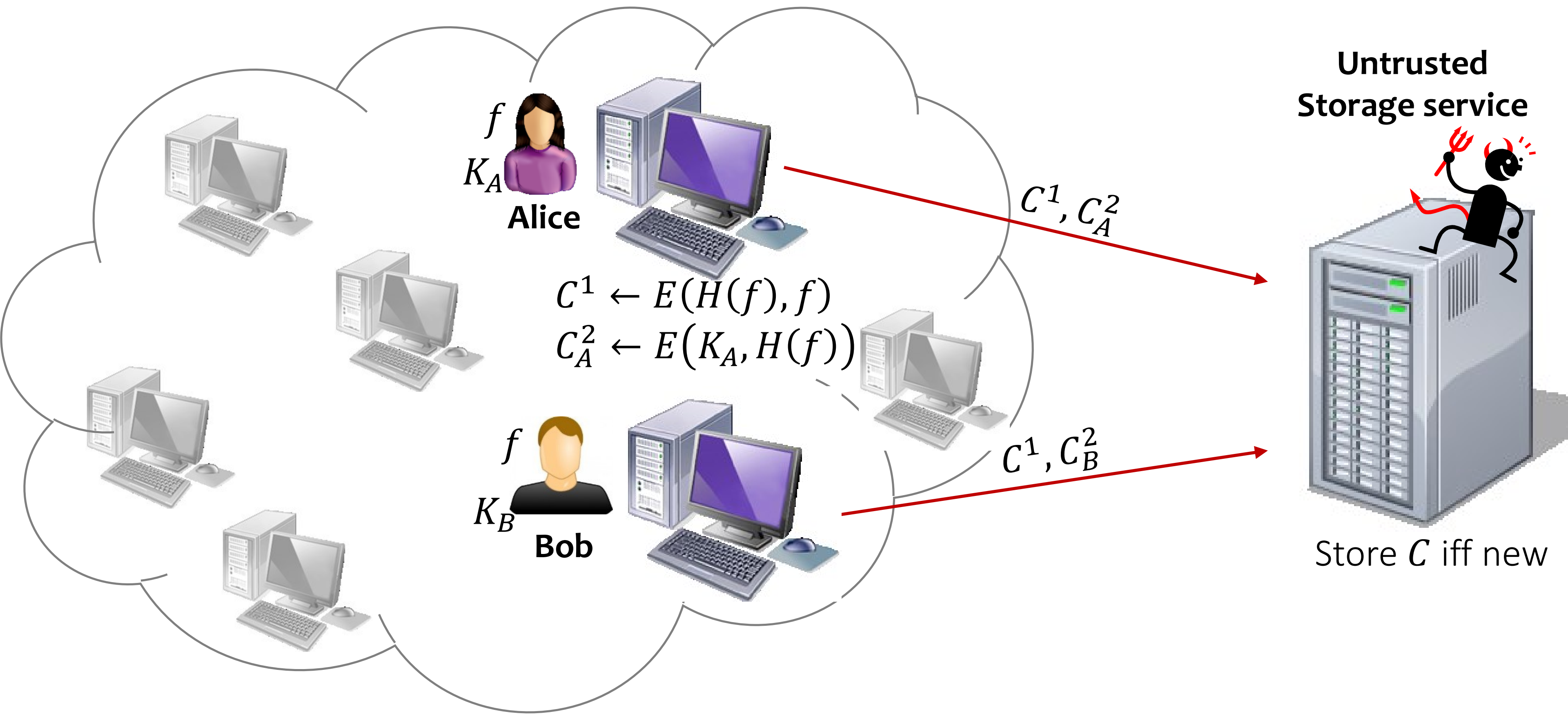


H: Hash fn. -------------------→ *SHA256*

$\mathcal{E} = (\mathrm{E}, \mathrm{D})$: Enc. scheme ----→ *CTR[AES128]*

Attempt 3: Convergent encryption

[ABC*02], [SGLM08],…



✓ **Deduplication**: Everyone encrypting f gets C^1

✓ **Compromise resilience**: No system-wide secret

Attempt 3: Convergent encryption

Brute force attacks: The dirty secret of convergent encryption

If m comes from $S = \{m_1, m_2, \ldots, m_n\}$
attacker can recover m from $c \leftarrow \mathrm{E}(\mathrm{H}(m), m)$

$\mathrm{BruteForce}_S(c)$

For $m_i \in S$ do
 $m' \leftarrow \mathrm{D}(\mathrm{H}(m_i), c)$
If $m_i = m'$ then return m_i

Attack runs in time proportional to $|S|$

Security only when $|S|$ too large to exhaust ← - - - - Unpredictable

Real files are often predictable!

Message-Locked encryption [BKR13]

- Generalizes convergent encryption
- Captures properties needed for secure deduplication

Thm: Brute-force attacks exist for all message-locked encryption schemes

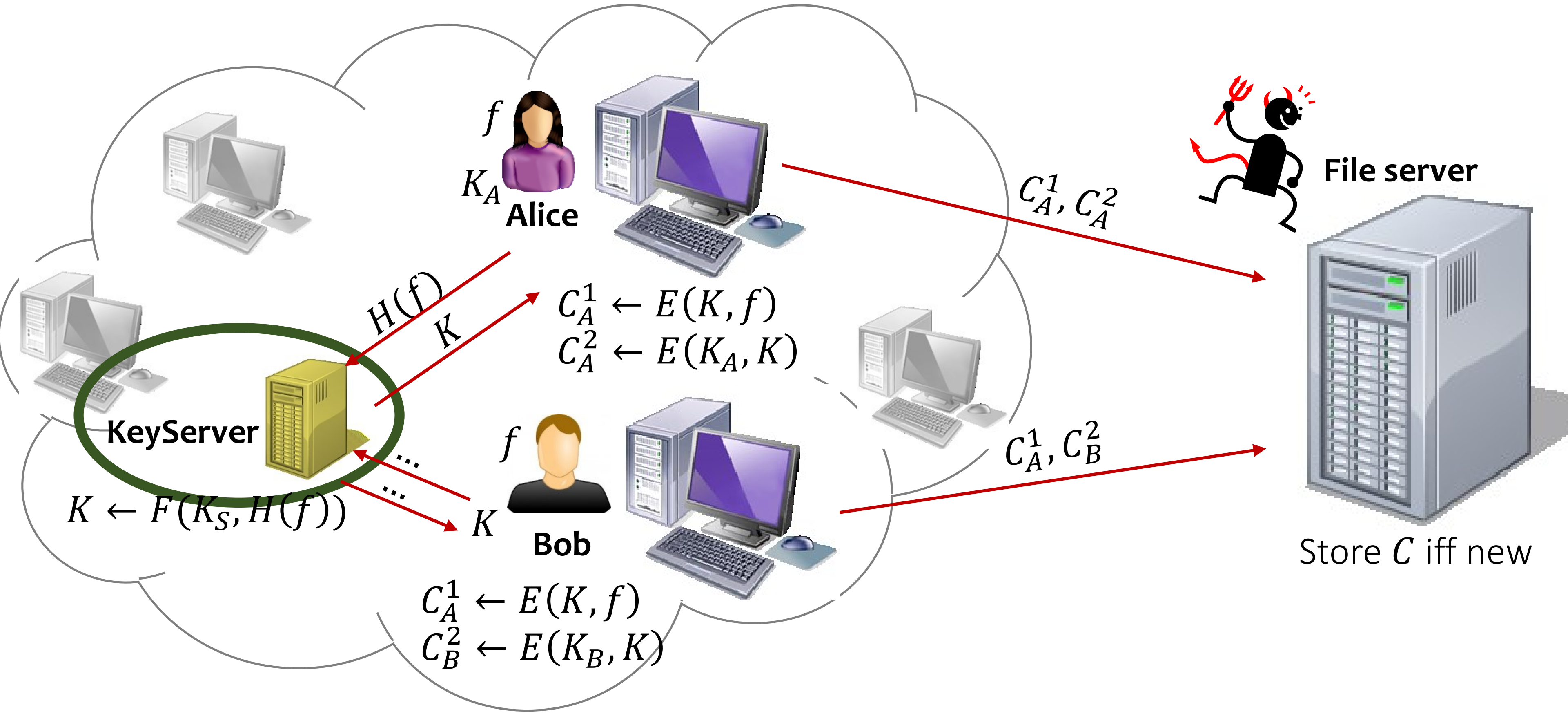
State of the art

	Systems			
Property	**Client specific keys**	**Network wide key**	**Convergent encryption**	**DupLESS**
Deduplication	N	Y	Y	Y
Compromise resilience	Y	N	Y	Y
Brute-force attack resilience	Y	Y	N	Y

DupLESS: First to achieve all three properties!

Server-aided encryption

Our key insight: Server-aided encryption



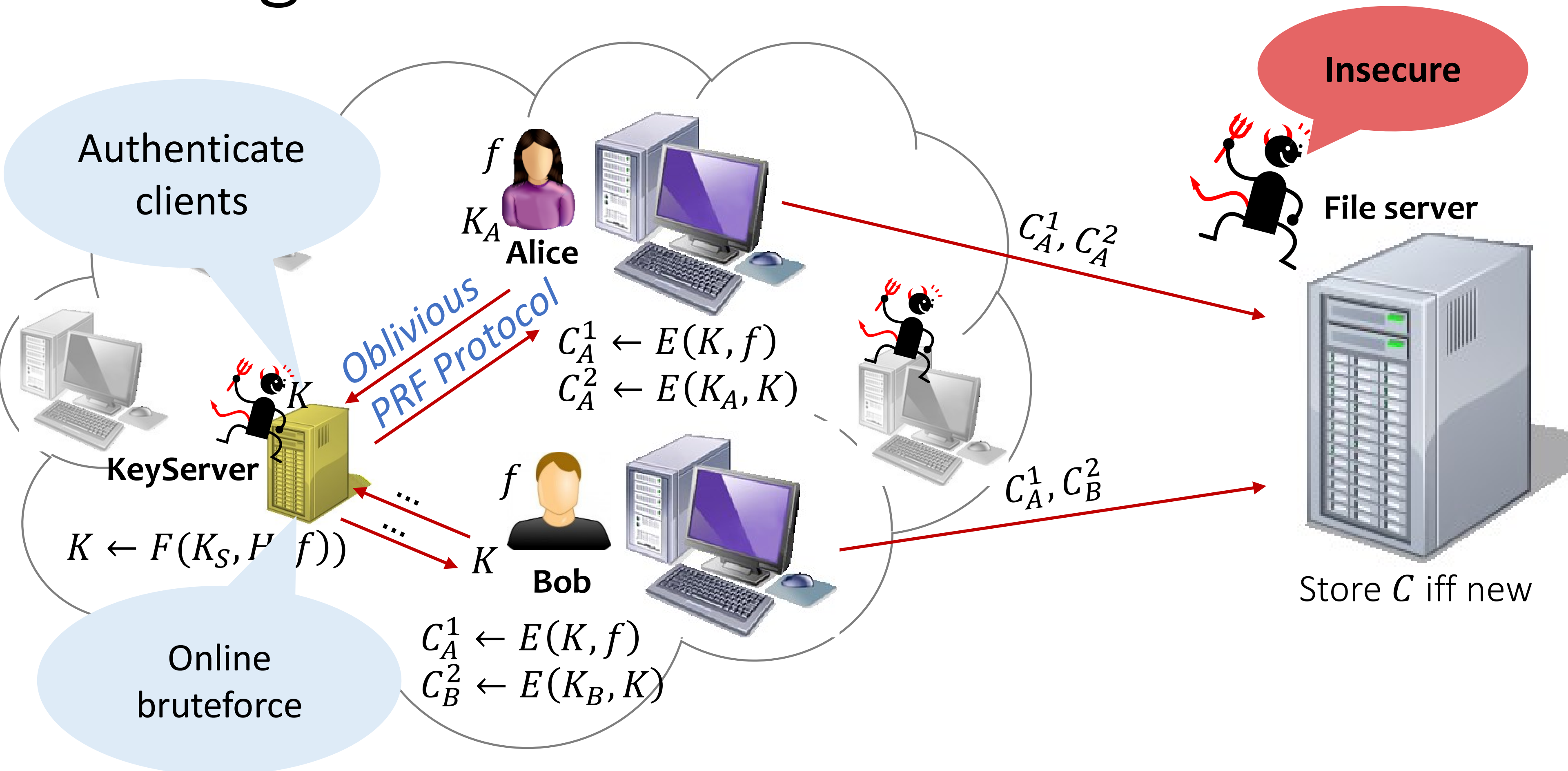
F: A pseudorandom function (PRF)

Examples: AES128, HMAC[SHA256]

Deduplication: Any client encrypting f produces same C^1

C^2 ciphertexts cannot be dedup'ed, but they are tiny

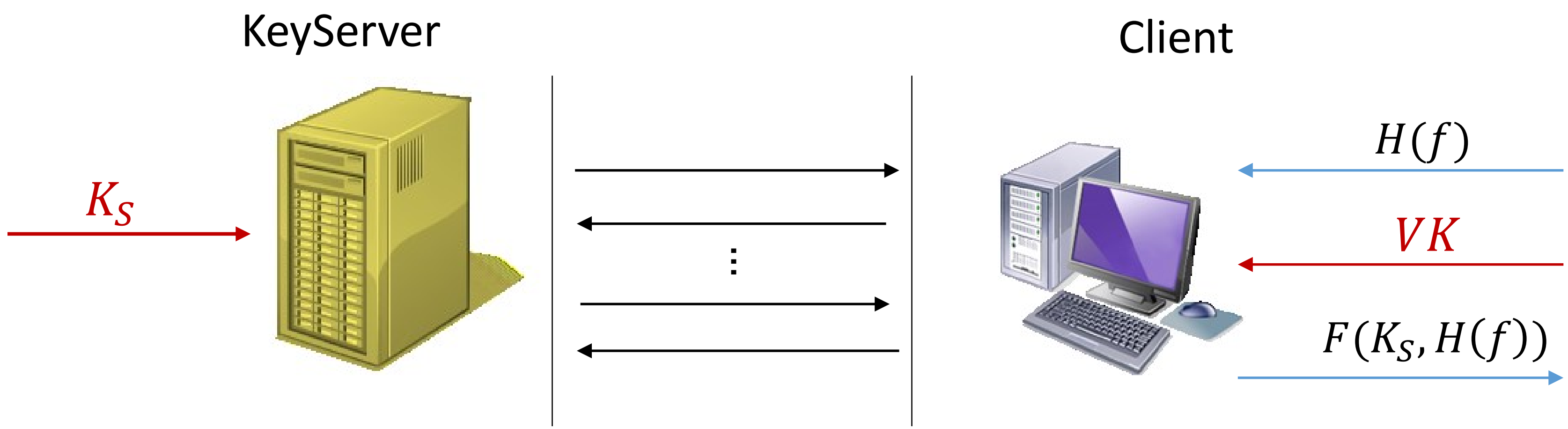
Dealing with attacks



Attack type	Reason for security	Best attack
External attacks	Authenticating clients	Break encryption **(very hard)**
Client compromise	KS interaction overhead	Online brute-force **(slow)**
KeyServer compromise	Obliviously evaluating F	Brute-force attacks

Oblivious PRF (OPRF) protocol [NR97]

F: A Pseudorandom function (PRF)



Verifiable OPRF: Client can verify $K = F(K_S, H(f))$

Security, informally:

1. F is a PRF (when not given VK)
2. Server learns nothing, client learns only K
3. Client can detect when server does not return K

Oblivious PRF protocol [NR97]

Securely evaluate AES circuit? **Too slow!**

Oblivious PRFs from unique blind signatures [CNS07, DeCSTW12]

Blind Signatures from RSA-FDH [C82, BNPS09]

Main idea	Server signs messages with RSA-FDH signatures Obliviousness through blinding

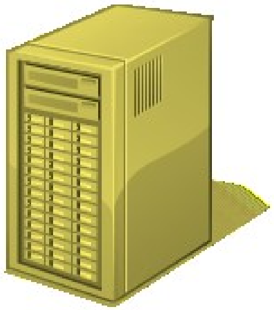
- Verifiable
- Single round
- KeyServer: 1 RSA exponentiation
- Client: 2 RSA exponentiations + 1 inverse

Client-KS protocol

Assume PKI with trusted CA

KeyServer

CERT_S, K_S



Standard protocol

TLS 2way auth handshake
+ OPRF query & response over secure channel

4 rounds for each query

Client

CERT_C, VK

M

Optimized protocol

Session initialization

TLS 2way auth handshake

Session key sent over secure channel

Making a query

Client sends OPRF input

KeyServer performs checks, returns OPRF output

Over UDP

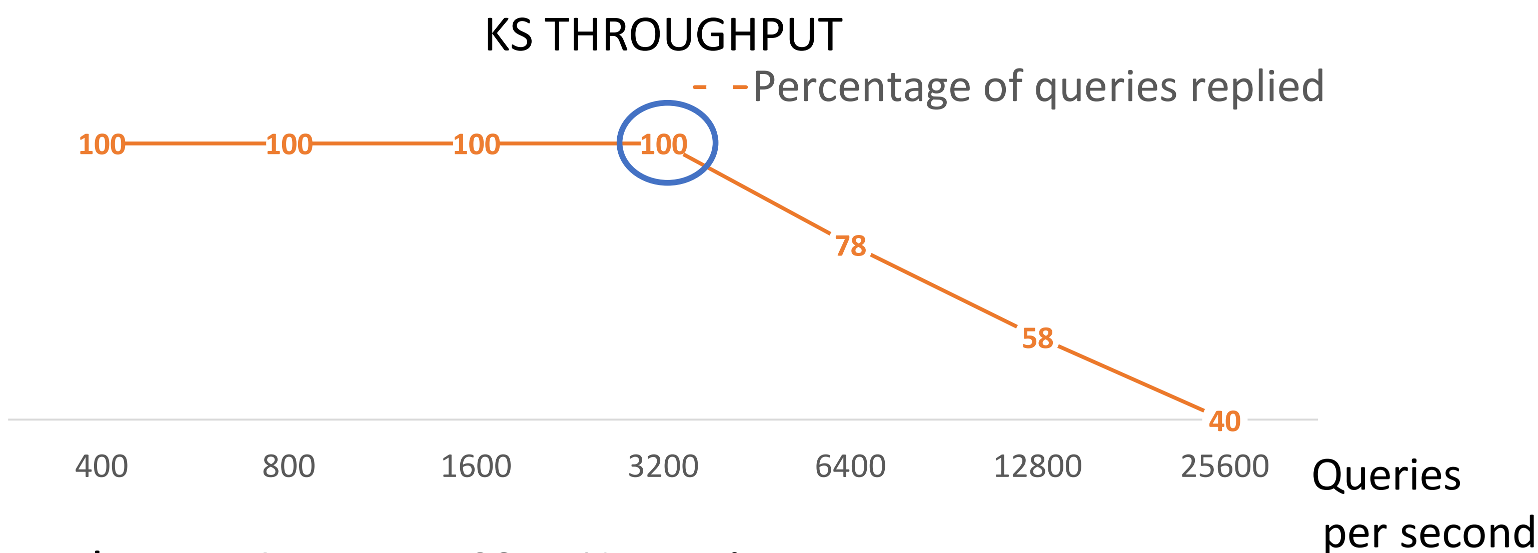
Preventing query forgery

Per session keys + sequence numbers + MAC

1 round for each query

KS performance

Naïve HTTPS based	384ms
Optimized	
Initialization	278 ms
Query response (Low load)	**83 ms**
Query response (Heavy load)	118 ms
Ping times	78 ms



KS located on an Amazon EC2 m1Large instance.
Heavy load $\approx 3k$ queries per second

Rate limiting

Goal: Slow down online brute-force trials
from attacker controlled clients

Strategy: Limit clients to q queries per epoch
One epoch lasts τ units of time

Setting bound q

Too low
Normal usage affected

Too high
Attacks not slowed down

Setting epoch duration τ

- Must handle bursty workloads
- Systems exhibit periodic patterns, Eg: 1 week

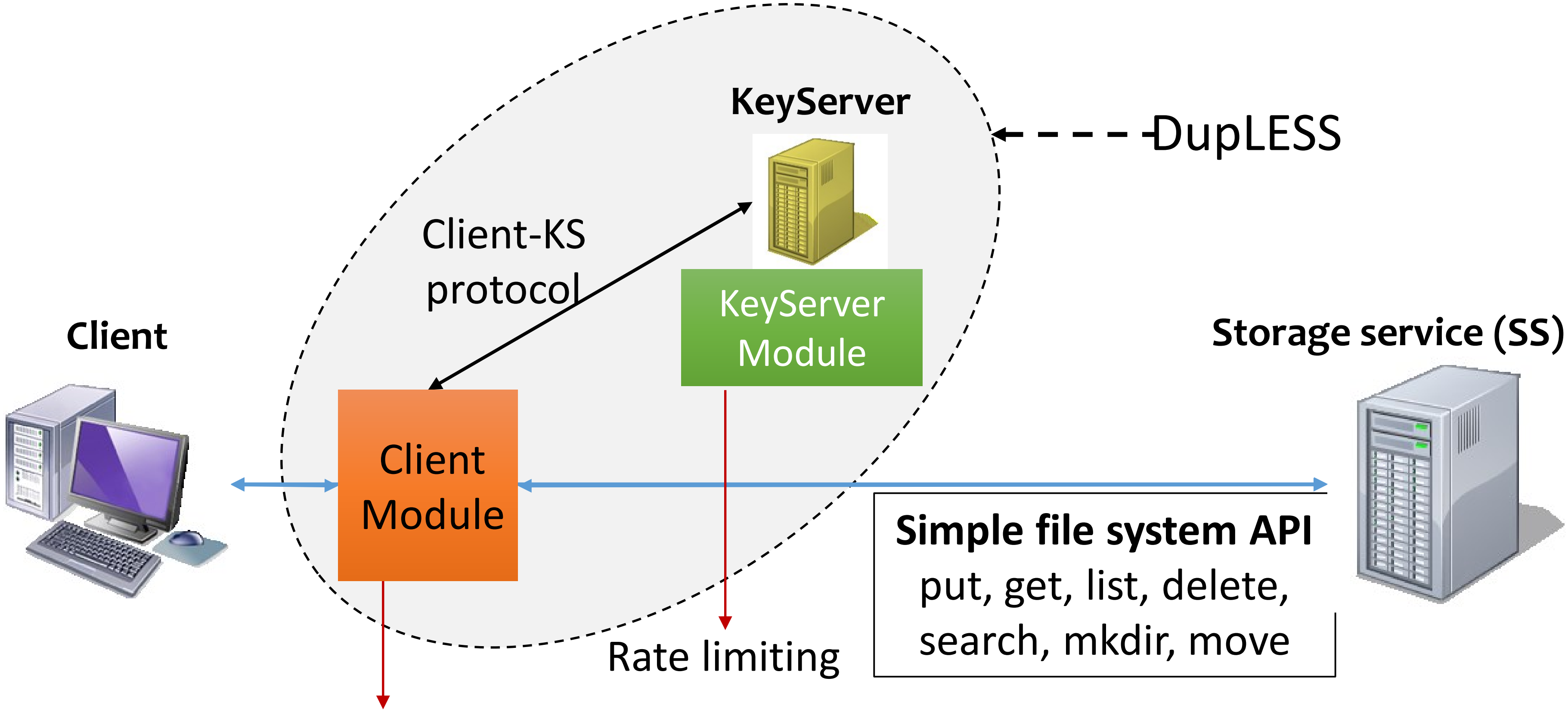
Randomized encryption when KS unavailable

- Availability not affected by bad parameter choices

Rate limiting can slow down brute-force attacks by 4000x

DupLESS system design

DupLESS (DuplicateLess Encryption for simple storage)



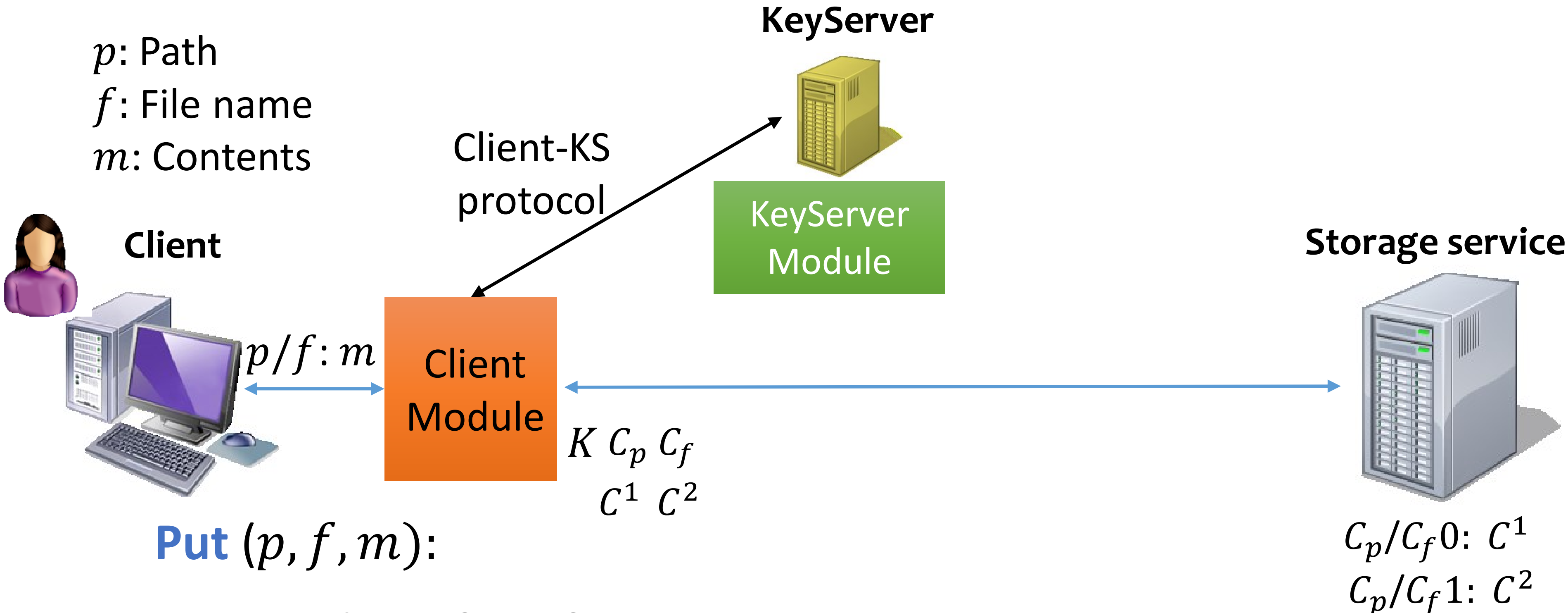
Implement API over encrypted data

Encrypt and decrypt files

Handle file names and paths

Run Transparently :
- Low overhead
- Works when KS is down
- No client-side state

A put query in DupLESS



Put (p, f, m):

1. Derive key K for m from KeyServer
2. $C_p \leftarrow \mathrm{DAE}(K_A, p)$; $C_f \leftarrow \mathrm{DAE}(K_A, f)$
3. If not shouldDedup(p, f, m) then pick K at random
4. $C^1 \leftarrow E(K, m)$; $C^2 \leftarrow E(K_A, K)$
5. PutSS (C_p, C_f0, C^1), PutSS (C_p, C_f1, C^2)

Deterministic authenticated encryption [RS07]

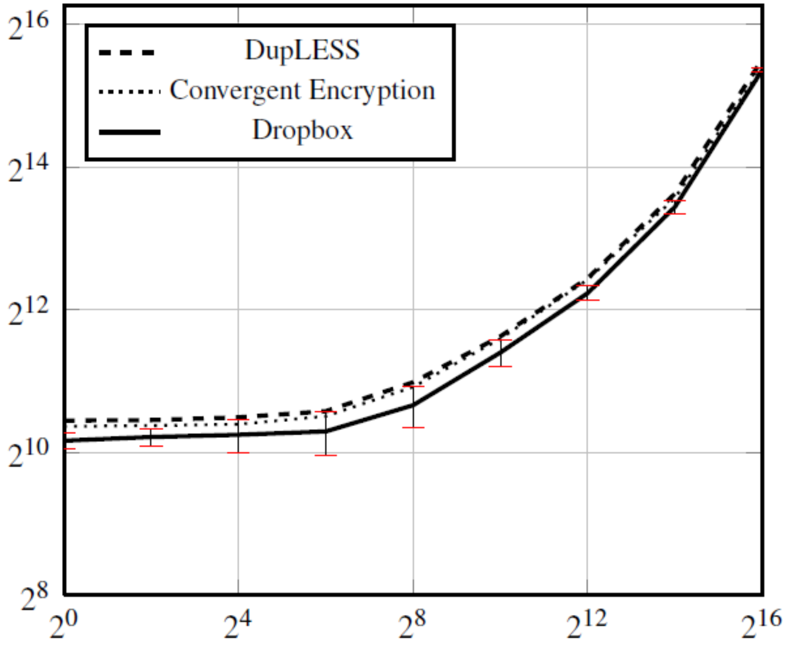
Dedup heuristics e.g. file length

Performance: Latency

DupLESS client

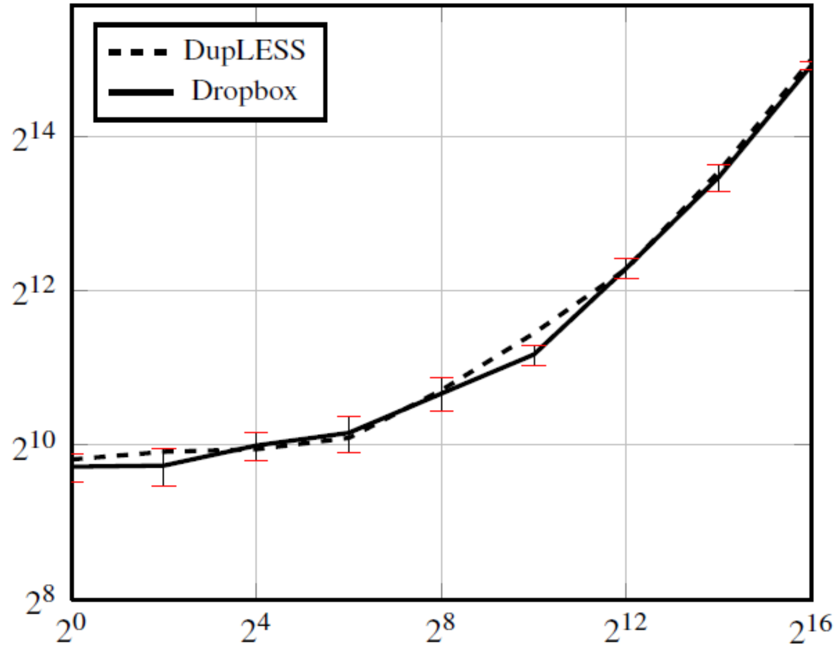
- Written in Python, command-line interface
- Dropbox and Google Drive can work as storage service

Put



File size	16KB	16MB
Overhead*	16%	14%

Get



File size	16KB	16MB
Overhead*	10%	5%

X-axis: File size (KB) Y-axis: Time (ms)

* Overhead of DupLESS over Dropbox

Bandwidth overhead

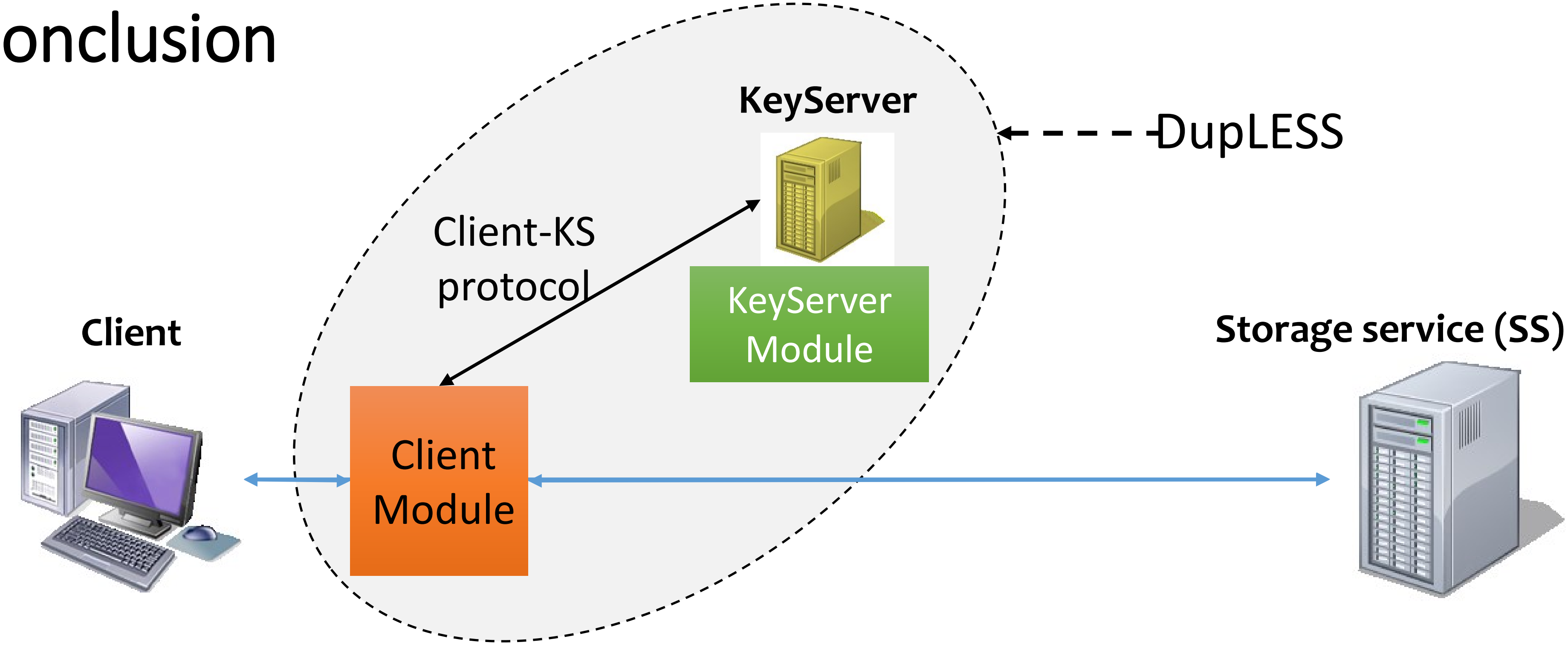
File size	16KB	16MB
DupLESS bandwidth overhead compared to plain Dropbox	16%	<1%

Storage overhead

DupLESS storage overhead compared to dedup over plaintexts	4.4%

Amazon AMI dataset, total size: 2035 GB

Conclusion



Encrypted deduplication with the aid of a KeyServer

- **First solution** to provide secure deduplication + compromise resilience
- Can be deployed **transparently** over existing systems
  - Implementations over Dropbox, Google Drive
- **Nominal performance overhead** over plaintext dedup
- Storage savings match plaintext dedup

Future work

- Supporting keyword search
- Defense in depth at the KeyServer
  - Combine DoS prevention and rate-limiting
- Support complex file-systems
  - NFS, CIFS, etc.
- Exploring dedup heuristics
  - Rules on which files to select for dedup

DupLESS

Server-Aided Encryption for Deduplicated Storage

Mihir Bellare[1] Sriram Keelveedhi[1] Thomas Ristenpart[2]

Thank you!

Paper available at
eprint.iacr.org/2013/429.pdf

Code available at
cseweb.ucsd.edu/users/skeelvee/dupless

[1]University of California, San Diego [2]University of Wisconsin-Madison