Secure Deduplication Across Files

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Master of Engineering
IN
Faculty of Engineering

BY

Nithin V Nath

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2016

Declaration of Originality

I, **Nithin V Nath**, with SR No. **04-04-00-10-41-14-1-11158** hereby declare that the material presented in the thesis titled

Secure Deduplication Across Files

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2014-2016**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discusions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: **Dr. Bhavana Kanukurthi** Advisor Signature

© Nithin V Nath
June, 2016
All rights reserved

DEDICATED TO

The Cryptography, Security and Privacy Group (CrySP)

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Dr. Bhavana Kanukurthi for the continuous support of my research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I would also like to thank my reader Dr. Sanjit Chatterjee for his insightful feedback.

I thank my fellow labmates in Cryptography, Security and Privacy Group for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had.

Last but not the least, I would like to thank my family: my parents and my sister for supporting me throughout writing this thesis and my life in general.

Abstract

With cloud computing advances, more and more data is being stored in cloud storage. Deduplication allows space savings for the cloud provider by storing only a single copy of repeating data. Achieving this in a secure, privacy preserving way opens up several challenges. Traditional file-level deduplication allows space savings only when the files are identical to each other. In many scenarios, users upload files that are close to each other. We discuss the construction of a secure deduplication scheme that enables deduplication not only for the files that are identical, but also for files that are close to one another under certain conditions[1]. We put forward a construction and proves its correctness and security.

[1]We divide the entire message space into codewords. Messages that come under the same codeword can be deduplicated easily

Contents

List of Figures

Chapter 1

Introduction

Rise of cheap cloud storage solutions has encouraged users as well as enterprises to resort to commercial cloud storage services such as Google Drive [2] and Dropbox [1]. This has in turn triggered a large influx of data to be stored in remote servers. In order to keep the prices competitive, service providers require space saving techniques that can be deployed on a large scale without sacrificing response time and reliability. In this setting, deduplication plays an important role [7]. Deduplication is a well known technique that enables storage providers to store a single copy of the data regardless of how many clients has uploaded the same data.

To see how a typical deduplication scheme works, imagine Alice uploads a file M to the server. Bob then requests to upload his copy of the same file M. The server identifies that M is already stored and simply updates the metadata associated with M to show that the file is owned by both Alice and Bob.

Based on the granularity of deduplication, two different flavours exist:

1. *File-level deduplication*: In this level, deduplication is exploited on the file level. Entire files are compared for deduplication which means only a single copy of each file will be stored.

2. *Block-level deduplication*: The files are divided into blocks and each block is checked for redundancy. This is more fine-grained and has its own advantages and disadvantages.

Based on the deduplication architecture, there are two different strategies:

1. *Server-side deduplication*: Here the clients are oblivious to any of the underlying deduplication techniques. The file is uploaded to the server which may perform deduplication

techniques. As far as the client is concerned, the upload and download functionalities work the same. This technique is also referred to as *target-based deduplication*

2. *Client-side deduplication*: In this technique, the client sends a *tag* to the server before uploading the file. The *tag* serves as a unique identifier of the entire data (e.g., a hash value) and is much shorter than the file. The server checks for redundancy and if a match occurs, the file is not sent over the network. Client-side also called *source-based deduplication*, has the added advantage of saving network bandwidth.

Deduplication along with privacy is a conflicting idea. On one hand, users will want their data to be encrypted for a variety of reasons including personal privacy, corporate policy or even legal reasons. On the other hand, the cloud providers would like to save space by identifying the file uploaded by the user and storing a single copy of the file. Secure deduplication aims to resolve this conflict.

Motivation

File level deduplication achieves significant space savings - as much as 75% to 87% depending on the file type [7]. Using deduplication, cloud storage providers can save storage and thereby money which can in turn be transferred to users in the form of cheaper storage options. By making deduplication compatible with privacy, this can be achieved without the users having to sacrifice privacy.

Most of the existing deduplication schemes enables deduplication when files are identical. There are several use cases when the files uploaded will be very close to each other. For instance, when a user uploads several photos which are taken quickly one after the other, the files will be very close to one another. In this paper, we put forward a scheme that can achieve deduplication across files. In this case, if a user uploads a file to the server and a file that is close to this already exists in the server, then the server will not store the entire file but a small piece (a Δ) using which the original can be recovered later.

Our Work

This paper puts forward a scheme called $\mathsf{DD-Across}$ (deduplication across files) which enables deduplication even for files that are close to each other. The basic idea is to divide the message space into several balls of same radius τ each with a message ψ. We use the hamming distance metric to do this. To encrypt a message m, it first is mapped to the corresponding ψ that it belongs to. This ψ is then encrypted (C_ψ) and then uploaded along with a Δ. Δ combined

with ψ will give back the original message m. As long as the distance τ is small and the original message has high enough entrpy, the Δ leaked will not compromise the security of the encryption.

To see that this enables deduplication, consider Figure 1.1. Suppose Alice has message w_1 that needs to be saved in the server. This will be uploaded to the server as encryption of ψ (C_ψ) and Δ_1. Later, Bob wants to upload w_2 and this will be uploaded as C_ψ and Δ_2. The server needs to store only Δ_2. Of course, if Bob were also to upload w_1, deduplication is still achieved.

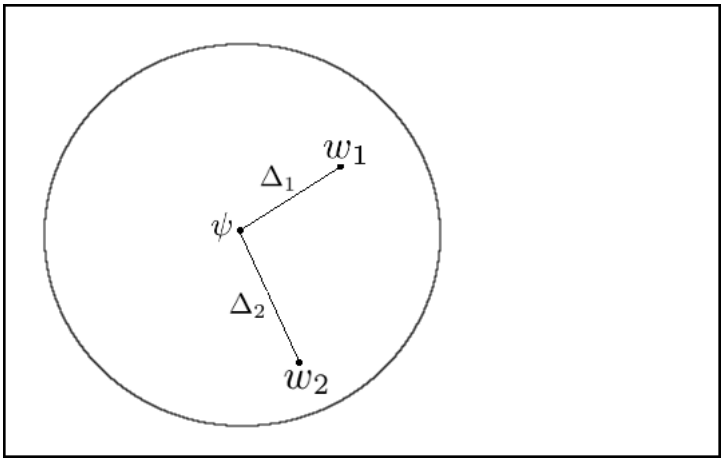

Figure 1.1: Rectangle box represents the entire message space. Only one of many balls is shown

The rest of the paper is organised as follows: Chapter 3 discusses the preliminaries needed and the ingredients that are used in this paper. Chapter 4 explains Interactive MLE scheme introduced in [3], the adversarial model and the security game used. The construction of $\mathsf{DD-Across}$ is explained in chapter 5 and its security is proved in chapter 6.

Chapter 2

Review

We will briefly look at the related works in the Secure Deduplication field

Douceur et al. [5] were the first to propose a novel deterministic cryptosystem called *convergent encryption* (CE) that enables deduplication to work with encryption. The idea was to derive the encryption key from the message itself. This will result in users possessing the same message to end up with the same ciphertext.

Halevi et al. [6] identified several attacks that exploit client-side deduplication and introduced the concept of proofs-of-ownership to mitigate these. This work was extended to get a secure client-side deduplication by Xu et al. in [9].

Bellare et al. in their 2013 paper [8] formalized the notions of security in secure deduplication using a new cryptographic primitive called *Message-Locked Encryption* (MLE) which subsumes CE. The paper was the first to formally argue the security of secure deduplication and analysed the security of existing schemes as well as put forward new practical security schemes.

In [3] Bellare et al. built upon MLE and put forward a new scheme they called *Interactive Message-Locked Encryption* (iMLE) in which upload and download are protocols. They modelled privacy and security as games that enabled to argue for stronger notions of security such as when an adversary controls multiple clients.

Chapter 3

Preliminaries

We discuss in brief the preliminaries that are used in our construction.

Metric spaces

A metric space is a set $\mathcal{M}$ along with a distance function $\mathsf{dis} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^+$ such that distances are defined between all members of the set [4]. In this paper, we will be using the *Hamming metric*.

Hamming Metric

In this metric, $\mathcal{M} = \{0,1\}^n$ and $\mathsf{dis}(w, w')$ is the number of positions in which strings w and w' differ.

We now define two functions Diff and Comb.

Diff - It takes as input two n-length messages $(w_1, w_2) \in \mathcal{M} \times \mathcal{M}$ and outputs the indices in which w_1 and w_2 differs. We will refer to this output as Δ, i.e. $\mathsf{Diff}(w_1, w_2) = \Delta$. If the difference between w_1 and w_2 is bounded by τ, then $|\Delta|$ can be bounded by $Olog(n) \cdot \tau$.

Comb - It takes as inputs, Δ and a message $w_1 \in \mathcal{M}$ and outputs $w_2 \in \mathcal{M}$ such that $\mathsf{Diff}(w_1, w_2) = \Delta$

Error-corecting codes

We will not be using the traditional definition of codes in this. We have an $(\mathcal{M}, K, \tau)$ error correcting code. $\mathcal{M}$ is the message space. The code C is a subset of size K of $\mathcal{M}$. The error correcting distance of C, denoted by τ, is the largest number $\tau > 0$ such that there is at most

one valid code word $c \in C$ for a message w such that $\mathsf{dis}(w, c) \leq \tau$. We denote the encoding and decoding funtions as Enc and Dec respectively.

The encoding and decoding runs in polynomial time. Let Dec be the decoding function. We use the term decoding for the map that finds, given w, the $c \in C$ such that $\mathsf{dis}(w, c) \leq \tau$[4] [1].

Randomness Extractors

We have a family of extractors $\mathsf{Ext} = \{\mathsf{Ext}_\lambda\}$ where $\mathsf{Ext}_\lambda : \{0,1\}^{s(\lambda)} \times \{0,1\}^{l(\lambda)} \to \{0,1\}^{\kappa(\lambda)}$. s is the seed length, l is the input length and κ is the output length. Ext_λ is an (l, m, κ, ϵ)-strong extractor which means that for all min-entropy m distributions W on $\{0,1\}^l$, $\mathbf{SD}((\mathsf{Ext}(W; X), X), (U_\kappa, X)) \leq \epsilon$, where X is uniform on $\{0,1\}^s$[4].

MLE

The paper [8] introduced a new cryptographic primitive called MLE that enables secure deduplication. An MLE scheme is a five-tuple of polynomial-time algorithms $\mathsf{MLE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{T})$. $\mathcal{D}$ and $\mathcal{T}$ are deterministic and the others are randomized.

$$\begin{aligned}
\text{Paramter Generation:} &\quad P \leftarrow \$\mathcal{P}(1^\lambda) \\
\text{Key Generation:} &\quad K \leftarrow \$\mathcal{K}_P(M) \\
\text{Encryption:} &\quad C \leftarrow \$\mathcal{E}_P(K, M) \\
\text{Decryption:} &\quad \mathcal{D}_P(K, C) \in \{0,1\}^* \cup \{\perp\} \\
\text{Tag Generation:} &\quad T \leftarrow \mathcal{T}_P(C)
\end{aligned}$$

There is a *message space* associated with every lambda.

$$MsgSp_{\mathsf{MLE}}(\lambda) \subseteq \{0,1\}^* \quad \forall \lambda \in \mathbb{N}$$

Decryption Correctness:

$$\mathcal{D}_P(K, C) = M \quad \forall \lambda \in \mathbb{N}, \forall P \in [\mathcal{P}(1^\lambda)]$$

Tag Correctness: There exists a negligible function $\delta : \mathbb{N} \to [0,1]$ such that $\forall \lambda \in \mathbb{N}$, $P \in$

[1] For some messages w, a corresponding codeword c may not exist, but when it exists it will be unique. We will, for the rest of this paper, assume that decoding always yields a codeword. The scheme can easily be adapted to handle cases when this doesn't happen, taking only a small hit in deduplication level. Decoding is not the inverse of encoding according to this definition. [4]

$[P(1^\lambda)], \forall M \in \text{MsgSp}_{\mathsf{MLE}}(\lambda)$

$$Pr[\mathcal{T}_P(C) \neq \mathcal{T}_P(C')] \leq \delta(\lambda)$$

where $C \leftarrow \$\mathcal{E}_P(\mathcal{K}_P(M), M)$ and $C' \leftarrow \$\mathcal{E}_P(\mathcal{K}_P(M), M$ are ciphertexts of the same message. MLE is deterministic if $\mathcal{K}$ and $\mathcal{E}$ are deterministic.

An MLE scheme exploits the fact that key is generated from the message and therefore people with identical messages will wnd up with identical ciphertexts. A tag is generated from ciphertext. Tags are used to compare different ciphertexts, and tags from different ciphertexts should only match if they are the same. Thus, tags enable comparison for the underlying messages. If a user uploads a ciphertext to a server and the tag of that ciphertext matches the tag of an existing file, then the server can identify that they are the same and thus need not store the redundant copy.

Privacy for MLE

Semantic security cannot be achieved using any MLE scheme. If the message space $\mathcal{M}$ is predictable, the adversary, given an encryption C of M, can recover M in $O(|\mathcal{M}|)$ trials.

Algorithm 1 Brute-force attack

```
for M' ∈ M do
    if D(K(M'), C) = M' then return M'
    end if
end for
```

So MLE schemes are asked to have semantic security when the message space is unpredictable [1]. That is, messages have high min-entropy.

Guessing Probability

The guessing probability of a random variable X is defined using the following equation

$$\mathbf{GP}(X) = max_x \mathsf{Pr}\,(X = x) = 2^{-H_\infty(X)} \tag{3.1}$$

Hash functions

A hash function $\mathcal{H} : \{0,1\}^n \rightarrow \{0,1\}^m$ is a collision-resistant hash function if

[1] Unpredictability of message space is formalized later in section 4.2

- $m < n$ and
- for all $\mathsf{PPT}\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all security parameters $\lambda \in \mathbb{N}$,

$$\mathsf{Pr}[(x_0, x_1) \leftarrow \mathcal{A}(1^\lambda, \mathcal{H}) : x_0 \neq x_1 \wedge \mathcal{H}(x_0) = \mathcal{H}(x_1)] \leq \mathsf{negl}(\lambda)$$

We denote the family of hash functions to be $\mathsf{H} = (\mathcal{HK}, \mathcal{H})$

Deterministic Symmetric Encryption

Deterministic Symmetric Encryption scheme (D-SE), is defined as a pair of algorithms $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$. Encryption algorithm E takes as input the plaintext $m \in \{0,1\}^*$ and key $k \in \{0,1\}^{\kappa(\lambda)}$ and outputs the ciphertext $c \leftarrow \mathsf{E}(1^\lambda, k, m)$. Decryption returns the plaintext $m \leftarrow \mathsf{D}(1^\lambda, k, c)$. [3] We require D-SE scheme with CPA-security and key-recovery security (KR-secure). We say that a scheme is KR-secure if the probability that the adversary can guess the key from an encryption of a message of his choice is negligible.

Immutability

A table T is said to be *immutable* if no entry $T[t]$ can be changed once it is set. That is, once the table is initialized, values can only be assigned once [3]. An immutable table supports the set-iff-empty (SiffE) operation which is defined as below. It takes as input, a table T, a message m and an index f.

Algorithm 2 $\mathsf{SiffE}(T, m, f)$

1: **if** $T[f] \neq \perp$ **then**
2: $\quad T[f] \leftarrow m$ **return** m
3: **else**
4: $\quad$ **return** $T[f]$
5: **end if**

Chapter 4

Interactive Message-Locked Encryption

This chapter looks at the definition and security for interactive message-locked encryption.

In [3], MLE was extended to a new scheme called iMLE. An interactive message-locked encryption scheme (iMLE is defined by four procedures defined below - one algorithm and three protocols.

1. $\mathsf{Init}(1^\lambda)$ - The initialization algorithm run by the server. This will set up the initial server-side state σ_S.

2. Reg - This protocol, initiated by the client will register the client with the server. The server returns client parameters $\sigma_C \in \{0,1\}^*$ which is output by the protocol.

3. $\mathsf{Put}(M, \sigma_C)$ - The put protocol takes as input the plaintext and the client credentials. It stores the file and outputs an identifier f. Client genertaes key $K \leftarrow \mathcal{K}_p(M)$ and then generates the ciphertext $C \leftarrow \mathcal{E}_p(K, M)$. Note that p is the public parameter generated by the Init procedure.

4. $\mathsf{Get}(f, \sigma_C)$ - The get protocol takes as input the client parameters and file identifier and outputs the plaintext $m \in \{0,1\}^*$

We follow the notation for protocols as given in [3].

- A protocol P with 2 players and q rounds is represented using $2 \times q$-tuple of algorithms.
- $\mathsf{P}[i, j]$ is the algorithm of the i^{th} player at the j^{th} step where $i \in [2]$ and $j \in [q]$.
- $\mathsf{P}[1]$ is the player who starts the protocol. In our case, the client always initiates and hence $\mathsf{P}[1]$ denotes the client.

- $\mathsf{P}[2]$ denotes the server.
- 1^λ, the input a, and a message $M \in \{0,1\}^*$ are passed to each algorithm when invoked.
- Each algorithm outputs a 3-tuple consisting of output s', an outgoing message $M' \in \{0,1\}^*$ and T which is a boolean variable to convey termination.

Soundness

There are two conditions for soundness as described in [3]

1. **Deduplication:** If a file is uploaded to the server by the client and the file already exists in the server, the storage should not increase by the file size. The increase should be independent of the file and must be bounded. Only a small increase for metadata updation is allowed.
   Formally, there exists a bound $l : \mathbb{N} \to \mathbb{N}$, such that for all $\sigma_S \in \{0,1\}^*$, for all $\sigma_C \in \{0,1\}^*$, the expected increase in size of σ''_S over σ'_S when $(f', \sigma'_S) \leftarrow \$\mathsf{Put}(\sigma_C, m)$ is run and then $(f', \sigma''_S) \leftarrow \$\mathsf{Put}(\sigma'_C, m)$ is run is bounded by $l(\lambda)$ [3].

2. **Correct recovery of files:** A legitimate client should be able to recover the file at any time after the file has been put on the server. This is formalized by the Rec game (see Figure 4.1). The adversary is given access to procedures Reg, Init, Step, Msg, State.

   - Reg: Set up a legitimate client L.
   - Init: Enables adversary A to run protocols on behalf of L. inp and P are given as inputs to Init where $\mathsf{P} \in \{\mathsf{Get}, \mathsf{Put}\}$. inp is a valid input for $\mathsf{P}[1,1]$. Returns $j \in \mathbb{N}$, the instance index to A.
   - Step: Takes as input the instance index j and advances it by one algorithm (or step) if the current instance is still active (not terminated). It returns to A the outgoing message sent. When an instance j terminates, WinCheckis run by Step. WinCheck maintains a table T which stores the message m and identifier f in the Put protocol. When j is an instance of Get protocol, WinCheck obtains the identifier f and the recovered message m'. This is checked with $T[f]$ to see if it matches. If there is a mismatch, WinCheck sets the win flag in which case the adversary A wins.

$\underline{\text{MAIN}(1^\lambda)}$ // $\text{REC}^{\mathsf{A}}_{\mathsf{iMLE}}(1^\lambda)$
$\mathsf{win} \leftarrow \mathsf{False}$; $\sigma_S \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$
$\mathsf{A}^{\text{REG},\text{INIT},\text{STEP},\text{MSG},\text{STATE}}(1^\lambda)$; Ret win
 	
$\underline{\text{REG}}$ // Set up the legitimate client L.
$(\sigma_C, \sigma_S) \leftarrow\!\!\$\ \mathsf{Run}(\mathsf{Reg}, \epsilon, \sigma_S)$
 	
$\underline{\text{INIT}(\mathsf{P}, \mathsf{inp})}$ // Start a protocol with L.
If $\mathsf{P} \notin \{\mathsf{Put}, \mathsf{Get}\}$ then ret $\perp$
$\mathtt{p} \leftarrow \mathtt{p} + 1$; $j \leftarrow \mathtt{p}$; $\mathbf{PS}[j] = \mathsf{P}$
$\mathbf{a}[j, 1] \leftarrow \mathsf{inp}$; $\mathtt{N}[j] \leftarrow 1$; $\mathtt{M}[j] \leftarrow \epsilon$; Ret j
 	
$\underline{\text{MSG}(\mathsf{P}, i, \mathtt{M})}$ // Send a message to the server.
If $\mathsf{P} \notin \{\mathsf{Reg}, \mathsf{Put}, \mathsf{Get}, \mathsf{Upd}\}$ then ret $\perp$
$(\sigma_S, \mathtt{M}, \mathtt{N}, \mathtt{T}) \leftarrow\!\!\$\ \mathsf{P}[2, i](1^\lambda, \sigma_S, \mathtt{M})$; Ret $\mathtt{M}$
	$\underline{\text{STEP}(j)}$ // Advance an instance by one step.
	$\mathsf{P} \leftarrow \mathbf{PS}[j]$; $n \leftarrow \mathtt{N}[j]$; $i \leftarrow \mathtt{rd}[j]$
	If $\mathtt{T}[j, n]$ then return $\perp$
	If $n = 2$ then $\mathsf{inp} \leftarrow \sigma_S$ else $\mathsf{inp} \leftarrow \mathbf{a}[j, i]$
	$(\mathsf{outp}, \mathtt{M}[j], \mathtt{T}[j, n]) \leftarrow\!\!\$\ \mathsf{P}[n, i](1^\lambda, \mathsf{inp}, \mathtt{M}[j])$
	If $n = 2$ then
	  $\sigma_S \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 1$; $\mathtt{rd}[j] \leftarrow \mathtt{rd}[j] + 1$
	Else $\mathbf{a}[j, i + 1] \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 2$
	If $\mathtt{T}[j, 1] \wedge \mathtt{T}[j, 2]$ then $\text{WINCHECK}(j)$
	Ret $\mathtt{M}[j]$
	
	$\underline{\text{WINCHECK}(j)}$ // Check if A has won.
	If $\mathbf{PS}[j] = \mathsf{Put}$ then
	  $(\sigma_C, m) \leftarrow \mathbf{a}[j, 1]$; $f \leftarrow \mathtt{last}(\mathbf{a}[j])$; $T[f] \leftarrow m$
	If $\mathbf{PS}[j] = \mathsf{Get}$ then
	  $(\sigma_C, f) \leftarrow \mathbf{a}[j, 1]$; $m' \leftarrow \mathtt{last}(\mathbf{a}[j])$
	  $\mathsf{win} \leftarrow \mathsf{win} \vee (m' \neq T[f])$

Figure 4.1: The recovery correctness game - REC [3]

Security

The primary requirement for any secure deduplication scheme is security for unpredictable data. If the distribution from which the plaintext comes does not have negligible guessing probability, then security cannot be achieved.

Privacy for unpredictable messages is captured by the PRIV game. Unpredictability is formalized using the following notion of a source.

1. An algorithm called *source* S which on inputs 1^λ and string $d \in \{0,1\}^*$ outputs a pair of tuples $(\mathbf{m}_0, \mathbf{m}_1)$.

2. All components of $\mathbf{m}_0$ and $\mathbf{m}_1$ are unique. The length of each component is given by function $\ell : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. $|\mathbf{m}_0[i]| = |\mathbf{m}_1[i]| = \ell(\lambda, i)$. Similarly, there exists $m : \mathbb{N} \to \mathbb{N}$ so that $|\mathbf{m}_0| = |\mathbf{m}_1| = m(\lambda)$.

3. The guessing probability of S is

$$\mathbf{GP}_{\mathsf{S}}(\lambda) = max_{i,b,d}(\mathbf{GP}(\mathbf{m}_b[i]))$$

when $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \$\mathsf{S}(1^\lambda, d)$

4. To model unpredictability, the guessing probability of the source S must be negligible.

PRIV game is formalized as follows.

1. Run INIT to set up the server-side state.
2. Run the source S to get $(\mathbf{m}_0, \mathbf{m}_1)$. A random bit b is selected and $\mathbf{m}_b$ is used as messages to be put in the server.
3. A is invoked with oracle access to REG, PUT, STEP, MSG and STATE. These oracles behave the same way as in REC game, except that STEP does not call WINCHECK. PUT(i) means put plaintext $\mathbf{m}_b[i]$.

MAIN(1^λ) // PRIV$^{\mathsf{S},\mathsf{A}}(1^\lambda)$ $b \leftarrow\$ \{0,1\}$; $\mathsf{p} \leftarrow 0$; $\sigma_S \leftarrow\$ \mathsf{Init}(1^\lambda)$; $\mathbf{m}_0, \mathbf{m}_1 \leftarrow\$ \mathsf{S}(1^\lambda, \epsilon)$ $b' \leftarrow\$ \mathsf{A}^{\text{PUT},\text{UPD},\text{STEP},\text{MSG},\text{REG},\text{STATE}}(1^\lambda)$; Ret $(b = b')$ REG $(\sigma_C, \sigma_S) \leftarrow\$ \mathsf{Run}(\mathsf{Reg}, \epsilon, \sigma_S)$ PUT(i) // Start a Put instance $\mathsf{p} \leftarrow \mathsf{p} + 1$; $\mathbf{PS}[\mathsf{p}] = \mathsf{Put}$; $\mathbf{a}[\mathsf{p}, 1] \leftarrow \vec{m}_b[i]$ $\mathtt{N}[\mathsf{p}] \leftarrow 1$; $\mathtt{M}[\mathsf{p}] \leftarrow \epsilon$; Ret p STATE If $\mathtt{cheat} = \mathsf{False}$ then $\mathsf{done} \leftarrow \mathsf{True}$; ret σ_S else ret $\perp$	MSG$(\mathsf{P}', \mathtt{M})$ // Send a message to the server If $\mathsf{P}' \notin \{\mathsf{Reg}, \mathsf{Put}, \mathsf{Get}, \mathsf{Upd}\}$ then ret $\perp$ $(\sigma_S, \mathtt{M}, \mathtt{T}) \leftarrow\$ \mathsf{P}'[2](1^\lambda, \sigma_S, \mathtt{M})$; Ret $(\sigma_S, \mathtt{M}, \mathtt{N}, \mathtt{T})$ STEP(j) // Advance an instance by one step. $\mathsf{P} \leftarrow \mathbf{PS}[j]$; $n \leftarrow \mathtt{N}[j]$; $i \leftarrow \mathtt{rd}[j]$ If $\mathtt{T}[j, n]$ or done then return $\perp$ If $n = 2$ then $\mathsf{inp} \leftarrow \sigma_S$ else $\mathsf{inp} \leftarrow \mathbf{a}[j, i]$ $(\mathsf{outp}, \mathtt{M}[j], \mathtt{T}[j, n]) \leftarrow\$ \mathsf{P}[n, i](1^\lambda, \mathsf{inp}, \mathtt{M}[j])$ If $n = 2$ then $\sigma_S \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 1$; $\mathtt{rd}[j] \leftarrow \mathtt{rd}[j] + 1$ Else $\mathbf{a}[j, i+1] \leftarrow \mathsf{outp}$; $\mathtt{N}[j] \leftarrow 2$ If $n = 1$ and $\mathtt{T}[j, n]$ then $T_f[\mathbf{a}[j, 1]] \leftarrow \mathtt{last}(\mathbf{a}[j])$

Figure 4.2: The privacy game - PRIV [3]

Adversarial Model

The adversarial model used in this paper is similar to the one used in [3]. A secure deduplication system consists of a server and several clients. The attacker could control several clients simultaneously and could gain access to server storage. The adversary could also interfere the communications where he can read, relay and drop messages of legitimate clients. To simplify the protocol, we assume that the adversary cannot tamper with message contents, reorder messages within a protocol, or redirect from one protocol to another. The adversarial model captures an iMLE scheme running in the presence of an adversary with above mentioned

abilities.

The adversarial model is achieved by using an abstract game G. The games for soundness, security and other properties follow the structure similar to the one below. The objective of the game is for the adversary to violate some property of iMLE guaranteed to legitimate clients.

- G sets up and controls an instance of a server.
- Adversary $\mathcal{A}$ is invoked with access to a set of procedures.
- MSG procedure allows adversary to set up multiple clients and to send arbitrary messages to the server.
- INIT procedure starts protocol instances on behalf of a legitimate client L, using inputs chosen by A.
- STEP procedure advances a protocol instance by running the next step algorithm.
- STATE procedure returns the server's state - including stored ciphertexts, public parameters, etc. Only read only access is gained using this. Otherwise adversary can always tamper with the ciphertexts and win the game.

Chapter 5

DD – Across Construction

We will see the construction of $\mathsf{DD}-\mathsf{Across}$.

Construction

We have with us the following:

- A metric space $(\mathcal{M}, \mathsf{dis})$ with hamming distance as the distance metric.
- An (l, m, κ, ϵ)-strong extractor.
- An error-correcting code $C = (\mathcal{M}, K, \tau)$.
- A collision resistant hash function family $\mathsf{H} = (\mathcal{HK}, \mathcal{H})$.
- $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$ denotes a symmetric encryption scheme.

The $\mathsf{DD}-\mathsf{Across}[C, \mathsf{H}, \mathsf{SE}]$ is defined by one procedure - Init - and three protocols - Reg, Put and Get. The server maintains three tables:

- **fil**: which contains the encryptions of the files uploaded by the clients. Each ciphertext in this table is indexed using a unique identifier derived from itself which we call the *tag*. This table is immutable.
- **delt**: which stores the Δ as discussed in section 1.2. This table is also immutable.
- **own**: which stores the ownership information.

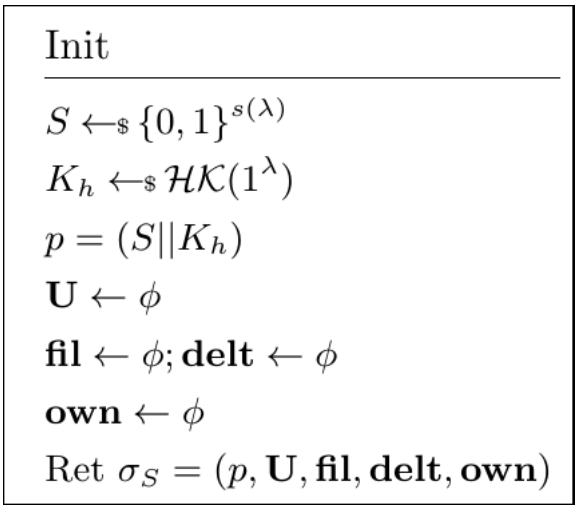

Figure 5.1: The Init procedure. This will set up the server, create empty databases fil and own and runs $\mathcal{P}(1^\lambda)$. fil stores the encrypted files uploaded by the client and is indexed by the tag. own stores the ownership information and associates with each client, the tag of the files uploaded by the client.

The Reg protocol registers a new client with the server. The server picks a unique client identifier and returns it to the client.

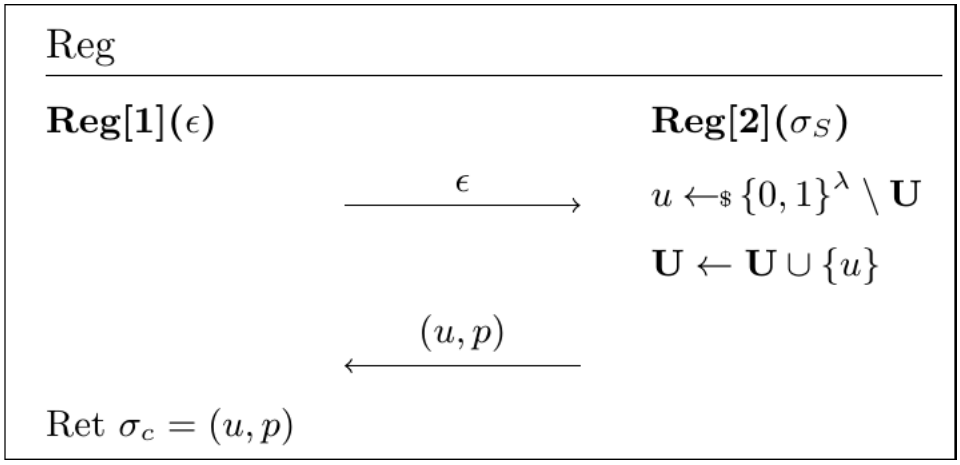

Figure 5.2: The Reg protocol. This returns the client parameters σ_c which includes the client credentials and the keys generated in Init

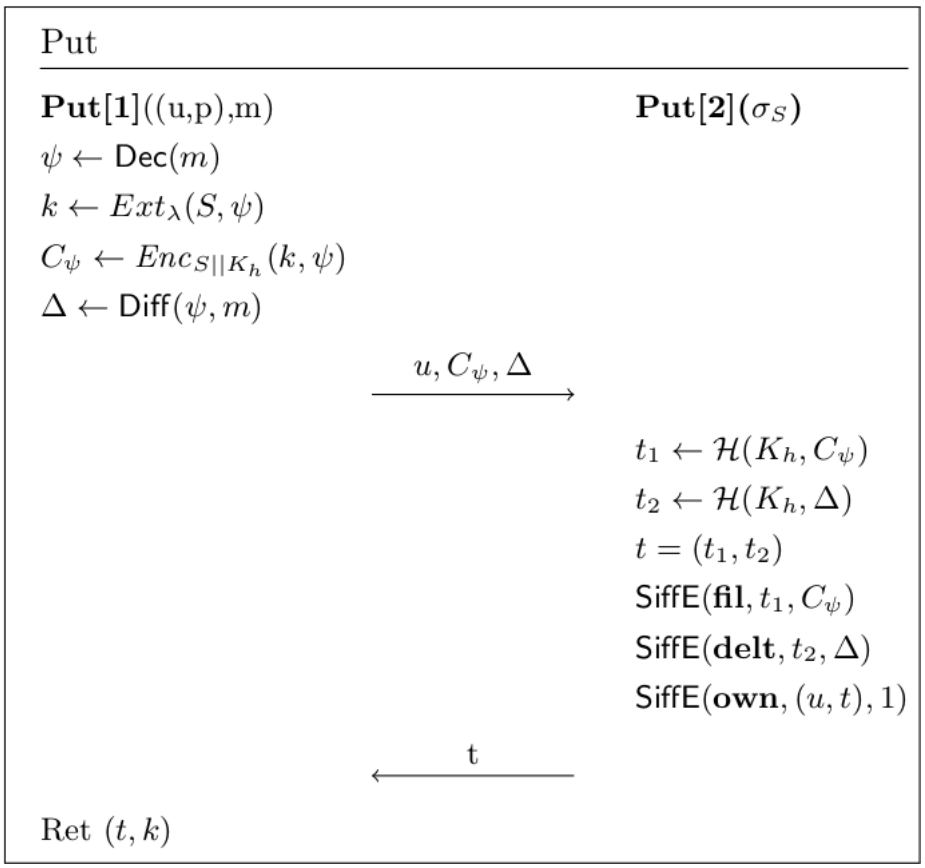

Figure 5.3: The Put protocol. **fil**, **delt** and **own** tables are immutable. Here the Δ is stored in **delt** table and **own** table stores a 1 to show that the client with credentials u owns the file with tag t

In the Put protocol, the message m is first mapped to its codeword ψ. $\Delta = \mathsf{Diff}(\psi, m)$ is computed. The client then sends $(\mathsf{E}(k, \psi), \Delta)$ to the server. The server hashes ψ and Δ to get the tag $t = (t_1 = \mathcal{H}(K_h, \psi), t_2 = \mathcal{H}(K_h, \Delta))$. The message is finally stored in the **fil** and **delt** tables. $\mathbf{fil}[t_1] = C_\psi$ and $\mathbf{delt}[t_2] = \Delta$. The ownership table is also updated to reflect this. $\mathbf{own}[u, t] = 1$. If another client (with credentials u') tries to upload the same file m, then the server only needs to update it on the ownership table. $\mathbf{own}[u', t] = 1$.

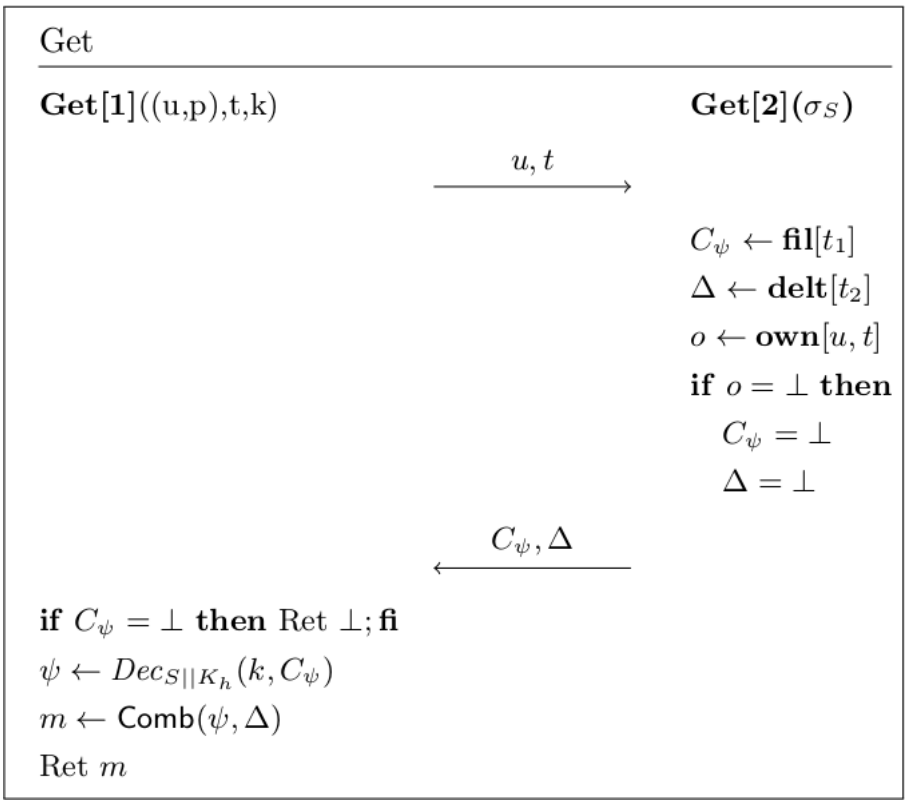

Figure 5.4: The Get protocol.

In the Get protocol, client sends the tag t and credentials to the server. The server verifies if the client owns the file by checking the ownership table and if it does, returns C_ψ and Δ. The client decrypts C_psi to get ψ and applies Comb(ψ, Δ) to get back m.

Chapter 6

Results

Deduplication

It is clear that $\mathsf{DD}-\mathsf{Across}$ enables deduplication since different users with the same file will end up with the same tag and therefore only one copy is saved in the server. We will now state two theorems regarding the privacy and security of $\mathsf{DD}-\mathsf{Across}$

Recovery

Theorem 6.1 *If $\mathcal{E}$ is a correct deterministic symmetric encryption (D-SE) and H is collision resistant, then the above scheme $\mathsf{DD}-\mathsf{Across}[\mathsf{H},\mathcal{E},C]$ is* Rec*-secure.*

Proof

For adversary A to win the Rec game, there must be a mismatch in the plaintext m put on the server and the plaintext m' recovered using Get.
Whenever the client sends the ciphertext $C=(C_\psi,\Delta)$, the tag $t=(t_1,t_2)$ is computed by the server and $\mathbf{fil}[t_1]$, $\mathbf{delt}[t_2]$ and $\mathbf{own}[t]$ are filled. Since the tables are immutable, the ciphertext will not change once it is entered. When the client requests for tag t, the server checks $\mathbf{own}[t]$ and returns $\mathbf{fil}[t_1]$ and $\mathbf{delt}[t_2]$. Thus recovery correctness is always ensured except when a collision occurs. Since we are using a collision resistant hash function, the probability of this happening is negligible.

Privacy

Definition 6.1 *The error-correcting code $C=(\mathcal{M},K,\tau)$ is said to be compatible with a source S with min-entropy $\mu(\lambda)$ iff $2^{\mu(\lambda)-\tau}$ is negligible.*

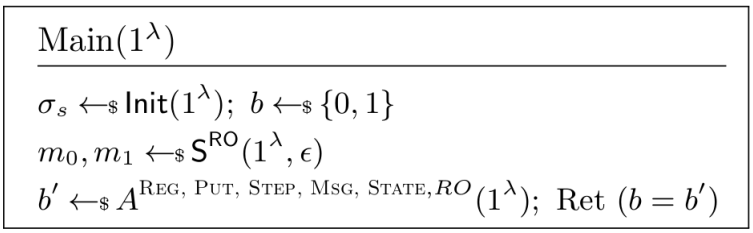

Figure 6.1: The *main* function of the Priv game

This definition is used to put a constraint on the error correcting distance of C. If the error correcting distance τ is large, we cannot ensure privacy. This is because we are leaking Δ which has an entropy bounded by τ. Thus even after leaking τ bits of information, there needs to be enough entropy in the message space to ensure unpredictability.

Theorem 6.2 *If $\mathcal{E}$ is CPA-secure and KR-secure and the code $C = (\mathcal{M}, K, \tau)$ is compatible with the source S, then $\mathsf{DD} - \mathsf{Across}_{RO}[\mathcal{E}, C]$* [1] *is* Priv*-secure.*

Proof

In the Priv game, the source S outputs two vectors $\mathbf{m}_0, \mathbf{m}_1$ where $\mathbf{m}_i$ is a vector over $\{0,1\}^*$. A random bit b is chosen. Adversary can put and get components of $\mathbf{m}_b$ and finally learns the server-side state. A wins if it correctly guesses b.

Let S be an unpredictable PT source and C be a compatible error correcting coding scheme. S outputs $m(\lambda)$ plaintexts each of length $\ell(\lambda, i)$. We denote the polynomial time adversary using A. The number of messages stored by the adversary is bounded by $n : \mathbb{N} \to \mathbb{N}$ and the number of random oracle queries made by source and adversary is bounded by $q_S(\lambda) : \mathbb{N} \to \mathbb{N}$ and $q_A(\lambda) : \mathbb{N} \to \mathbb{N}$
We will use a hybrid argument to show the privacy of the $\mathsf{DD} - \mathsf{Across}$. Consider the Priv game in the figure 6.1.

The Reg, Step, Msg and State are same as in the Priv game. The Put protocol changes in each hybrid game. Let H_1 denote the actual Put protocol from figure 5.3 with the only difference being that instead of H, it uses RO. In the game H_2, the k is sampled randomly instead of using the extractor output. This will give the adversary with an ϵ advantage since

[1] $\mathsf{DD} - \mathsf{Across}_{RO}$ is the ROM analogue of $\mathsf{DD} - \mathsf{Across}$ which models H as a random oracle

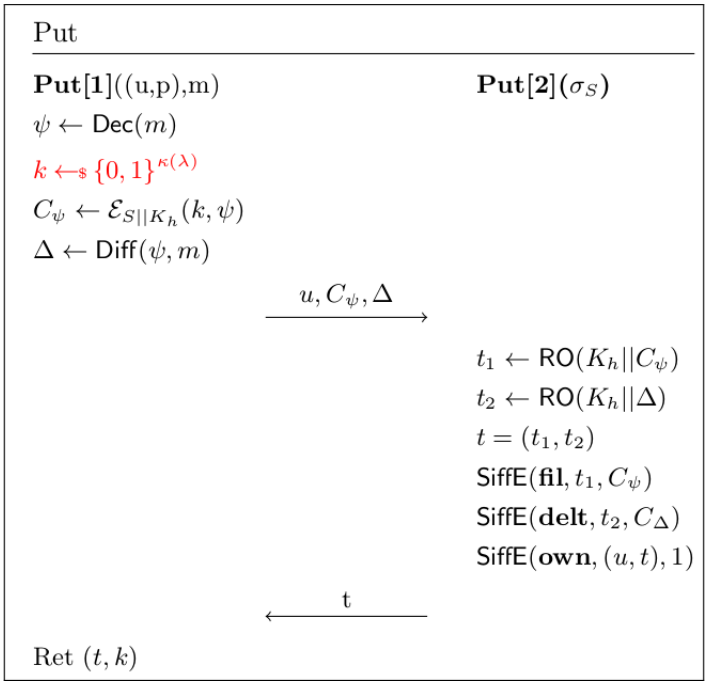

Figure 6.2: The Put protocol in game H_2

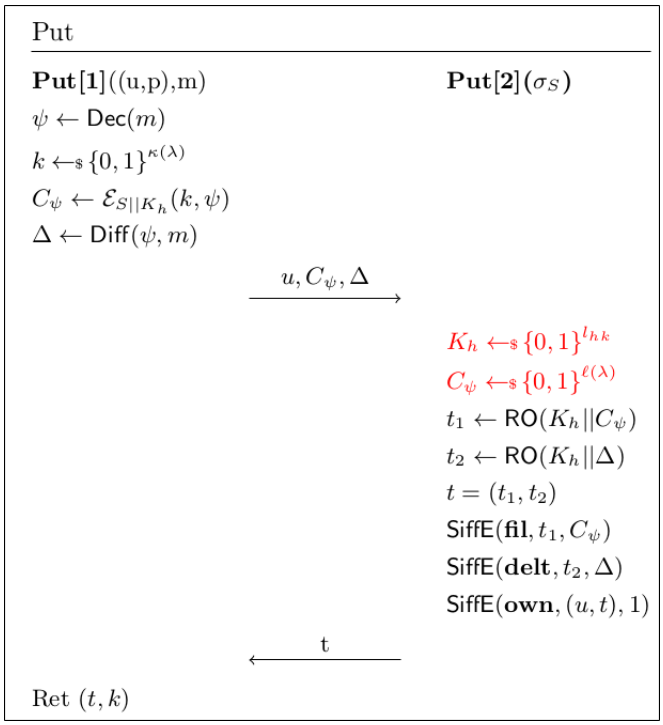

Figure 6.3: The Put protocol in game H_3

the extractor output is ϵ close to uniform distribution.

$$\mathsf{Pr}\left(\textsc{Priv}^{\mathsf{S,A}}_{\mathsf{DD-Across}}(a^\lambda)\right) = \mathsf{Pr}\left(H_1^{\mathsf{S,A}}\right) \leq \mathsf{Pr}\left(H_2^{\mathsf{S,A}}\right) + \epsilon \tag{6.1}$$

In H_3, the RO queries for t_1 are made using random strings. The adversary can detect this only if it had queried $\mathsf{RO}(K_h||C_\psi)$. The probability for this bad event is bounded by $q_A(\lambda)n(\lambda)/2^{\ell(\lambda,i)-\tau}$ where q_A is the bound on the number of RO queries made by A.

$$\mathsf{Pr}\left(H_2^{S,A} \text{ sets } \mathsf{bad}\right) - \mathsf{Pr}\left(H_3^{S,A} \text{ sets } \mathsf{bad}\right) \leq \frac{q_A(\lambda)n(\lambda)}{2^{m(\lambda)-\tau}} + q_s(\lambda)n(\lambda)GP_S(\lambda) \tag{6.2}$$

In game H_4, ψ is an encryption of a random string. The CPA-security of the *Enc* means that adversary cannot distinguish with any significant advantage. In game H_4, there is no information about the message m at all other than Δ. By Definition 6.1, the min-entropy of the message m is still high. Thus,

$$\mathsf{Pr}\left(H_4^{A} \text{ sets } \mathsf{bad}\right) \leq \frac{q_A(\lambda)n(\lambda)}{2^{m(\lambda)-\tau}} + q_s(\lambda)n(\lambda)GP_S(\lambda) \tag{6.3}$$

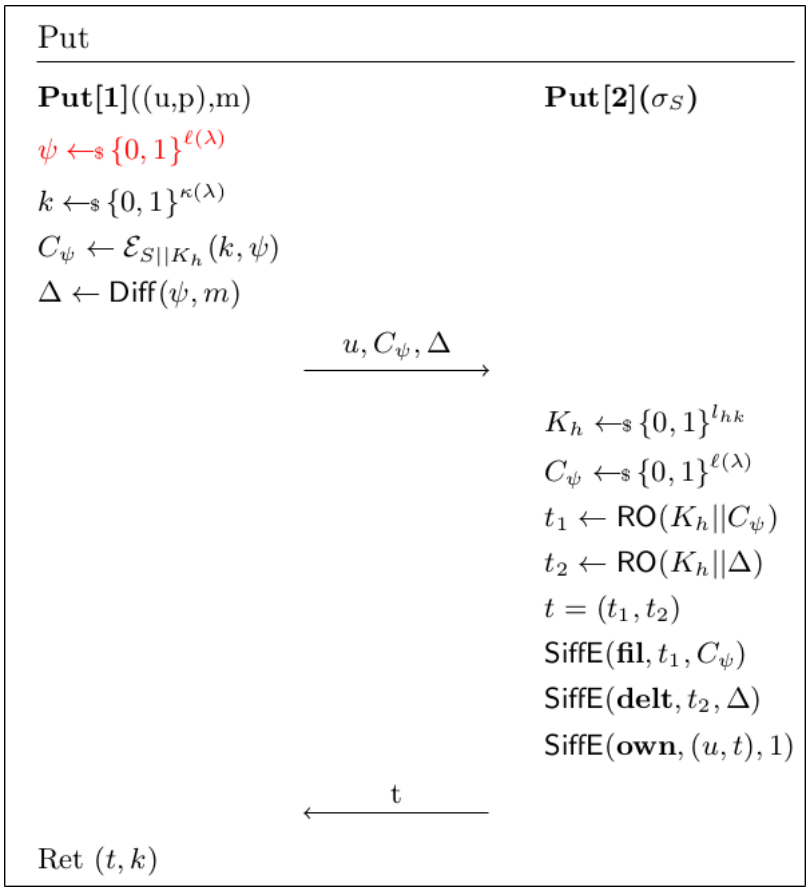

Figure 6.4: The Put protocol in game H_4

Since at each of the hybrid steps there is only a negligible hit in the probability of adversary winning, adding all the probabilities will still give only a negligible advantage to the adversary. Thus DD − Across is Priv-secure.

The scheme is efficient since when files that are close together are uploaded, the storage grows only by an additional factor of Δ for each new file. If the files are identical, then the overhead is minimal.

Chapter 7

Conclusion and Future Works

We have put forward a scheme called $\mathsf{DD-Across}$ that enables deduplication across files and have proved its soundness and privacy. We are able to save space by only storing the "delta" between two files that are close to each other and maps to the same codeword.

As part of the future work, the scheme can be implemented to test real world space savings and efficiency. Incorporating the properties of Fuzzy extractors [4] to this is both a challenging and interesting future work.

Bibliography

[1] Dropbox. https://www.dropbox.com/. 1

[2] Google Drive. https://drive.google.com. 1

[3] Mihir Bellare and Sriram Keelveedhi. *Public-Key Cryptography – PKC 2015: 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 – April 1, 2015, Proceedings*, chapter Interactive Message-Locked Encryption and Secure Deduplication, pages 516–538. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-46447-2. doi: 10.1007/978-3-662-46447-2_23. URL http://dx.doi.org/10.1007/978-3-662-46447-2_23. v, 3, 4, 8, 9, 10, 11, 12

[4] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. *Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data*, pages 523–540. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-24676-3. doi: 10.1007/978-3-540-24676-3_31. URL http://dx.doi.org/10.1007/978-3-540-24676-3_31. 5, 6, 23

[5] J.R. Douceur, A. Adya, W.J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624, 2002. doi: 10.1109/ICDCS.2002.1022312. 4

[6] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 491–500, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046765. URL http://doi.acm.org/10.1145/2046707.2046765. 4

[7] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 1–1,

Berkeley, CA, USA, 2011. USENIX Association. ISBN 978-1-931971-82-9. URL http://dl.acm.org/citation.cfm?id=1960475.1960476. 1, 2

[8] Thomas Ristenpart Mihir Bellare, Sriram Keelveedhi. Message-locked encryption and secure deduplication. In *Advances in Cryptology EUROCRYPT 2013*, CCS '11, pages 491–500, New York, NY, USA, 2011. Springer Berlin Heidelberg. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046765. URL http://doi.acm.org/10.1145/2046707.2046765. 4, 6

[9] Jia Xu, Ee-Chien Chang, and Jianying Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 195–206, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1767-2. doi: 10.1145/2484313.2484340. URL http://doi.acm.org/10.1145/2484313.2484340. 4