Secure Deduplication Across Files

Nithin V Nath

Advisor: Dr. Bhavana Kanukurthi

Department of Computer Science and Automation
Indian Institute of Science

23-Jun-2016

Outline

Deduplication

- Large amount of data stored in cloud storage.
- Multiple users store the same file.
- Service providers need to employ space saving techniques to keep cost down.

Definition

Technique that enables storage providers to store a single copy of the data.

Deduplication in Action

- Alice uploads a file M to the server S.
- Bob requests to upload his copy of the same file M to S.
- The server identifies that M is already stored.
- Server updates *only the metadata* associated with M to show that the file is owned by both Alice and Bob.

Outline

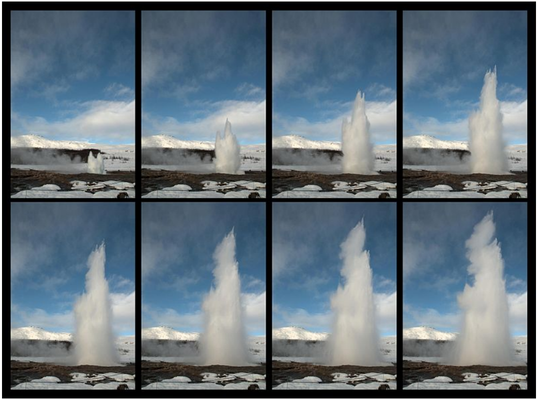
Secure Deduplication

- Deduplication along with privacy is a conflicting idea
- Users would like their data to be encrypted
- To enable deduplication, servers need to “know the data”

How to achieve Secure Deduplication

- **Key Idea**: Derive the key from the message itself.
- Generate a "tag" from the ciphertext.
- Compare the tags of different ciphertexts to see if they are the same.
- We can achieve security only for unpredictable data.

Motivation I



- Photos taken one after the other are often *almost* identical to each other.

Motivation II

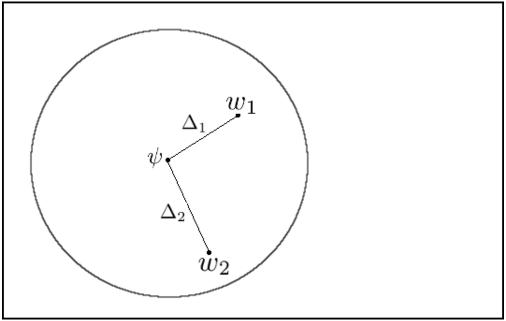
- These multiple files are not supported by existing file level deduplication.
- **Challenge**: Identify that plaintexts underneath these ciphertexts are close to each other and store only the difference.
- **Problem Statement:** Achieve deduplication across files, which are close to each other, in a privacy preserving way.

Photo courtesy: https://commons.wikimedia.org/wiki/User:Papa_November

Outline

Our Work

- DD – Across (deduplication across files) which enables deduplication even for files that are close to each other.



Related Work - Interactive Message Locked Encryption

- Uses interaction.
- Defined using one algorithm and three protocols
    1. Init(1^λ) - The initialization algorithm.
    2. Reg - Register a client with the server.
    3. Put(M, σ_C) - Puts a plaintext M and returns f, an identifier
    4. Get(f, σ_C) - Fetches the file f.

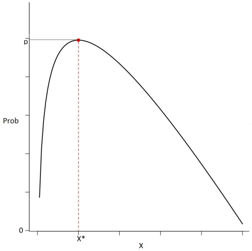
Outline

Entropy

- Entropy is a measure of randomness

- Min-entropy of X is the negative log of maximum predictability.

$$H_\infty(X) = -\log\left(\max_x \Pr\left[X = x\right]\right)$$



- Guessing Probability of X is $GP(X) = 2^{-H_\infty(X)}$

Extractors

- A family of extractors $\text{Ext} = \{\text{Ext}_\lambda\}$
- $\text{Ext}_\lambda : \{0,1\}^{s(\lambda)} \times \{0,1\}^{l(\lambda)} \rightarrow \{0,1\}^{\kappa(\lambda)}$
  - s - seed length
  - l - input length
  - κ - output length
- (l, m, κ, ϵ)-strong extractor $\Rightarrow \forall$ min-entropy m distributions W on $\{0,1\}^l$

$$\mathbf{SD}\left(\left(\text{Ext}\left(W; X\right), X\right), \left(U_\kappa, X\right)\right) \leq \epsilon$$

where X is uniform on $\{0,1\}^s$

Source

- The source S is an algorithm: $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow S(1^\lambda, d)$ where $d \in \{0,1\}^*$.
- $\mathbf{m}_0$ and $\mathbf{m}_1$ are vectors over $\{0,1\}^*$
- $|\mathbf{m}_0| = |\mathbf{m}_1| = m(\lambda)$.
- Guessing probability of source:

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

- We require $\mathbf{GP}_{\mathsf{S}}(\lambda)$ to be negligible.

Deterministic Encryption

- $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$
- $c \leftarrow \mathsf{E}(1^\lambda, k, m)$
- $m \leftarrow \mathsf{D}(1^\lambda, k, c)$
- **Correctness:** $\mathsf{D}\left(1^\lambda, k, \mathsf{E}\left(1^\lambda, k, m\right)\right) = m, \quad \forall$ plaintexts $m \in \{0,1\}^*, \quad \forall$ keys $k \in \{0,1\}^{\kappa(\lambda)}, \quad \forall \lambda \in \mathbb{N}.$

Error Correcting Codes

- $(\mathcal{M}, K, \tau)$-code C.
- C is a subset $\{w_0, w_1, \ldots, w_K\}$ of $\mathcal{M}$.
- $\tau > 0$ is the largest number such that there is at most one valid code word $c \in C$ for a message w such that $\text{dis}(w, c) \leq \tau$.
- Enc - The map from i to w_i.
- Dec - The map that finds, given w, the $c \in C$ such that $\text{dis}(w, c) \leq \tau$

Collision Resistant Hash Functions

- $\mathcal{H} : \{0, 1\}^n \rightarrow \{0, 1\}^m$
- Collision resistant if
  - $m < n$ and
  - $\forall$PPT$\mathcal{A}$, $\exists$ a negligible function negl(λ) such that $\forall$ security parameters $\lambda \in \mathbb{N}$,

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

- Family of hash functions: $\mathrm{H} = (\mathcal{HK}, \mathcal{H})$

Outline

Setting

- An honest-but-curious server.
- A set of clients.
- $\mathcal{A}$ can control a subset of these clients.
- Formally modelled using a game G.
- G sets up and controls an instance of a server.

Adversarial Model

- Adversary $\mathcal{A}$ is invoked with oracle access to the following:
  - $\textsc{Msg}()$: allows adversary to set up multiple clients and to send arbitrary messages to the server.
  - $\textsc{Init}()$: starts protocol instances on behalf of a legitimate client L, using inputs chosen by A.
  - $\textsc{Step}()$: advances a protocol instance by running the next step algorithm.
  - $\textsc{State}()$: returns the server's state - including stored ciphertexts, public parameters, etc.

The recovery game Rec

Challenger

$win \leftarrow False$

$\sigma_S \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$

Adversary

Reg() //Set up a legitimate client

Init()

Step()

Msg()

State()

$win \leftarrow$ WinCheck()

$\downarrow$

win

The privacy game $\textsc{Priv}$

Challenger

$b \leftarrow_{\$} \{0,1\}$

$\sigma_S \leftarrow_{\$} \mathsf{Init}(1^\lambda)$

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

Ret $b = b'$

Adversary

$\textsc{Reg}()$

$\textsc{Put}(i)$

$\textsc{Step}()$

$\textsc{Msg}()$

$\textsc{State}()$

$\downarrow$

b'

Outline

1. Introduction
   - Secure Deduplication
   - Contribution
   - Preliminaries
2. Construction
   - Adversarial Model
   - DD-Across
   - Recovery and Privacy
3. Conclusion

DD-Across Ingredients

- A metric space $(\mathcal{M}, \text{dis})$ 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.

DD-Across Construction

- $\mathrm{DD} - \mathrm{Across}[C, \mathrm{H}, \mathrm{SE}]$.
- Server maintains 3 tables
  - **fil**: which contains the encryptions of the files uploaded by the clients.
  - **delt**: which stores the Δ.
  - **own**: which stores the ownership information.

DD-Across Construction - Init

Init

$S \xleftarrow{\$} \{0,1\}^{s(\lambda)}$

$K_h \xleftarrow{\$} \mathcal{HK}(1^\lambda)$

$p = (S||K_h)$

$\mathbf{U} \leftarrow \phi$

$\mathbf{fil} \leftarrow \phi$; $\mathbf{delt} \leftarrow \phi$

$\mathbf{own} \leftarrow \phi$

Ret $\sigma_S = (p, \mathbf{U}, \mathbf{fil}, \mathbf{delt}, \mathbf{own})$

DD-Across Construction - Reg

Reg		
Reg[1](ϵ)		**Reg[2](σ_S)**
	$\xrightarrow{\epsilon}$	$u \leftarrow_{\$} \{0,1\}^\lambda \setminus \mathbf{U}$
		$\mathbf{U} \leftarrow \mathbf{U} \cup \{u\}$
	$\xleftarrow{(u,p)}$	
Ret $\sigma_c = (u,p)$		

DD-Across Construction - Put

Put

Put[1]((u,p),m)		**Put[2]**(σ_S)
$\psi \leftarrow \mathsf{Dec}(m)$		
$k \leftarrow Ext_\lambda(S, \psi)$		
$C_\psi \leftarrow Enc_{S\|\|K_h}(k, \psi)$		
$\Delta \leftarrow \mathsf{Diff}(\psi, m)$		
	$\xrightarrow{u, C_\psi, \Delta}$	
		$t_1 \leftarrow \mathcal{H}(K_h, C_\psi)$
		$t_2 \leftarrow \mathcal{H}(K_h, \Delta)$
		$t = (t_1, t_2)$
		$\mathsf{SiffE}(\mathbf{fil}, t_1, C_\psi)$
		$\mathsf{SiffE}(\mathbf{delt}, t_2, \Delta)$
		$\mathsf{SiffE}(\mathbf{own}, (u, t), 1)$
	$\xleftarrow{\text{t}}$	
Ret (t, k)		

DD-Across Construction - Get

Get

Get[**1**]((u,p),t,k) | **Get**[**2**](σ_S)

$$\xrightarrow{u, t}$$

$C_\psi \leftarrow \textbf{fil}[t_1]$

$\Delta \leftarrow \textbf{delt}[t_2]$

$o \leftarrow \textbf{own}[u, t]$

if $o = \perp$ **then**

$C_\psi = \perp$

$\Delta = \perp$

$$\xleftarrow{C_\psi, \Delta}$$

if $C_\psi = \perp$ **then** Ret $\perp$; **fi**

$\psi \leftarrow Dec_{S||K_h}(k, C_\psi)$

$m \leftarrow \mathsf{Comb}(\psi, \Delta)$

Ret m

Outline

DD-Across Recovery

- Recovery is guaranteed.
- For $\mathcal{A}$ to "win", $m_{\text{put}} \neq m_{\text{retrieved}}$
- Immutability of the tables means once put, file cannot be changed.
- Reduces to the security of hash collision.

DD-Across Privacy

Definition

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.

Theorem

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

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

DD-Across Privacy Hybrids

Put

$\mathbf{Put[1]}((u,p),m)$		$\mathbf{Put[2]}(\sigma_S)$
$\psi \leftarrow \mathsf{Dec}(m)$		
$k \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$		
$C_\psi \leftarrow \mathcal{E}_{S\|\|K_h}(k,\psi)$		
$\Delta \leftarrow \mathsf{Diff}(\psi, m)$		
	$\xrightarrow{u, C_\psi, \Delta}$	
		$t_1 \leftarrow \mathsf{RO}(K_h\|\|C_\psi)$
		$t_2 \leftarrow \mathsf{RO}(K_h\|\|\Delta)$
		$t = (t_1, t_2)$
		$\mathsf{SiffE}(\mathbf{fil}, t_1, C_\psi)$
		$\mathsf{SiffE}(\mathbf{delt}, t_2, C_\Delta)$
		$\mathsf{SiffE}(\mathbf{own}, (u,t), 1)$
	$\xleftarrow{t}$	
Ret (t, k)		

Figure: The Put protocol in game H_2

DD-Across Privacy Hybrids

Put

$\mathbf{Put}[\mathbf{1}]((\mathrm{u,p}),\mathrm{m})$		$\mathbf{Put}[\mathbf{2}](\sigma_S)$
$\psi \leftarrow \mathsf{Dec}(m)$		
$k \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$		
$C_\psi \leftarrow \mathcal{E}_{S\|\|K_h}(k,\psi)$		
$\Delta \leftarrow \mathsf{Diff}(\psi, m)$		
	$\xrightarrow{u, C_\psi, \Delta}$	
		$K_h \leftarrow_{\$} \{0,1\}^{l_{hk}}$
		$C_\psi \leftarrow_{\$} \{0,1\}^{\ell(\lambda)}$
		$t_1 \leftarrow \mathsf{RO}(K_h\|\|C_\psi)$
		$t_2 \leftarrow \mathsf{RO}(K_h\|\|\Delta)$
		$t = (t_1, t_2)$
		$\mathsf{SiffE}(\mathbf{fil}, t_1, C_\psi)$
		$\mathsf{SiffE}(\mathbf{delt}, t_2, \Delta)$
		$\mathsf{SiffE}(\mathbf{own}, (u,t), 1)$
	$\xleftarrow{\mathrm{t}}$	
Ret (t, k)		

Figure: The Put protocol in game H_3

DD-Across Privacy Hybrids

Put

Put[**1**]((u,p),m)

$\psi \leftarrow_{\$} \{0,1\}^{\ell(\lambda)}$

$k \leftarrow_{\$} \{0,1\}^{\kappa(\lambda)}$

$C_\psi \leftarrow \mathcal{E}_{S||K_h}(k, \psi)$

$\Delta \leftarrow \mathsf{Diff}(\psi, m)$

$\xrightarrow{u, C_\psi, \Delta}$

Put[**2**](σ_S)

$K_h \leftarrow_{\$} \{0,1\}^{l_{hk}}$

$C_\psi \leftarrow_{\$} \{0,1\}^{\ell(\lambda)}$

$t_1 \leftarrow \mathsf{RO}(K_h || C_\psi)$

$t_2 \leftarrow \mathsf{RO}(K_h || \Delta)$

$t = (t_1, t_2)$

$\mathsf{SiffE}(\mathbf{fil}, t_1, C_\psi)$

$\mathsf{SiffE}(\mathbf{delt}, t_2, \Delta)$

$\mathsf{SiffE}(\mathbf{own}, (u, t), 1)$

$\xleftarrow{\text{t}}$

Ret (t, k)

Figure: The Put protocol in game H_4

Open Problems and Future Work

- DD – Across allows deduplication across files when the files map to same code-word.
- Implementing the scheme to record real world performance gains.

For Further Reading

Mihir Bellare, Sriram Keelveedhi, Thomas Ristenpart
Message-Locked Encryption and Secure Deduplication.
Advances in Cryptology EUROCRYPT, 2013.

Mihir Bellare, Sriram Keelveedhi
Interactive Message-Locked Encryption and Secure Deduplication.
Public Key Cryptography, 2015.

Thank You!