# 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


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

# 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 and simply updates the metadata associated with $M$ to show that the file is owned by both Alice and Bob.
- Make this an image

# Outline

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

# Secure Deduplication

- Deduplication along with privacy is a conflicting idea
- Users would like their data to be encrypted
- Storage providers would like to identify the file uploaded by user to enable deduplication.

# Motivation

- Photos taken one after the other are often *almost* identical to each other.
- These multiple files are not supported in traditional file level deduplication.
- **Challenge**: Identify that plaintexts underneath these ciphertexts are close to each other and store only the difference.

# Problem Statement

- Achieving deduplication across files in a privacy preserving way.

# 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.

# Related Work - Interactive Message Locked Encryption

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

# Outline

1 Introduction
- Secure Deduplication
- Preliminaries
- Contributions

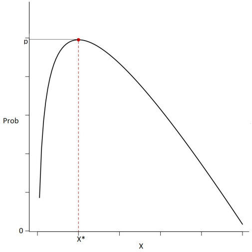
2 Construction
- Adversarial Model
- DD-Across
- Recovery and Privacy

3 Conclusion

# Entropy

- Entropy is a measure of randomness
- Min-entropy of $X$ is the negative log of maximum predictability.

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



# Extractors

# Source

- Formalizing the notion of unpredictability.
- $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow S(1^\lambda, d)$ where $d \in \{0,1\}^*$.
- All components of $\mathbf{m}_0$ and $\mathbf{m}_1$ are unique.
- $|\mathbf{m}_0| = |\mathbf{m}_1| = m(\lambda)$.

# Deterministic Encryption

- $\mathsf{SE} = (\mathsf{E}, \mathsf{D})$
- $c \leftarrow \mathsf{E}(1^{\lambda}, k, m)$
- $m \leftarrow \mathsf{D}(1^{\lambda}, k, c)$
- Why is this meaningful in this setting?

# Error Correcting Codes

- $(\mathcal{M}, K, \tau)$-code $C$.
- $C$ is a subset $\{w_0, w_1, \dots, w_K\}$ 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$(\lambda)$ such that $\forall$ security parameters $\lambda \in \mathbb{N}$,

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

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

# Outline

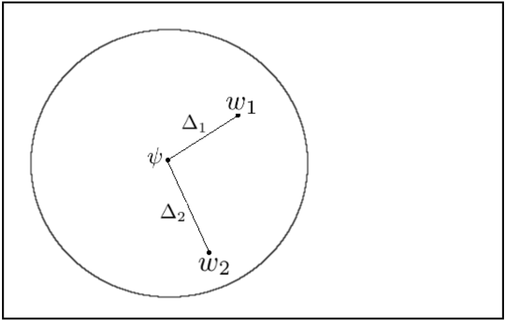
1 Introduction
- Secure Deduplication
- Preliminaries
- Contributions

2 Construction
- Adversarial Model
- DD-Across
- Recovery and Privacy

3 Conclusion

# Our Work

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



# Outline


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

# 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 $\textsc{Rec}$

**Challenger**

$win \leftarrow False$

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

**Adversary**

$\textsc{Reg}()$//Set up a legitimate client

$\textsc{Init}()$

$\textsc{Step}()$

$\textsc{Msg}()$

$\textsc{State}()$

$win \leftarrow \textsc{WinCheck}()$

$\downarrow$

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
- Preliminaries
- Contributions

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 $(l, m, \kappa, \epsilon)$-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 $\Delta$.
  - **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]($\epsilon$) | | Reg[2]($\sigma_S$) |
| --- | --- | --- |
| | $\xrightarrow{\epsilon}$ | $u \leftarrow_{\$} \{0,1\}^{\lambda} \setminus \mathbf{U}$ |
| | | $\mathbf{U} \leftarrow \mathbf{U} \cup \{u\}$ |
| | $\xleftarrow{(u,p)}$ | |
| Ret $\sigma_c = (u,p)$ | | |

# DD-Across Construction - Put

Put

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

# DD-Across Construction - Get

| Get | | |
|---|---|---|
| $\mathbf{Get}[\mathbf{1}]((u,p),t,k)$ | | $\mathbf{Get}[\mathbf{2}](\sigma_S)$ |
| | $\xrightarrow{u,t}$ | |
| | | $C_\psi \leftarrow \mathbf{fil}[t_1]$<br>$\Delta \leftarrow \mathbf{delt}[t_2]$<br>$o \leftarrow \mathbf{own}[u,t]$<br>**if** $o = \perp$ **then**<br>$C_\psi = \perp$<br>$\Delta = \perp$ |
| | $\xleftarrow{C_\psi, \Delta}$ | |
| **if** $C_\psi = \perp$ **then** Ret $\perp$; **fi**<br>$\psi \leftarrow Dec_{S\|\|K_h}(k, C_\psi)$<br>$m \leftarrow \mathsf{Comb}(\psi, \Delta)$<br>Ret $m$ | | |

# Outline


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

# DD-Across Recovery

- Recovery is guaranteed.
- For $\mathcal{A}$ to "win", $m_{\text{put on server}} \neq m_{\text{retrieved from server}}$
- 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* $\mathrm{P{\scriptsize RIV}}$*-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**

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

Figure: The Put protocol in game $H_2$

# DD-Across Privacy Hybrids

**Put**

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

Figure: The Put protocol in game $H_3$

# DD-Across Privacy Hybrids

Put

| **Put[1]**((u,p),m) | | **Put[2]**($\sigma_S$) |
|---|---|---|
| $\psi \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$ | | |
| $k \leftarrow_\$ \{0,1\}^{\kappa(\lambda)}$ | | |
| $C_\psi \leftarrow \mathcal{E}_{S\|\|K_h}(k,\psi)$ | | |
| $\Delta \leftarrow \mathsf{Diff}(\psi, m)$ | | |
| | $\xrightarrow{u, C_\psi, \Delta}$ | |
| | | $K_h \leftarrow_\$ \{0,1\}^?$ |
| | | $C_\psi \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$ |
| | | $t_1 \leftarrow \mathsf{RO}(K_h\|\|C_\psi)$ |
| | | $t_2 \leftarrow \mathsf{RO}(K_h\|\|\Delta)$ |
| | | $t = (t_1, t_2)$ |
| | | $\mathsf{SiffE}(\mathbf{fil}, t_1, C_\psi)$ |
| | | $\mathsf{SiffE}(\mathbf{delt}, t_2, \Delta)$ |
| | | $\mathsf{SiffE}(\mathbf{own}, (u,t), 1)$ |
| | $\xleftarrow{\text{t}}$ | |
| 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.
- Connection of Fuzzy Extractors with the existing scheme.
- Implementing the scheme to record real world performance gains.

# Thank you

- Questions?