

Efficient Dynamic Searchable Encryption with Forward Privacy

Mohammad
Etemad



Alptekin
Küpçü



Charalampos
Papamanthou



David
Evans

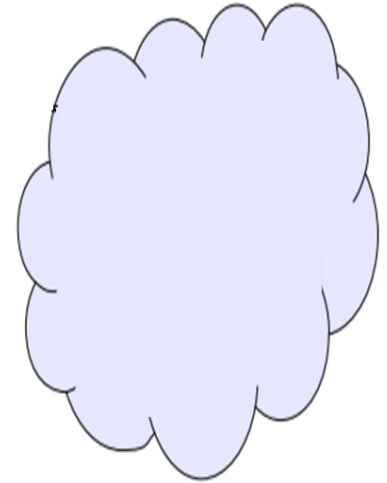
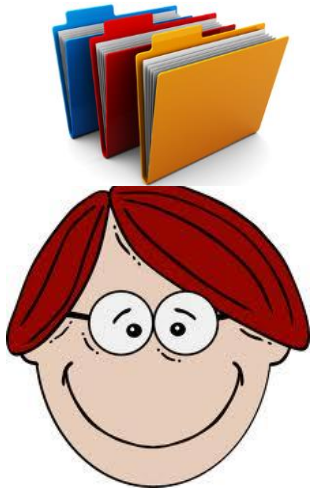


Problem Definition

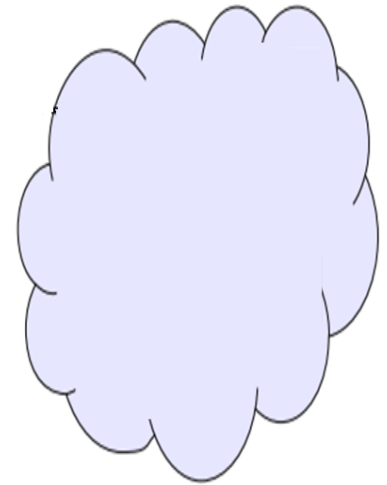
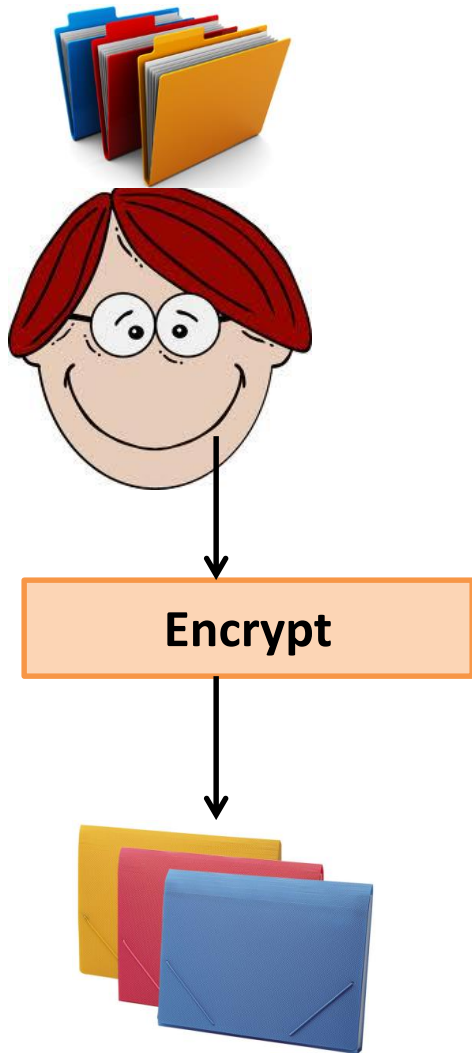
- Outsourced data should be **encrypted** for confidentiality.
- The user want to perform **search** to access a particular data or selectively retrieve the outsourced files.
- **Search over the encrypted data?**



Trivial Secure but Inefficient Solution



Trivial Secure but Inefficient Solution



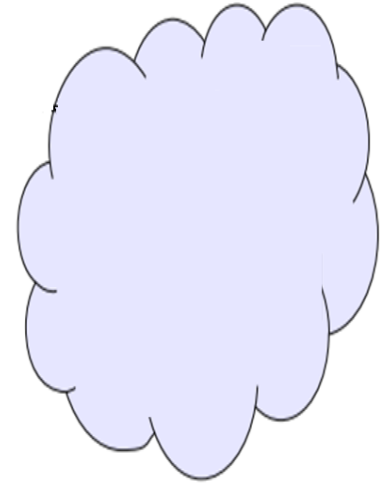
Trivial Secure but Inefficient Solution



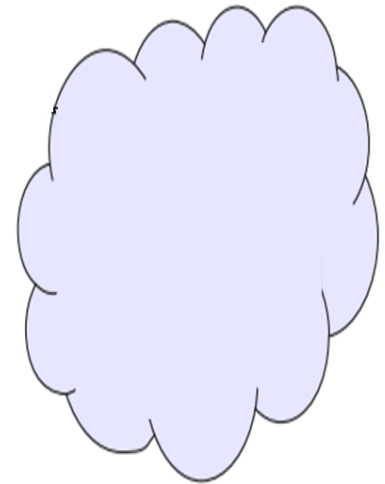
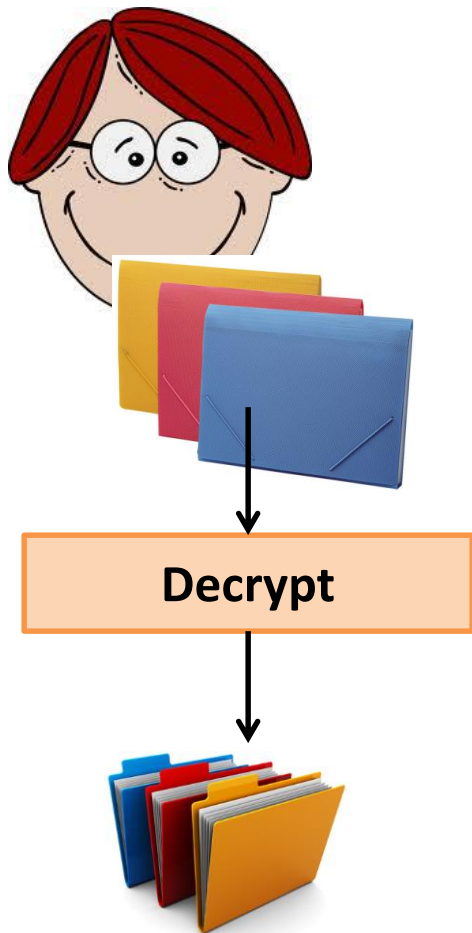
Trivial Secure but Inefficient Solution



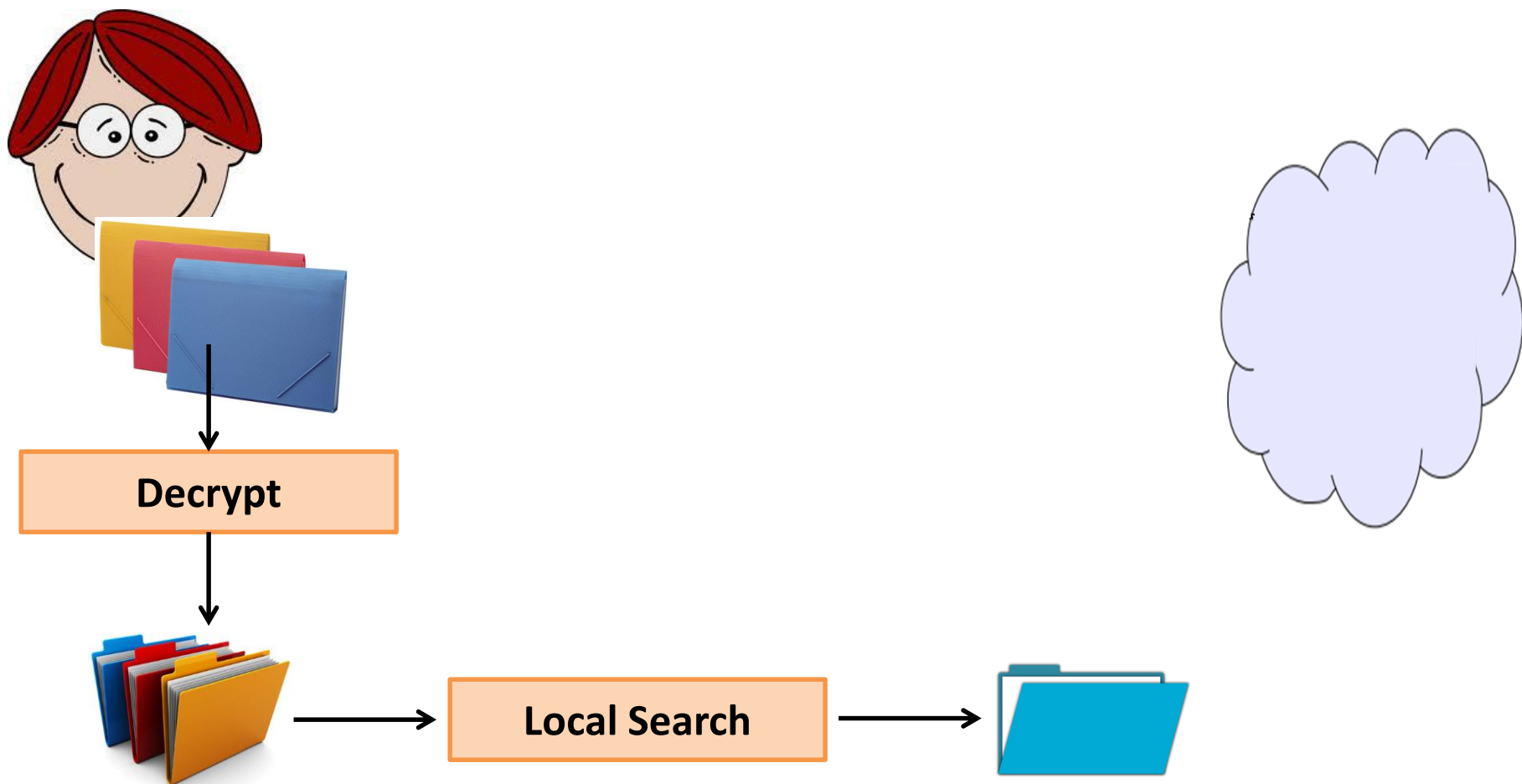
Trivial Secure but Inefficient Solution



Trivial Secure but Inefficient Solution



Trivial Secure but Inefficient Solution



Searchable Encryption

- Index-based solutions



Searchable Encryption

- Index-based solutions
- Files $\mathbf{f} = \{f_1, f_2, \dots, f_n\}$
- Dictionary $W = \{w_1, w_2, \dots, w_m\}$



Searchable Encryption

- Index-based solutions
- Files $\mathbf{f} = \{f_1, f_2, \dots, f_n\}$
- Dictionary $W = \{w_1, w_2, \dots, w_m\}$
- *For each keyword w_i in dictionary W :*
 - $F_{w_i} = \{\text{identifiers of all files containing } w_i\}$



Searchable Encryption

- Index-based solutions

- Files $\mathbf{f} = \{f_1, f_2, \dots, f_n\}$

- Dictionary $W = \{w_1, w_2, \dots, w_m\}$



- *For each keyword w_i in dictionary W :*

- $F_{w_i} = \{\text{identifiers of all files containing } w_i\}$
- Generate a key $K_{w_i} = F(K, w_i)$ ← Pseudo Random Function
- Encrypt F_{w_i} under K_{w_i}

Searchable Encryption

- Index-based solutions

- Files $\mathbf{f} = \{f_1, f_2, \dots, f_n\}$

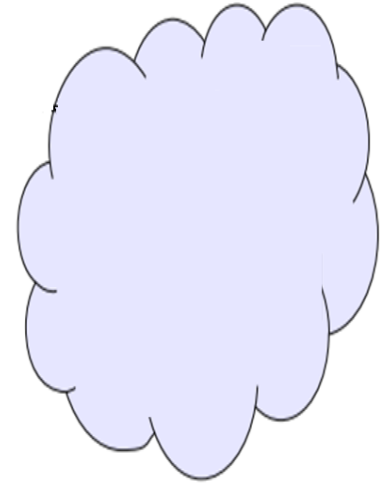
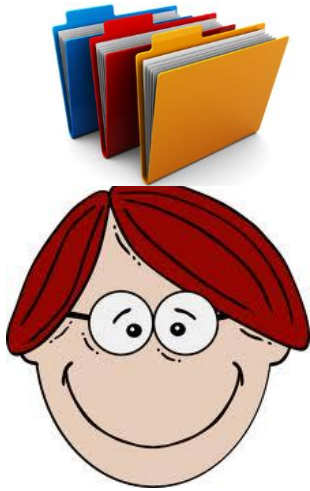
- Dictionary $W = \{w_1, w_2, \dots, w_m\}$



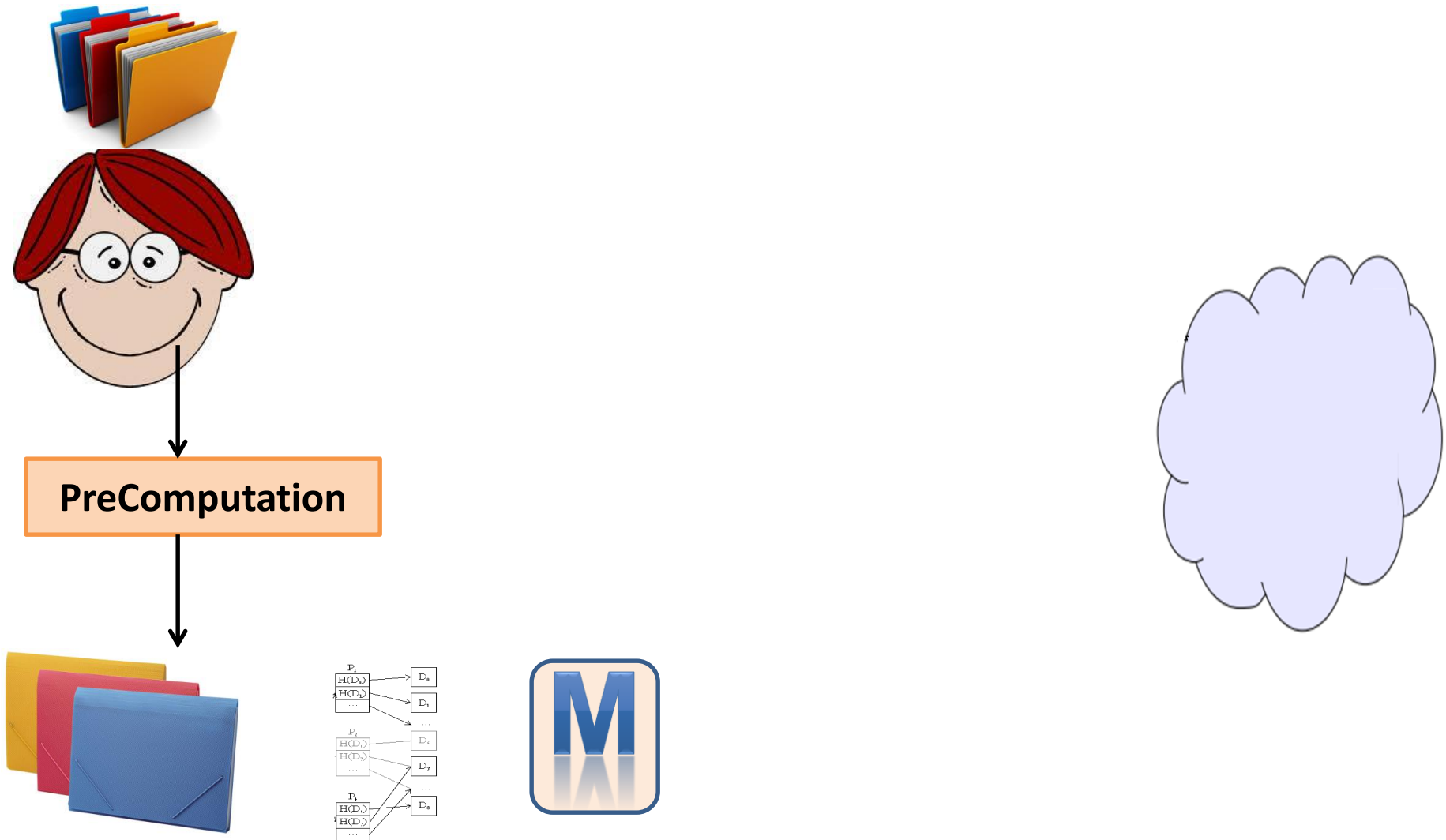
- *For each keyword w_i in dictionary W :*

- $F_{w_i} = \{\text{identifiers of all files containing } w_i\}$
- Generate a key $K_{w_i} = F(K, w_i)$ ← Pseudo Random Function
- Encrypt F_{w_i} under K_{w_i}
- Store them at (random) locations in the index
- Outsource the encrypted index together with the encrypted files

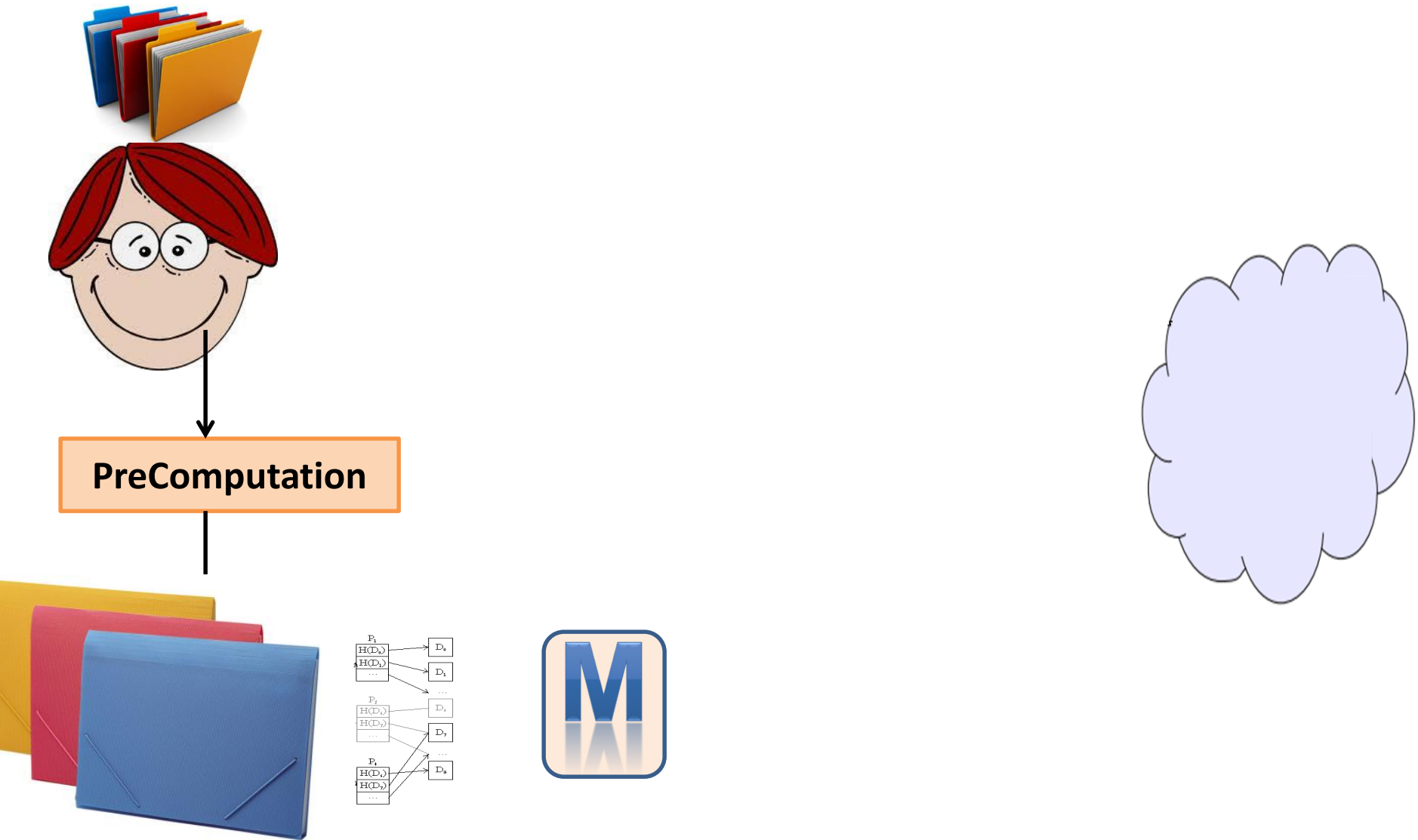
Searchable Encryption



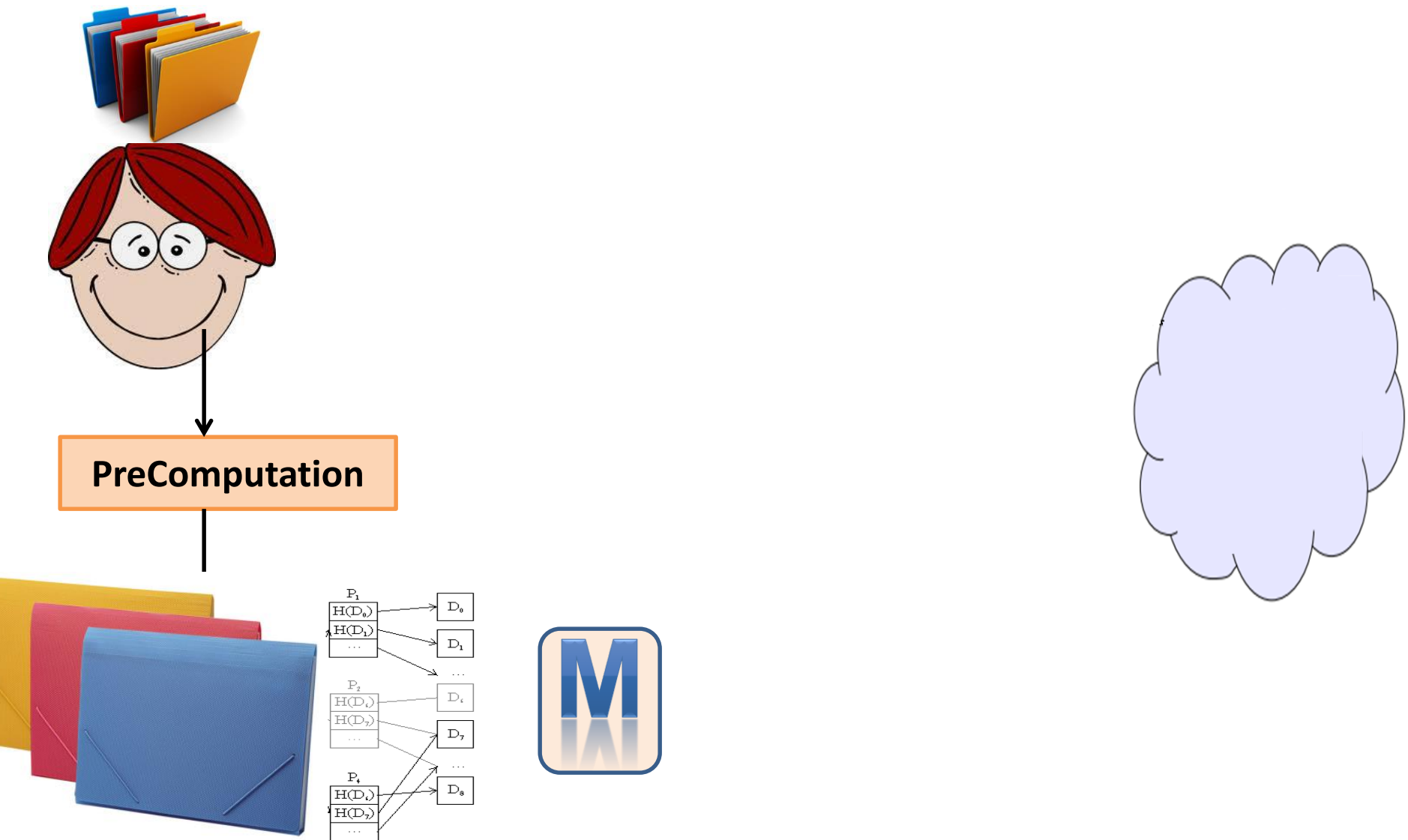
Searchable Encryption



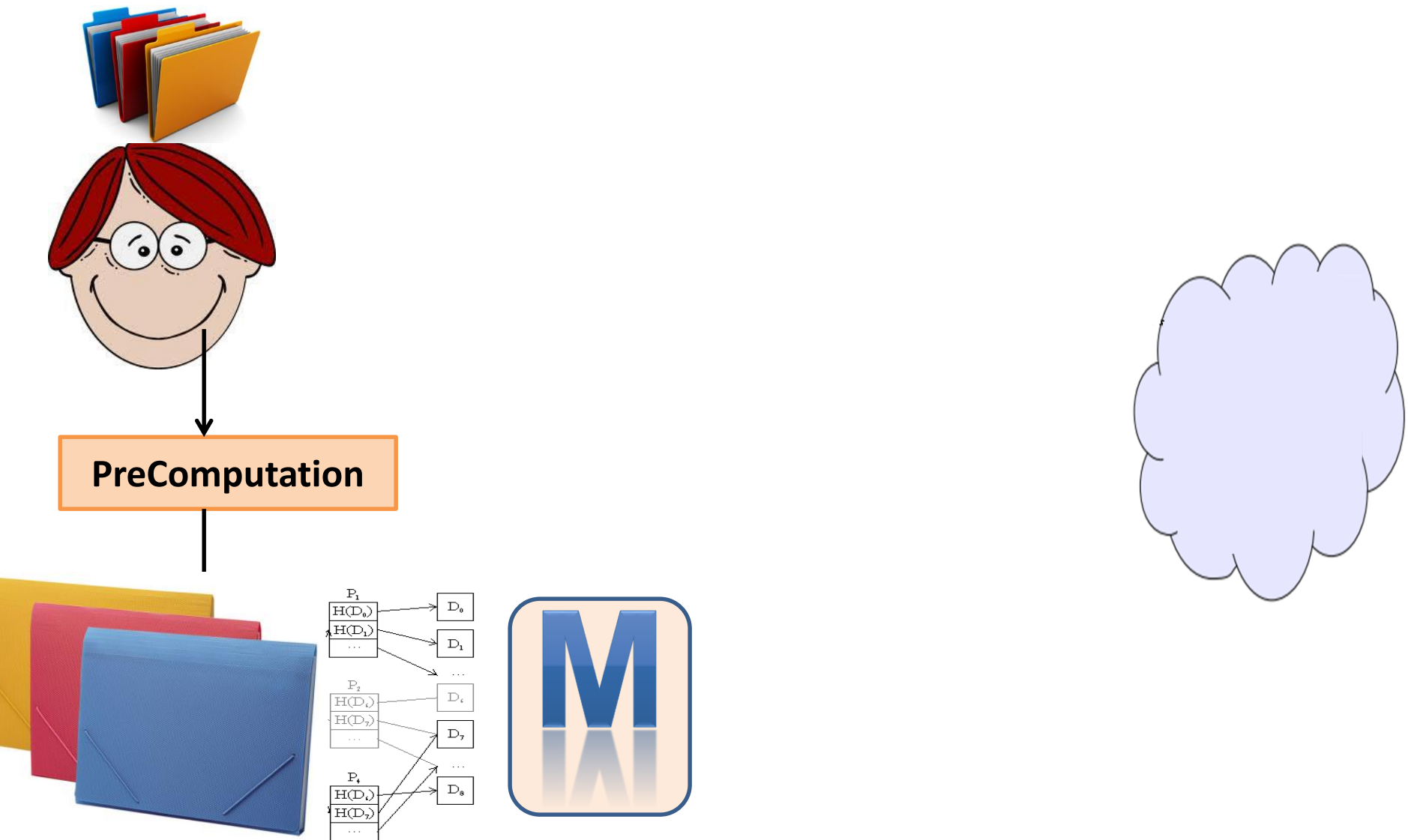
Searchable Encryption



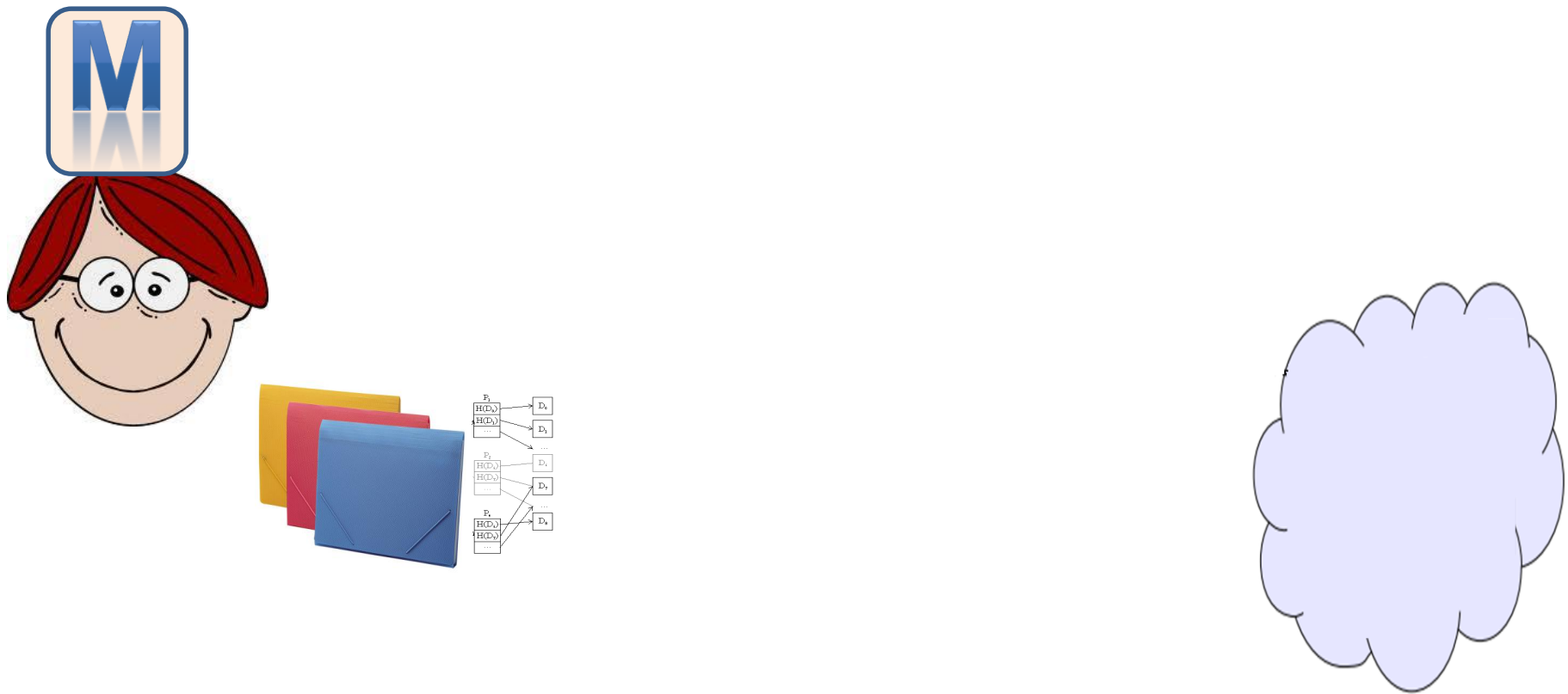
Searchable Encryption



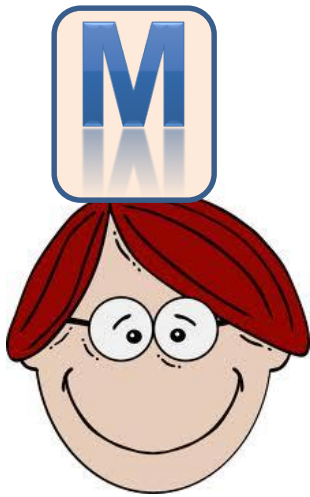
Searchable Encryption



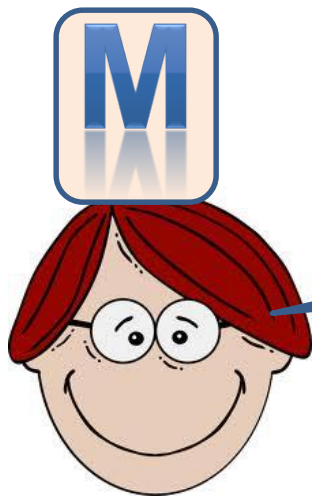
Searchable Encryption



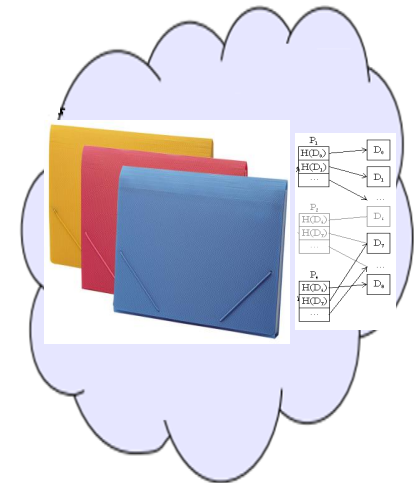
Searchable Encryption



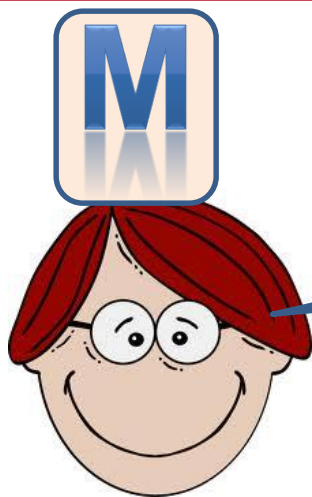
Searchable Encryption



Retrieve the files
containing a keyword w_i

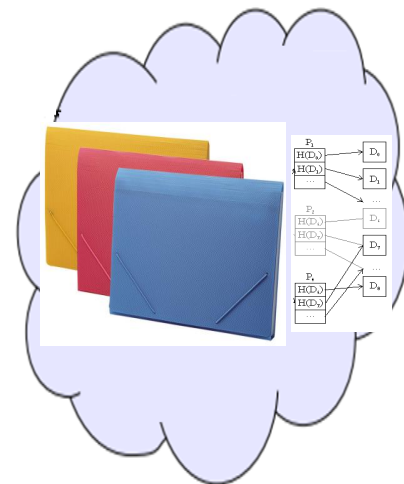


Searchable Encryption

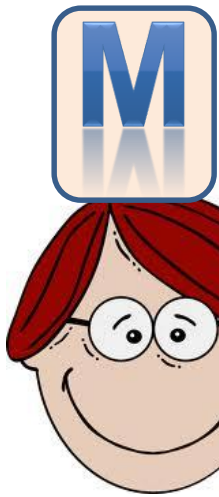


Retrieve the files
containing a keyword w_i

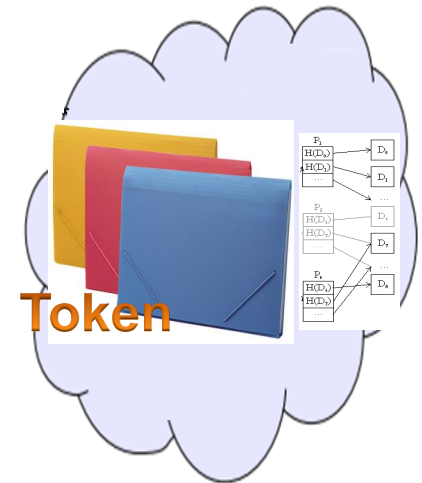
$w_i \longrightarrow$ Token



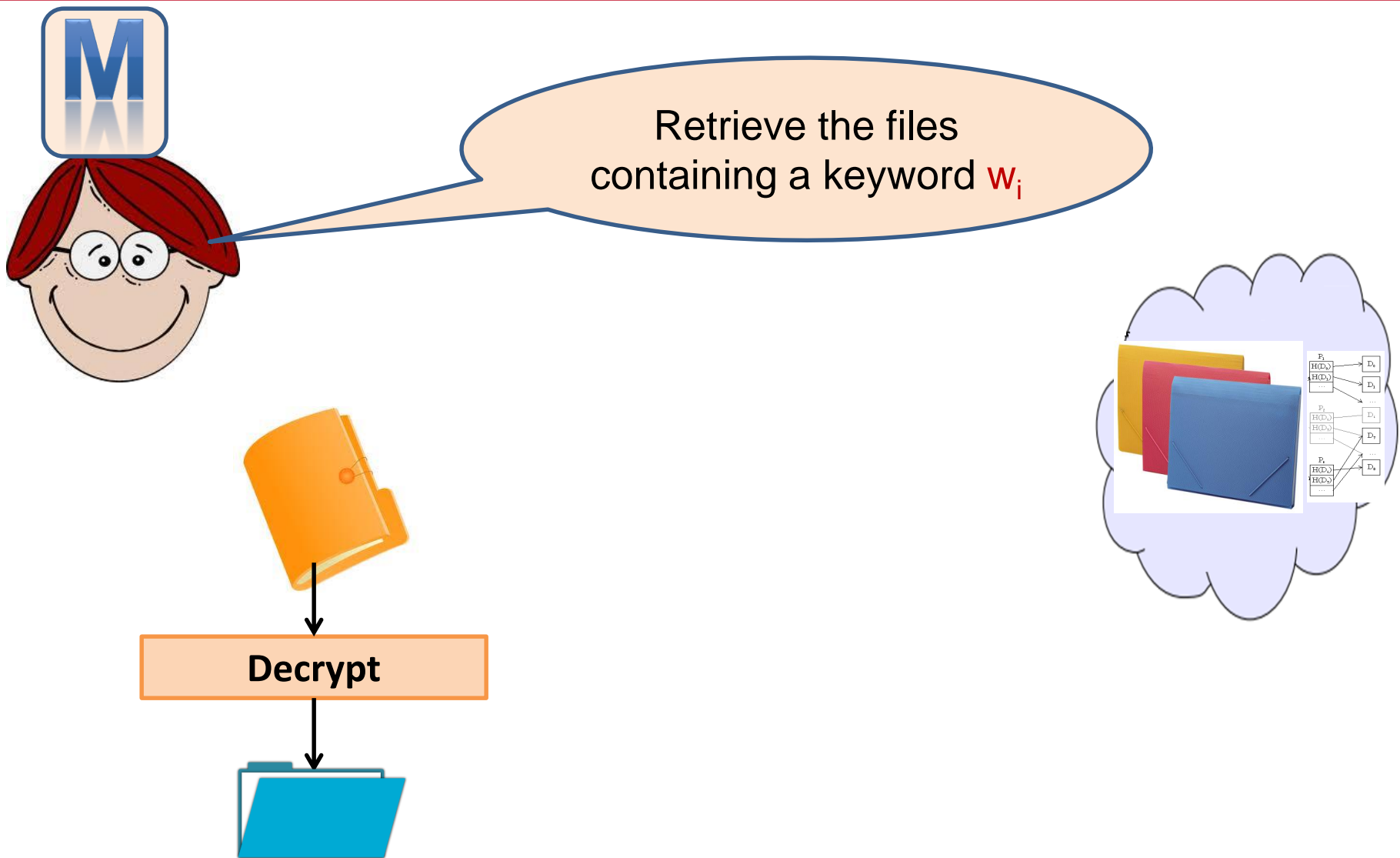
Searchable Encryption



Retrieve the files
containing a keyword w_i



Searchable Encryption



The Leakages

- **Search** leakage
 - The set of encrypted files containing w_i (**Access pattern**: $\mathbf{f}_{w_i,t}$)
 - Needed for **efficient** response
 - Server does **not** know the keyword or the contents of files!

The Leakages

- **Search** leakage
 - The set of encrypted files containing w_i (**Access pattern**: $\mathbf{f}_{w_i,t}$)
 - Needed for **efficient** response
 - Server does **not** know the keyword or the contents of files!
 - How many times a keyword is searched for (**Search pattern**: SP)
 - The tokens are **deterministic**!

$$\mathcal{L}_{Srch}(w_i, t) = \{\mathbf{f}_{w_i,t}, SP\}$$

The Leakages

- File **Insertion** leakage (for dynamic schemes **without** forward privacy)
 - File identifier (e_j)
 - File size ($|f_j|$)

The Leakages

- File **Insertion** leakage (for dynamic schemes **without** forward privacy)
 - File identifier (e_j)
 - File size ($|f_j|$)
 - Number of keywords in the file **and if any of them was previously queried**
 - They are encrypted under a key that is already revealed to the server.
 - **If all keywords of a new file have already been queried, the server knows all its (encrypted) keywords upon insertion!**

$$\mathcal{L}_{Add}(f_j) = (e_j, |f_j|, |\{w_i\}_{w_i \in f_j}|, \boxed{\{w_i\}_{w_i \text{ is queried}}})$$

Exploiting Leakage

- The leakages can be used to compromise confidentiality of the data and queries
 - Access pattern attacks [IKK12, NKW15, CGPR15]
 - Search pattern attacks [LZWT14]
 - File injection attacks [ZKP16]

Exploiting Leakage

- The leakages can be used to compromise confidentiality of the data and queries
 - Access pattern attacks [IKK12, NKW15, CGPR15]
 - Search pattern attacks [LZWT14]
 - File injection attacks [ZKP16]
- Without forward privacy, the server can link a new file to the previously queried keywords upon insertion for free!

Exploiting Leakage

- The leakages can be used to compromise confidentiality of the data and queries
 - Access pattern attacks [IKK12, NKW15, CGPR15]
 - Search pattern attacks [LZWT14]
 - File injection attacks [ZKP16]
- Without forward privacy, the server can link a new file to the previously queried keywords upon insertion for free!
- **Forward privacy** prevents this leakage.
 - Makes adaptive injection attacks less effective [ZKP16].

Forward Privacy

- With forward privacy, the **insertion leakage** is limited to:
 - File identifier
 - File size
 - Number of keywords in the file ~~and if any of them was previously queried~~

$$\mathcal{L}_{Add}(f_j) = (e_j, |f_j|, |\{w_i\}_{w_i \in f_j}|)$$

- The server cannot link the new file to the previous searches

Our Scheme

- Upon a search:
 - Client reveals the respective key to the server,
 - Server deletes all accessed index entries,
 - Client re-inserts them encrypted under a **fresh key** at **new random locations** in the index.

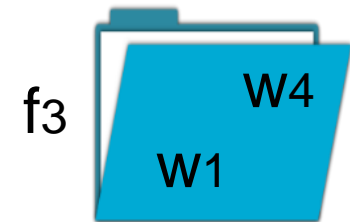
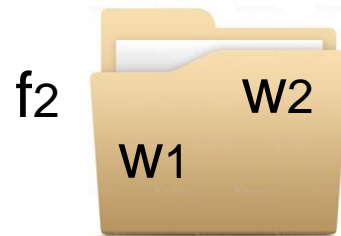
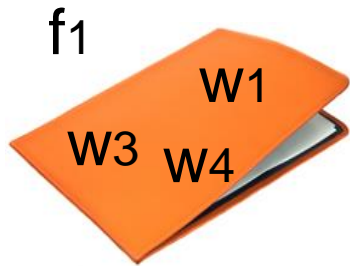
Our Scheme

- Upon a search:
 - Client reveals the respective key to the server,
 - Server deletes all accessed index entries,
 - Client re-inserts them encrypted under a **fresh key** at **new random locations** in the index.
- Slides:
 - Honest-but-curious server
 - Small but non-constant client storage

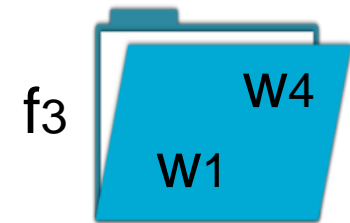
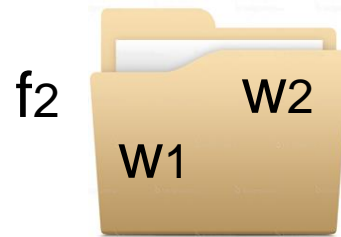
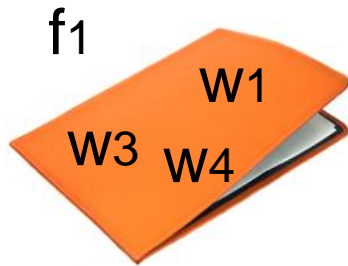
Our Scheme

- Upon a search:
 - Client reveals the respective key to the server,
 - Server deletes all accessed index entries,
 - Client re-inserts them encrypted under a **fresh key** at **new random locations** in the index.
- Slides:
 - Honest-but-curious server
 - Small but non-constant client storage
- Paper:
 - Dynamic, efficient, parallelizable, forward-private, simulation-secure

Our Scheme



Our Scheme



- $W = \{w_1, w_2, w_3, w_4\}$
- (w_i, f_j) : f_j contains w_i .

Server Side

[illegible]

TW

A diagram of a book with an orange cover. The cover has three labels: f_1 in the top left, w_3 in the bottom left, and w_4 in the bottom center. A blue circle highlights the label w_1 in the top right corner of the cover.

MW[illegible]

A diagram of a book with an orange cover. The cover has three labels: f_1 in the top left, w_3 in the bottom left, and w_4 in the bottom center. A blue circle highlights the label w_1 in the top right corner of the cover.

s

w1
w2
w3
w4

TW

[illegible]

A diagram of a book with an orange cover. The cover has three labels: f_1 in the top left, w_3 in the bottom left, and w_4 in the bottom center. A blue circle highlights the label w_1 in the top right corner of the cover.

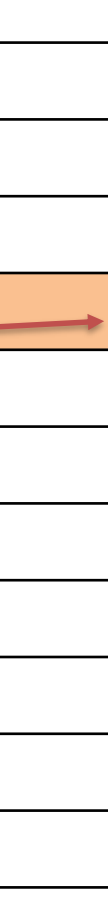
$$A_1 = F(K_{w1}, 1, 0)$$

MW[illegible]

A diagram of a book with an orange cover. The cover has three labels: f_1 in the top left, w_3 in the bottom left, and w_4 in the bottom center. A blue circle highlights the label w_1 in the top right corner of the cover.

$$A_1 = F(K_{w1}, 1, 0)$$

Encryption key

MW

$(w1, f1)$

Diagram illustrating a stack of three orange books. The top book is labeled f_1 . The middle book is labeled W_1 . The bottom book is labeled W_3 and W_4 , with W_3 circled in blue.

MW[illegible]

A diagram of a book with an orange cover. The cover has three labels: f_1 at the top left, W_1 at the top right, and W_4 at the bottom right. A blue circle highlights the label W_3 on the left side of the cover.

#

w1
w2
w3
w4

TW

(w1, f1)

$$A_3 = F(K_{w3}, 1, 0)$$

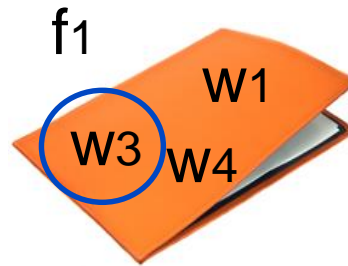
Address

~~TW~~

[illegible]

Build

Adding



$$K_{w3} = F(K, w3, 0)$$

$$A3 = F(K_{w3}, 1, 0)$$

$$K3 = F(K_{w3}, 1, 1)$$

	# files	# searches
w1	1	0
w2	0	0
w3	1	0
w4	0	0

MW

Address

TW

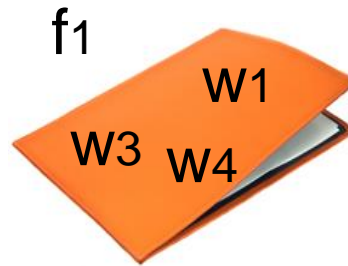
Encryption key



(w1, f1)
(w3, f1)

Build

After adding



	# files	# searches
w1	1	0
w2	0	0
w3	1	0
w4	1	0

MW

(w4, f1)
(w1, f1)
(w3, f1)

TW

Build

After adding



	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW

(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

TW

Search

Searching for w4

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)

TW
(Server)

(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

Search

Searching for w4

$$K_{w4} = F(K, w4, 0)$$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)

(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

TW
(Server)

Search

Searching for w_4

$$K_{w_4} = F(K, w_4, 0)$$

$$n_w = 2$$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)

TW
(Server)

(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

Search

Searching for w_4

$$K_{w_4} = F(K, w_4, 0)$$

$$n_w = 2$$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)

TW
(Server)

(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

Search

The server:

```
for i = 1 to nw
   $A_i = F(K_{w4}, i, 0)$ 
   $K_i = F(K_{w4}, i, 1)$ 
```

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)

TW
(Server)

(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

Search

The server:

```
for i = 1 to nw
   $A_i = F(K_{w4}, i, 0)$ 
   $K_i = F(K_{w4}, i, 1)$ 
```

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)



(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

TW
(Server)


Search

The server:

```
for i = 1 to nw
   $A_i = F(K_{w4}, i, 0)$ 
   $K_i = F(K_{w4}, i, 1)$ 
```

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)



(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

TW
(Server)



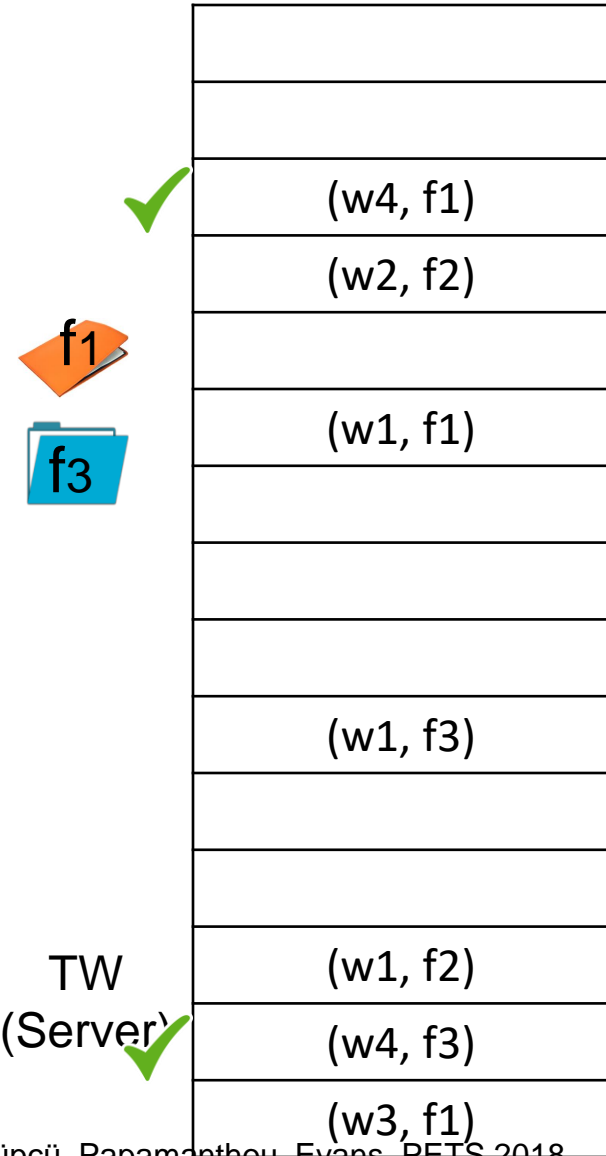
Search

The server:

```
for i = 1 to nw
   $A_i = F(K_{w4}, i, 0)$ 
   $K_i = F(K_{w4}, i, 1)$ 
```

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)



(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

TW
(Server)

Search

The server:

for $i = 1$ to n_w
 $A_i = F(K_{w4}, i, 0)$
 $K_i = F(K_{w4}, i, 1)$



	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)



(w4, f1)
(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w4, f3)
(w3, f1)

TW
(Server)



Search

The server removes the found entries from the index.

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	0

MW (Client)

TW
(Server)

(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w3, f1)

Search

The client

$$K_{w4} = F(K, w4, 1)$$

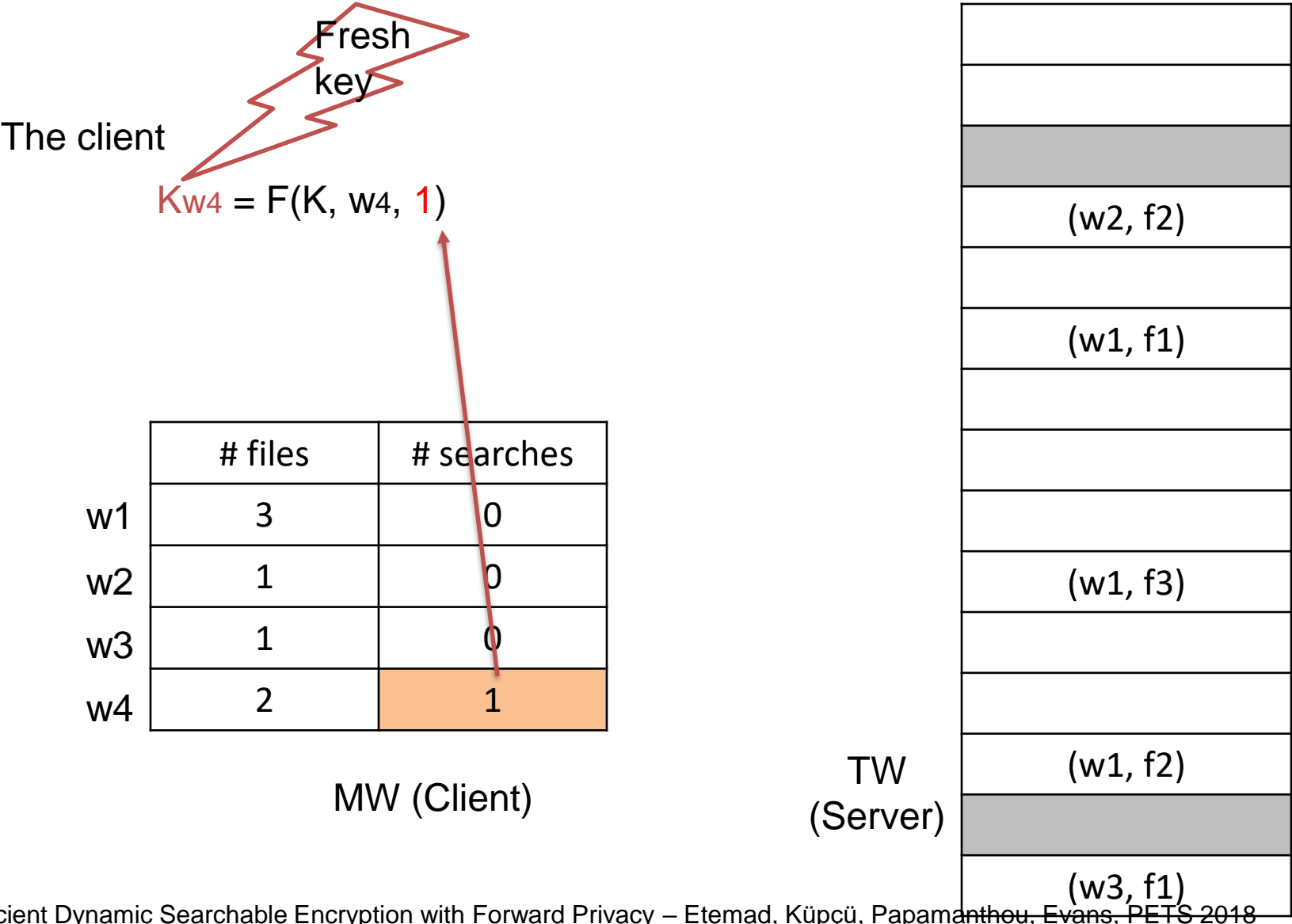
	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

TW
(Server)

(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w3, f1)

Search



Search

The client

$$K_{w4} = F(K, w4, 1)$$

for $i = 1$ to n_w

$$A_i = F(K_{w4}, i, 0)$$

$$K_i = F(K_{w4}, i, 1)$$



$(w4, f1)$ and $(w4, f3)$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

TW
(Server)

(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w3, f1)

Search

The client

$$K_{w4} = F(K, w4, 1)$$

for $i = 1$ to n_w

$$A_i = F(K_{w4}, i, 0)$$

$$K_i = F(K_{w4}, i, 1)$$



$(w4, f1)$ and $(w4, f3)$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)



TW
(Server)

(w2, f2)
(w1, f1)
(w1, f3)
(w1, f2)
(w3, f1)

Search

After searching for w4

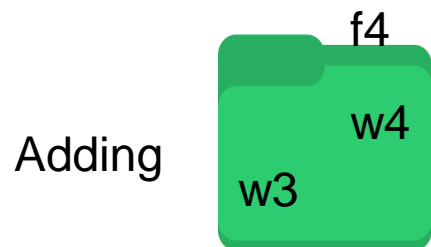
	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

TW
(Server)

(w2, f2)
(w1, f1)
(w4, f1)
(w1, f3)
(w4, f3)
(w1, f2)
(w3, f1)

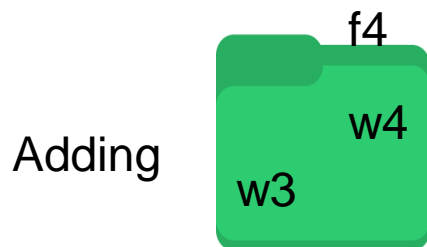
File Insertion



	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

File Insertion



	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

$$K_{w3} = F(K, w3, 0)$$

$$K_{w4} = F(K, w4, 1)$$

MW (Client)


File Insertion




	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

$$K_{w3} = F(K, w3, 0)$$

$$K_{w4} = F(K, w4, 1)$$

(w3, f4) 

(w4, f4) 

MW (Client)

File Insertion

After adding



TW
(Server)

(w4, f4)
(w2, f2)
(w1, f1)
(w3, f4)
(w4, f1)
(w1, f3)
(w4, f3)
(w1, f2)
(w3, f1)

File Insertion

After adding



Each entry is always encrypted with a key that is not known to the server.

TW
(Server)

(w4, f4)
(w2, f2)
(w1, f1)
(w3, f4)
(w4, f1)
(w1, f3)
(w4, f3)
(w1, f2)
(w3, f1)

File Insertion

After adding



Each entry is always encrypted with a key that is not known to the server.

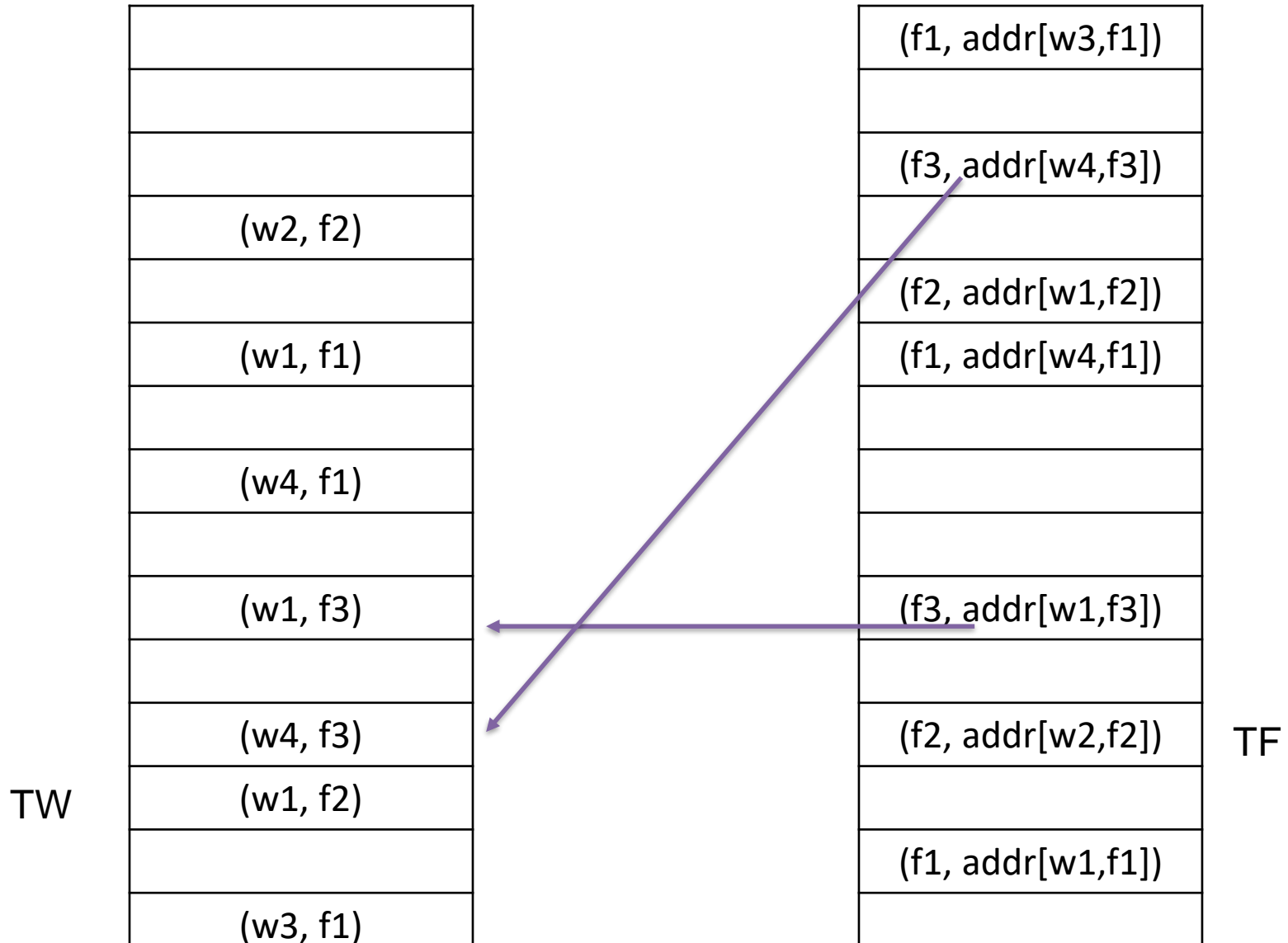
	# files	
w1	3	0
w2	1	0
w3	2	0
w4	3	1

MW (Client)

(w4, f4)
(w2, f2)
(w1, f1)
(w3, f4)
(w4, f1)
(w1, f3)
(w4, f3)
(w1, f2)
(w3, f1)

TW
(Server)

File Deletion - Indexes



File Deletion - Indexes

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

	# keywords
f1	3
f2	2
f3	2

MF (Client)

File Deletion - Client

Deleting f3

$$K_{f3} = F(K, f3)$$

$$n_{f3} = 2$$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

	# keywords
f1	3
f2	2
f3	2

MF (Client)

File Deletion - Client

Deleting f3

$$K_{f3} = F(K, f3)$$
$$n_{f3} = 2$$

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

	# keywords
f1	3
f2	2
f3	2

MF (Client)

File Deletion - Server

The server:

```
for i = 1 to nf3
  Ai = F(Kf3, i, 0)
  Ki = F(Kf3, i, 1)
```

TW

(w2, f2)
(w1, f1)
(w4, f1)
(w1, f3)
(w4, f3)
(w1, f2)
(w3, f1)

(f1, addr[w3,f1])
(f3, addr[w4,f3])
(f2, addr[w1,f2])
(f1, addr[w4,f1])
(f3, addr[w1,f3])
(f2, addr[w2,f2])
(f1, addr[w1,f1])

TF

File Deletion - Server

The server:

for $i = 1$ to n_{f3}
 $A_i = F(K_{f3}, i, 0)$
 $K_i = F(K_{f3}, i, 1)$

TW

(w2, f2)
(w1, f1)
(w4, f1)
(w1, f3)
(w4, f3)
(w1, f2)
(w3, f1)



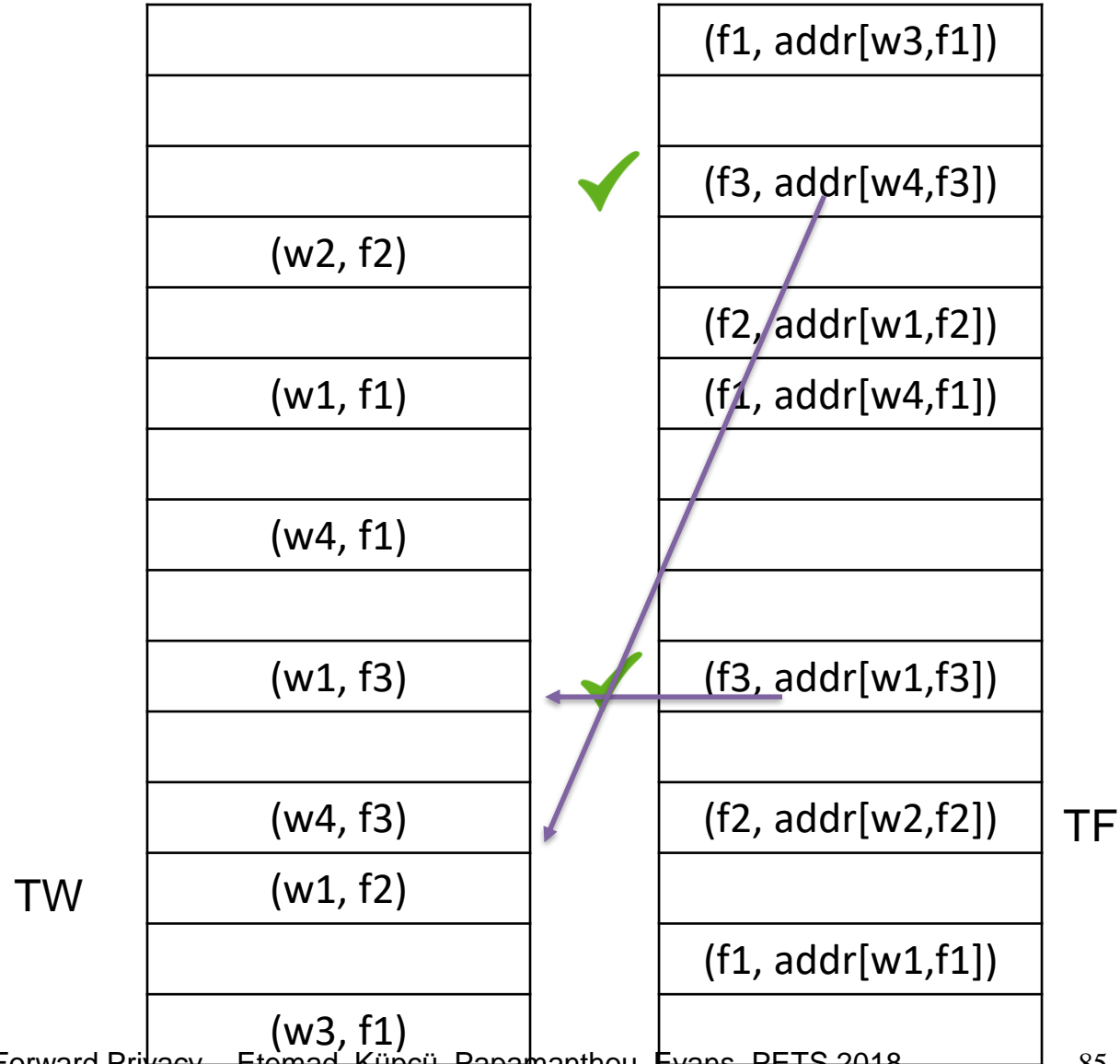
(f1, addr[w3,f1])
(f3, addr[w4,f3])
(f2, addr[w1,f2])
(f1, addr[w4,f1])
(f3, addr[w1,f3])
(f2, addr[w2,f2])
(f1, addr[w1,f1])

TF

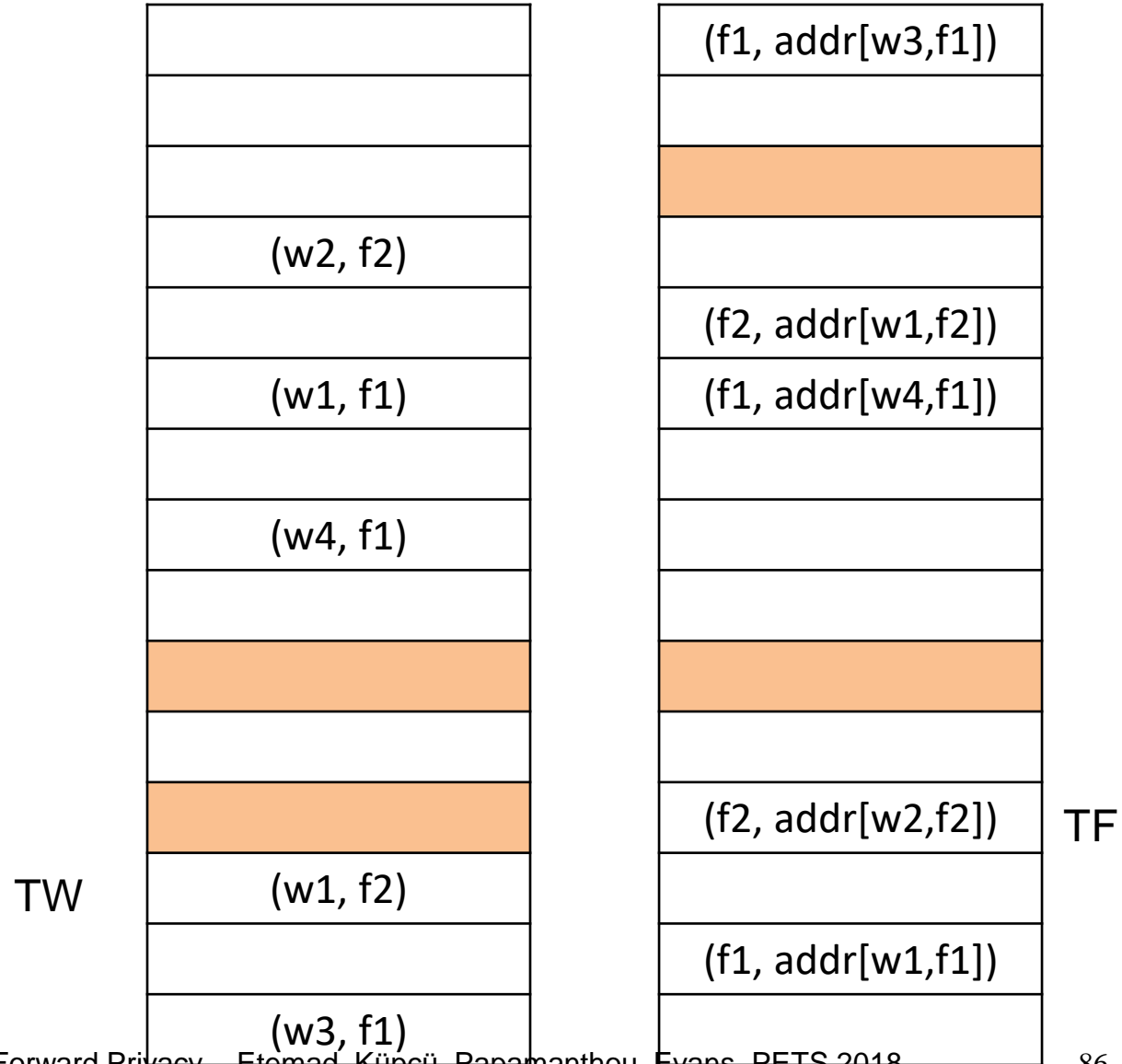
File Deletion - Server

The server:

for $i = 1$ to n_{f3}
 $A_i = F(K_{f3}, i, 0)$
 $K_i = F(K_{f3}, i, 1)$



File Deletion - Server



File Deletion - Client

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

	# keywords
f1	3
f2	2

MF (Client)

File Deletion - Client

This will be updated with searches and insertions!

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

	# keywords
f1	3
f2	2

MF (Client)

File Deletion - Client

This will be updated with searches and insertions!

	# files	# searches
w1	3	0
w2	1	0
w3	1	0
w4	2	1

MW (Client)

This will be updated with insertions and deletions!

	# keywords
f1	3
f2	2

MF (Client)

Related work

- [SPS14] uses an ORAM-based data structure as the index.
 - Search cost: $O(d \log^3 N)$
 - Update cost: $O(r \log^2 N)$
- Sophos ($\Sigma\phi\phi\varsigma$) [B16] uses public key operations.

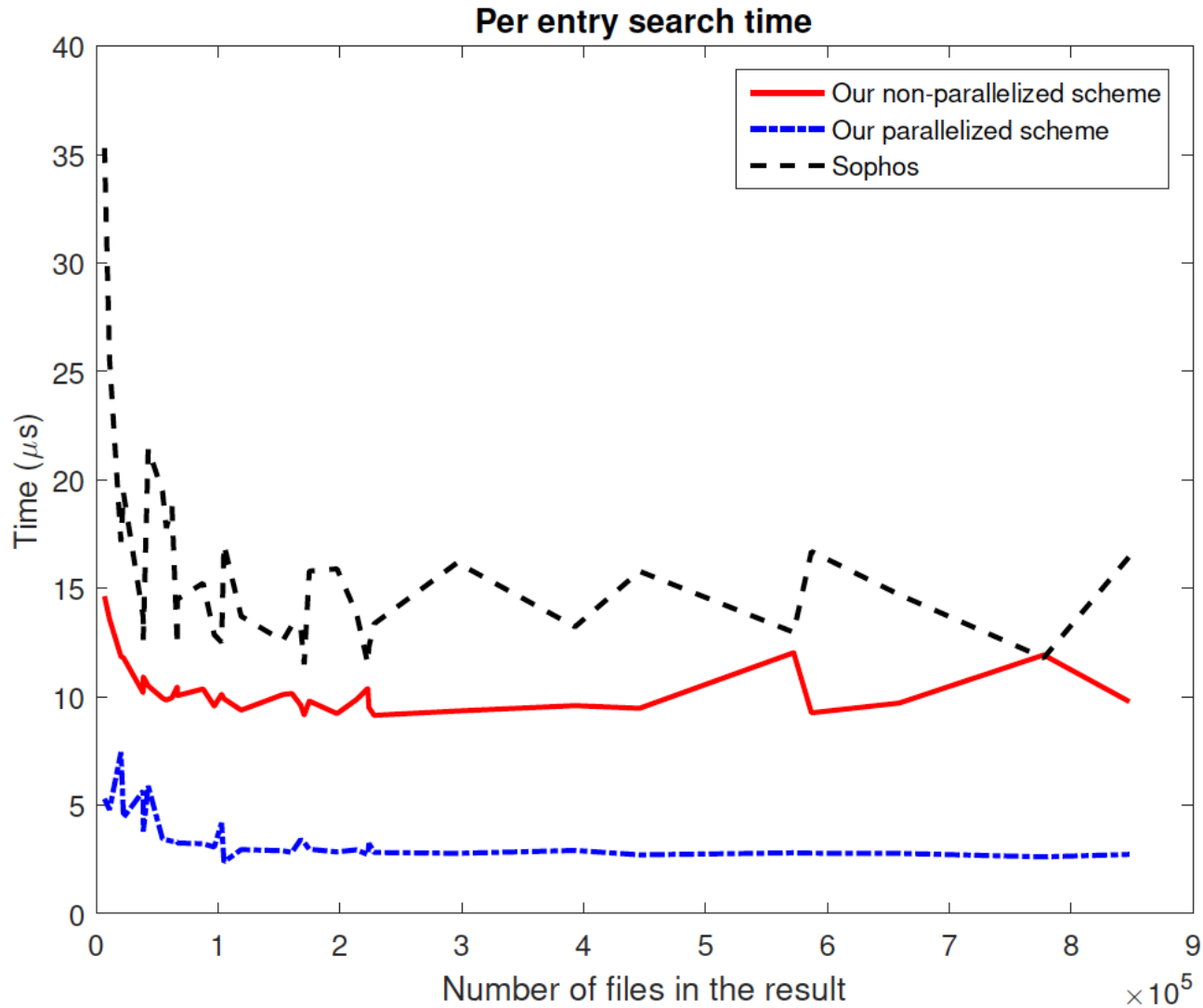
Scheme	Client storage	Server storage	Search cost	Update cost	Parallelism	Forward privacy
Practical SSE [27]	$O(\sqrt{N})$	$O(N)$	$O(d \log^3 N)$	$O(r \log^2 N)$	×	✓
Sophos [6]	$O(m)$	$O(N)$	$O(d + n_{ad})$	$O(r)$	×	✓
Ours	$O(m + n)^*$	$O(N)$	$O((d + n_d)/p)$	$O(r/p)$	✓	✓

n and m denote the total number of files and keywords, respectively. d is the number of files containing a keyword, and r is the number of unique keywords in a file. The number of processors and (keyword, file identifier) mappings is p and N , respectively. n_{ad} and n_d show the number of times a keyword has been affected by file deletions since beginning and since the last search for the same keyword, respectively ($n_{ad} \geq n_d$). “*” indicates that we outsource the local index (indeed, $O(n)$ part can be outsourced without any effect on other asymptotic costs, as explained in Section 3.3).

Performance

- Dataset:
 - n : ~4M Wikipedia pages (files)
 - m : ~10M dictionary size (keywords)
 - N : ~500M total index entries (server side)
- Implementation: C/C++ with the Crypto++ library
 - SHA1
 - Indexes are implemented as C++ maps.
- Server: Amazon EC2 using m4.4xlarge instances (64GB of memory, 16 CPU cores) running Ubuntu 16.04 LTS.
 - Single core employed.
- Client: Apple MacBook Air Laptop
- Sophos and our scheme are run and compared.

Performance - Search



Conclusion

- Forward privacy reduces the leakage and makes the attacks less effective.

Conclusion

- Forward privacy reduces the leakage and makes the attacks less effective.
- Our scheme:
 - Achieves forward privacy
 - Is parallelizable
 - Is efficient (only PRFs, hash functions, and simple maps)
 - Has security proof via simulation

Conclusion

- Forward privacy reduces the leakage and makes the attacks less effective.
- Our scheme:
 - Achieves forward privacy
 - Is parallelizable
 - Is efficient (only PRFs, hash functions, and simple maps)
 - Has security proof via simulation
- Future Work:
 - Backward privacy
 - Remove any linkage between a deleted file and later searches.
 - Existing solutions require index rebuild.

Conclusion

- Forward privacy reduces the leakage and makes the attacks less effective.
- Our scheme:
 - Achieves forward privacy
 - Is parallelizable
 - Is efficient (only PRFs, hash functions, and simple maps)
 - Has security proof via simulation
- Future Work:
 - Backward privacy
 - Remove any linkage between a deleted file and later searches.
 - Existing solutions require index rebuild.
 - Reducing the client storage
 - From $O(m+n)$ without adding extra rounds.

Thank You

Questions?

etemad@virginia.edu

akupcu@ku.edu.tr

cpap@umd.edu

evans@virginia.edu

References

- [SWP00] D. Song, D. Wagner, A. Perrig, **Practical techniques for searches on encrypted data**, IEEE Security and Privacy, 2000.
- [KPR12] S. Kamara, C. Papamanthou, T. Roeder, **Dynamic Searchable Symmetric Encryption**, ACM CCS, 2012.
- [KP13] S. Kamara, C. Papamanthou, **Parallel and Dynamic Searchable Symmetric Encryption**, FC, 2013.
- [KO12] K. Kurosawa, Y. Ohtaki, **UC-secure searchable symmetric encryption**, FC, 2012.
- [IKK12] M. Islam, M. Kuzu, M. Kantarcioglu, **Access pattern disclosure on searchable encryption: Ramification, attack and mitigation**, NDSS 2012.
- [ZKP16] Y. Zhang, J. Katz, C. Papamanthou, **All your queries are belong to us: The power of file-injection attacks on searchable encryption**, USENIX Security, 2016.
- [NKW15] M. Naveed, S. Kamara, C. V. Wright, **Inference attacks on property-preserving encrypted databases**, ACM CCS 2015.
- [LZWT14] C. Liu, L. Zhu, M. Wang, Y.-a. Tan, **Search pattern leakage in searchable encryption: Attacks and new construction**, Information Sciences, 2014
- [CGPR15] D. Cash, P. Grubbs, J. Perry, T. Ristenpart, **Leakage abuse attacks against searchable encryption**, ACM CCS 2015
- [SPS14] E. Stefanov, C. Papamanthou, E. Shi, **Practical dynamic searchable encryption with small leakage**, NDSS 2014.
- [B16] R. Bost, **Sophos - forward secure searchable encryption**, ACM CCS 2016.