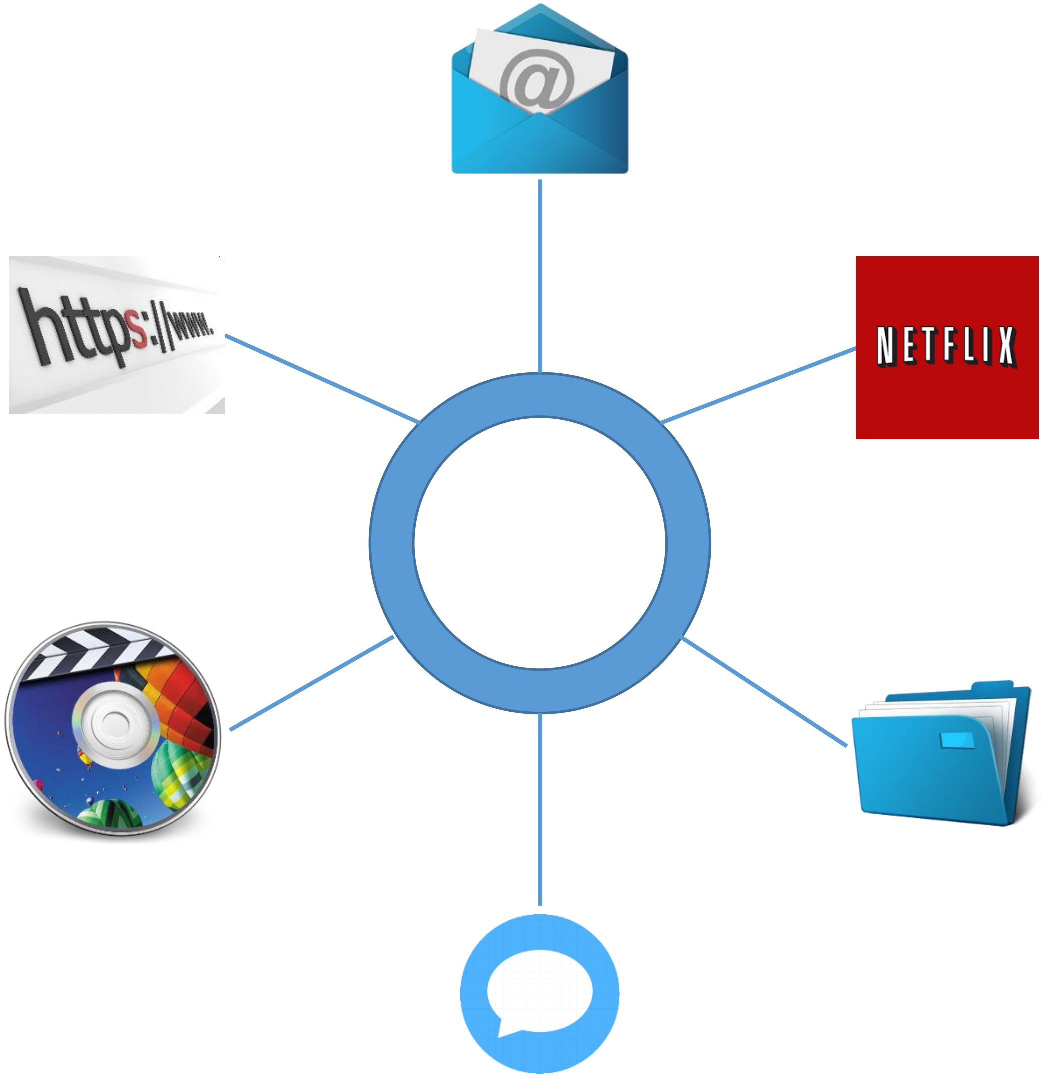
Metadata security

Mihai Ordean
Designing Secure Systems
University of Birmingham

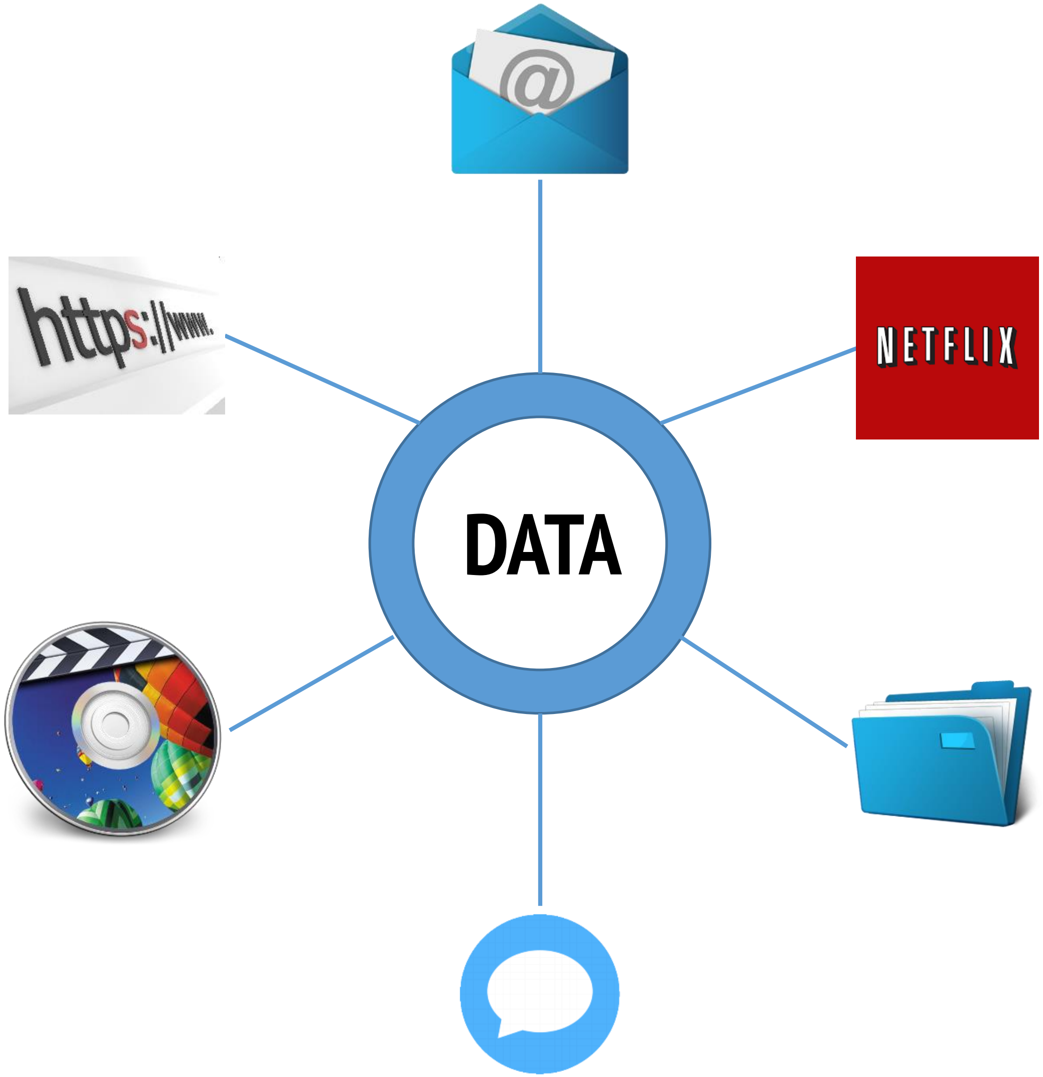
Overview

- Device security
    - Is code on the device vulnerable to exploits ? (e.g. buffer overflows)
    - Is the code authenticated ? (i.e. has not been tampered with)
- Data security (in the cloud)
    - Is the stored data is accessible to everyone? (e.g. encrypted)
    - Is the stored data authenticated?
- **Metadata security**
    - What does metadata reveal about data?
    - Can we tamper the metadata?
- Protocol security
    - Is data in transit visible?
    - Can data in transit be tampered with?

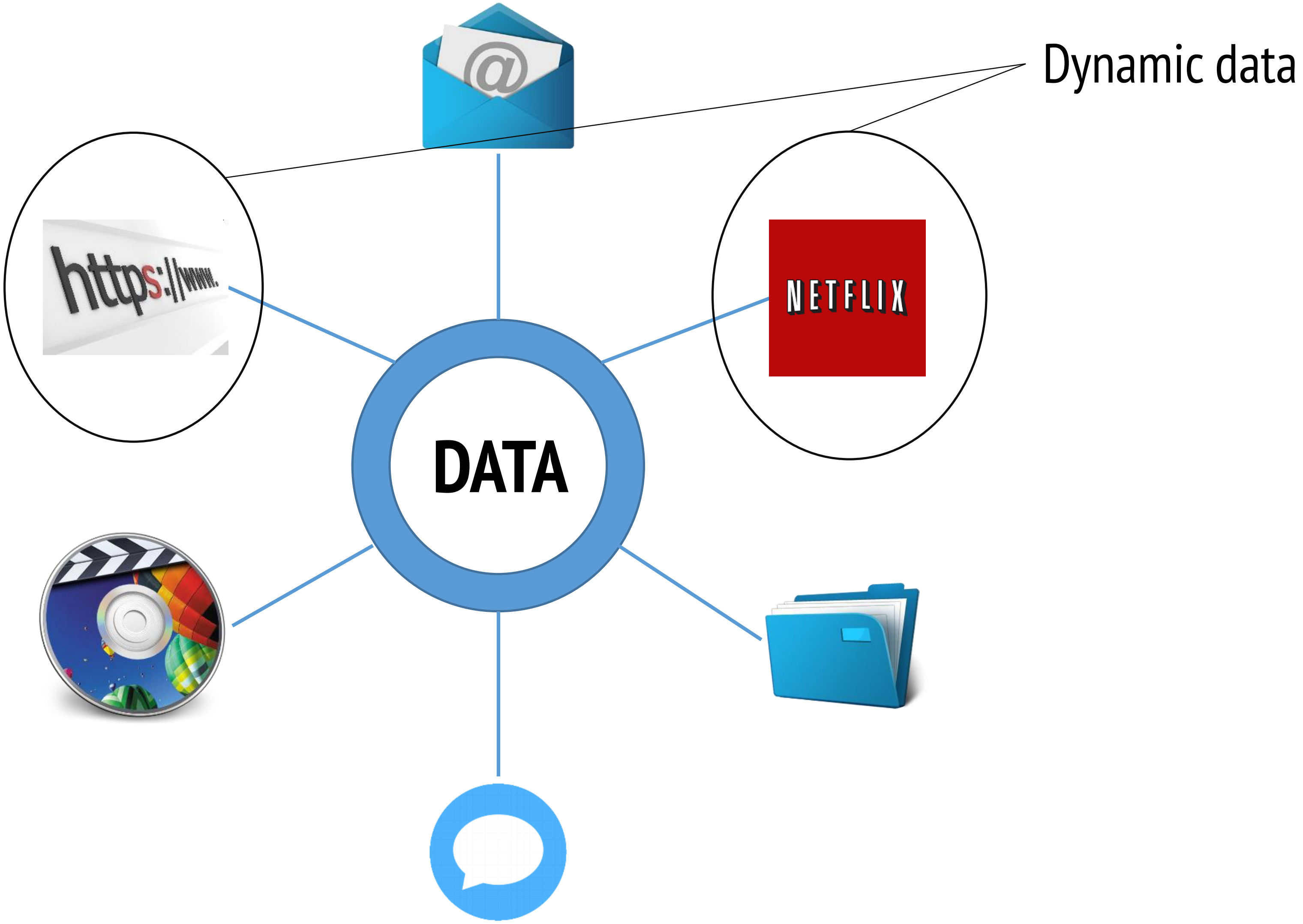
What can we encrypt?



What can we encrypt?



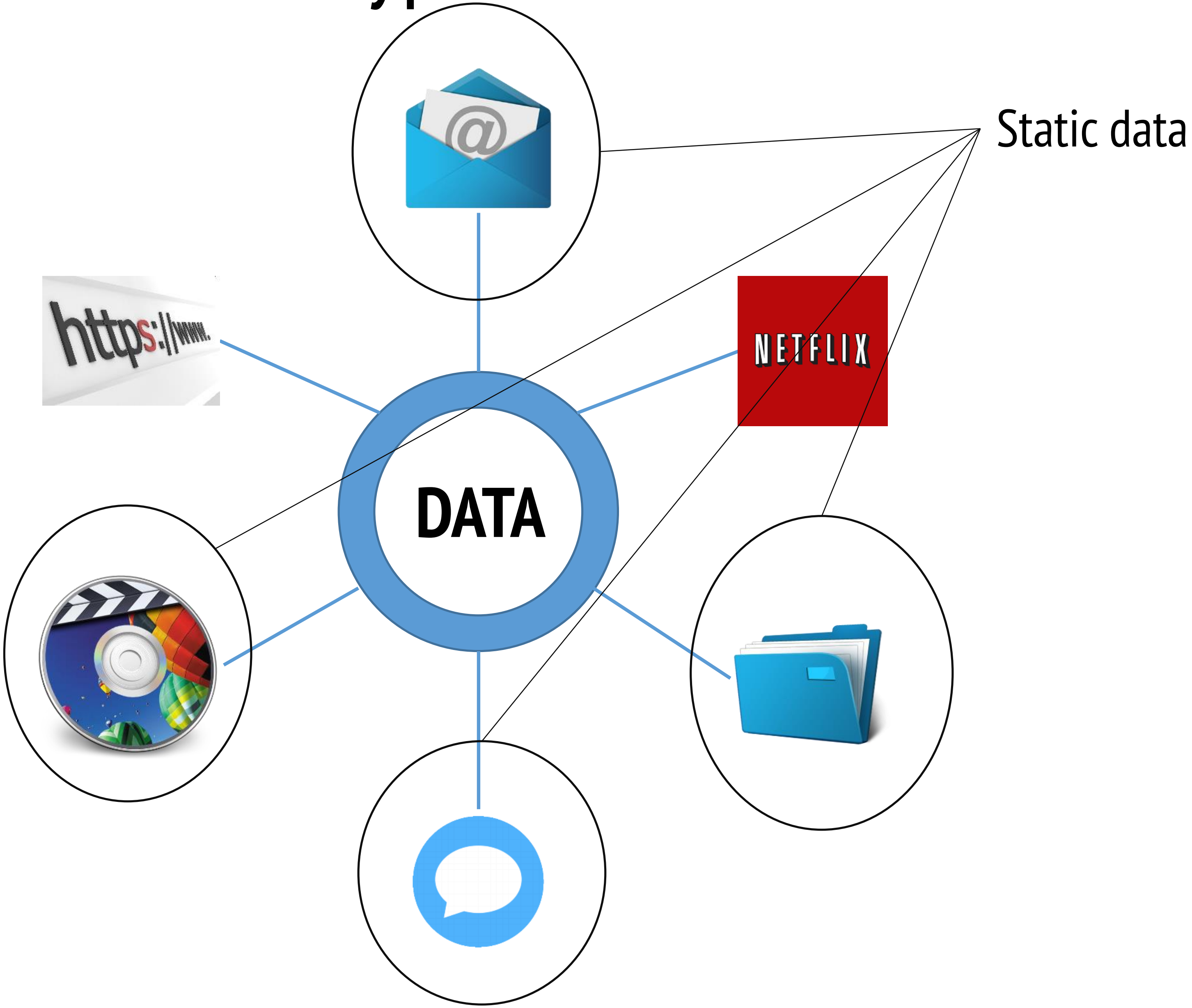
What can we encrypt?



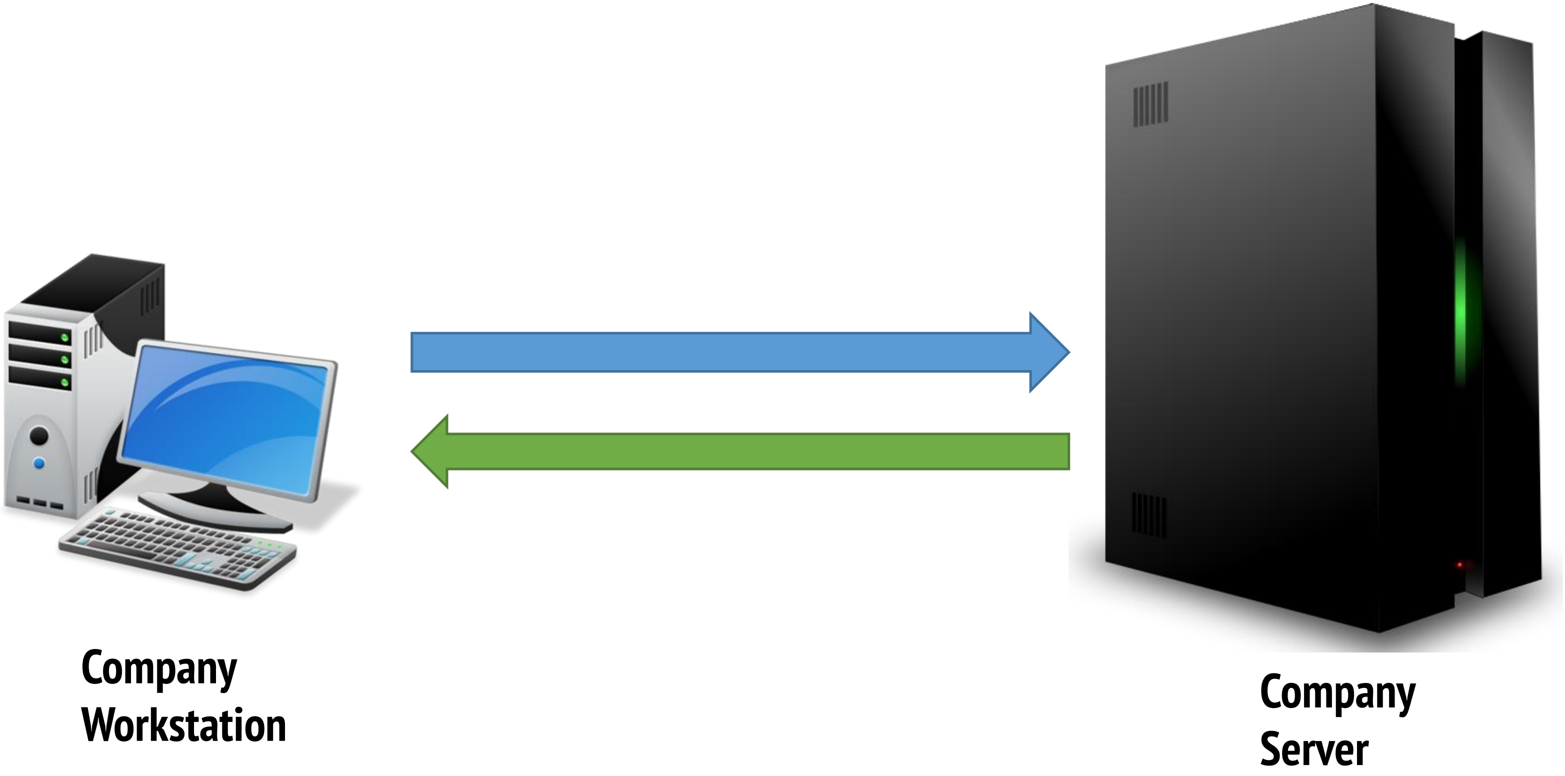
DATA

Dynamic data

What can we encrypt?



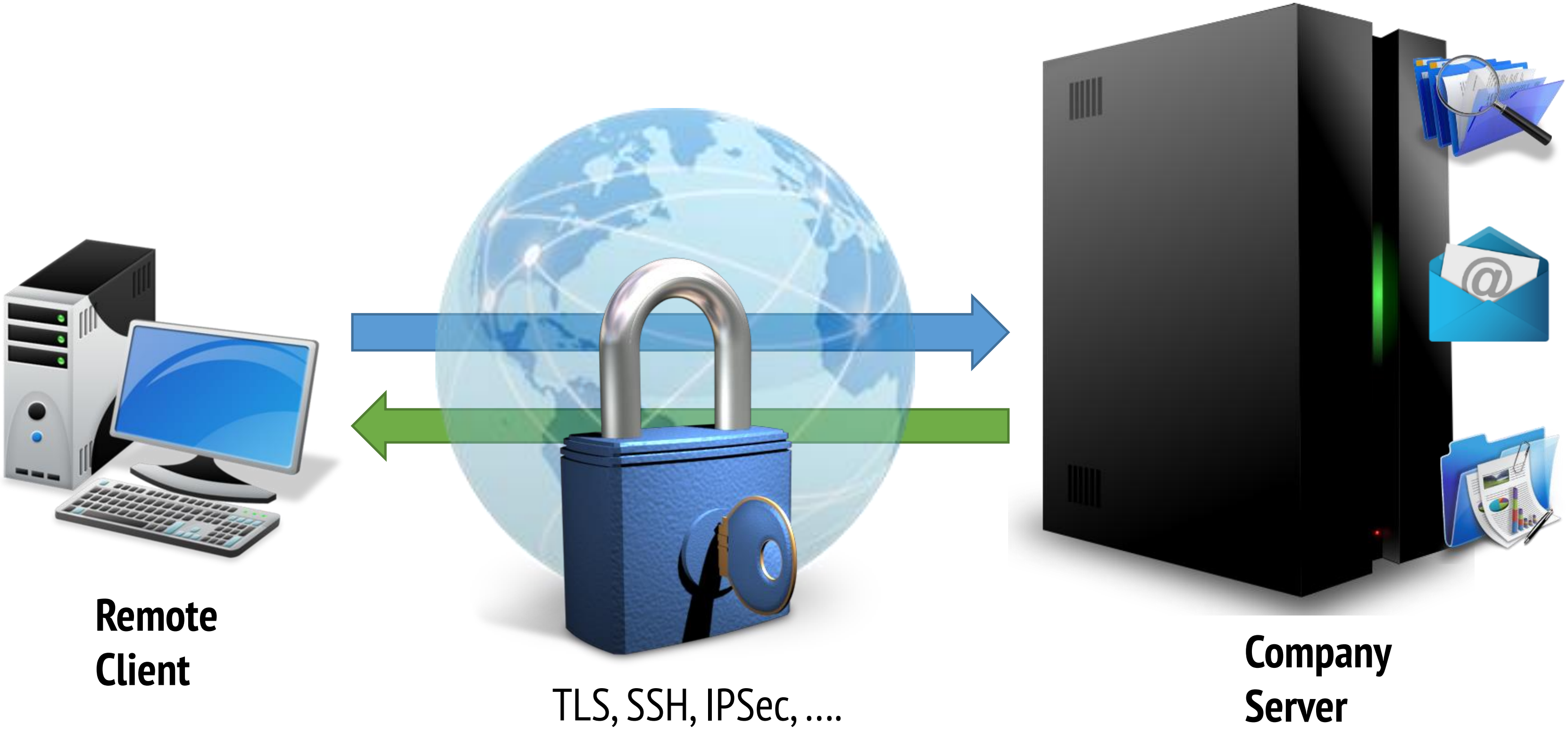
Protecting dynamic data



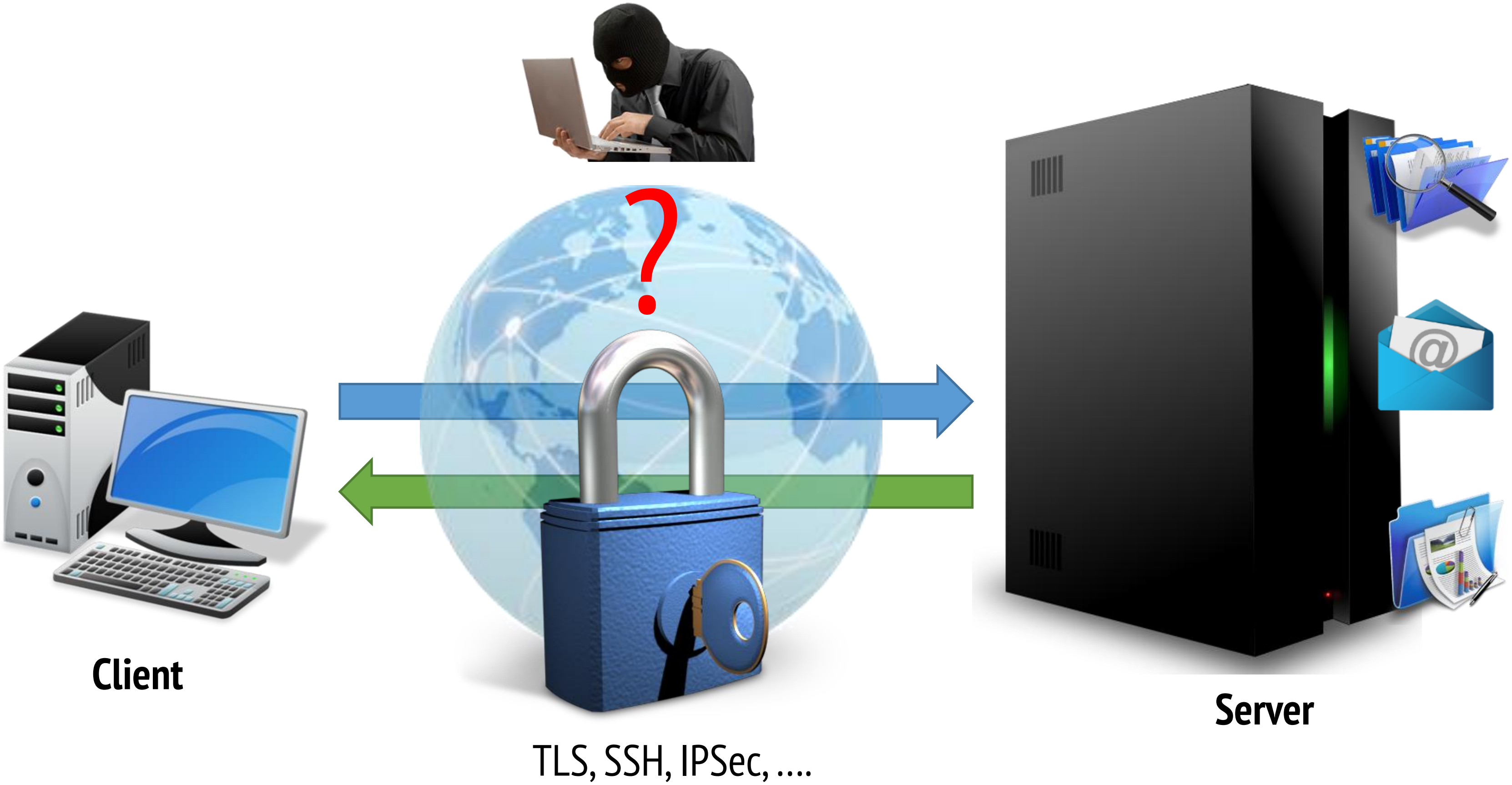
Company Workstation

Company Server

Protecting dynamic data



Protecting dynamic data



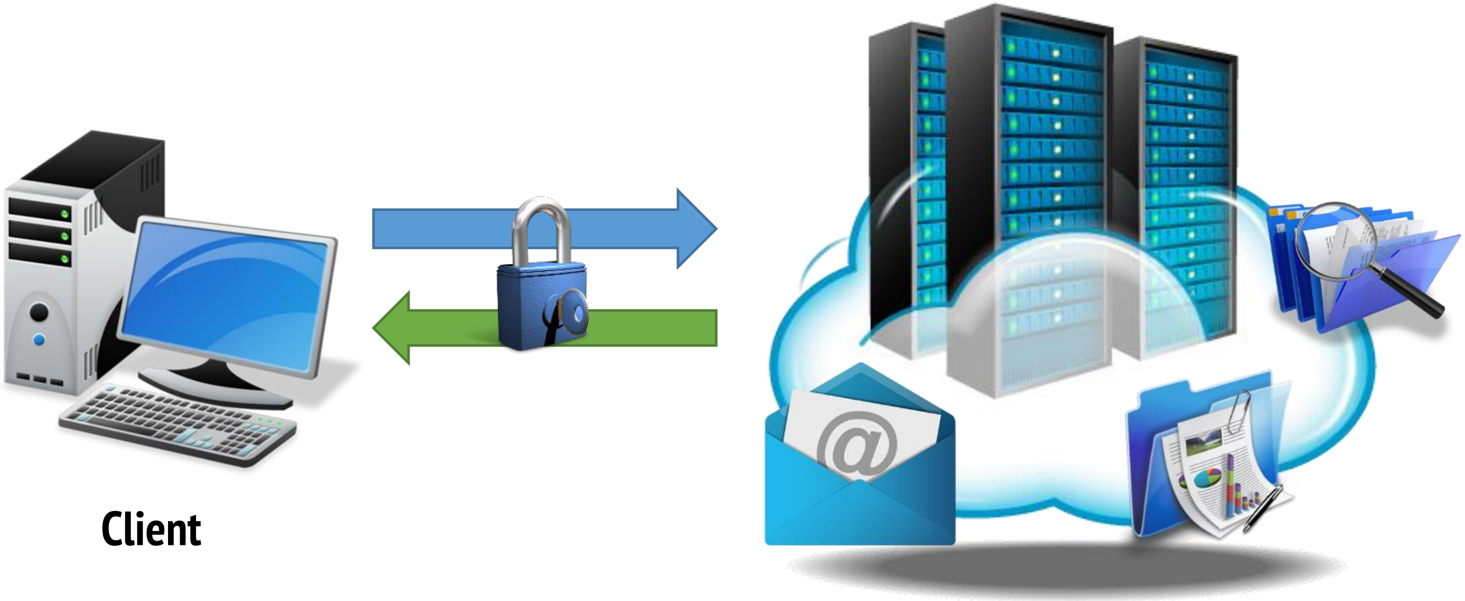
?

Client

Server

TLS, SSH, IPSec, ...

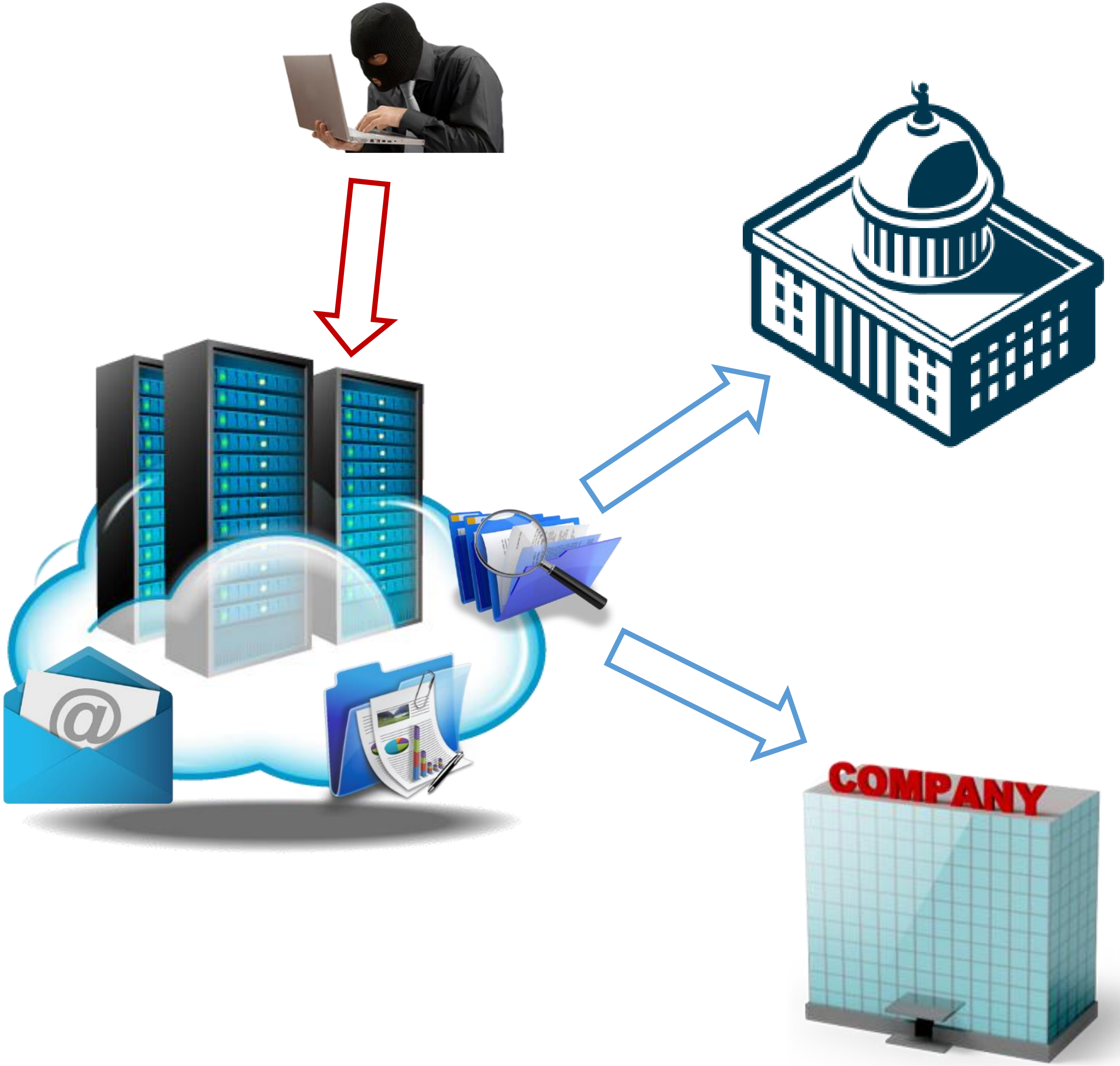
Protecting static data



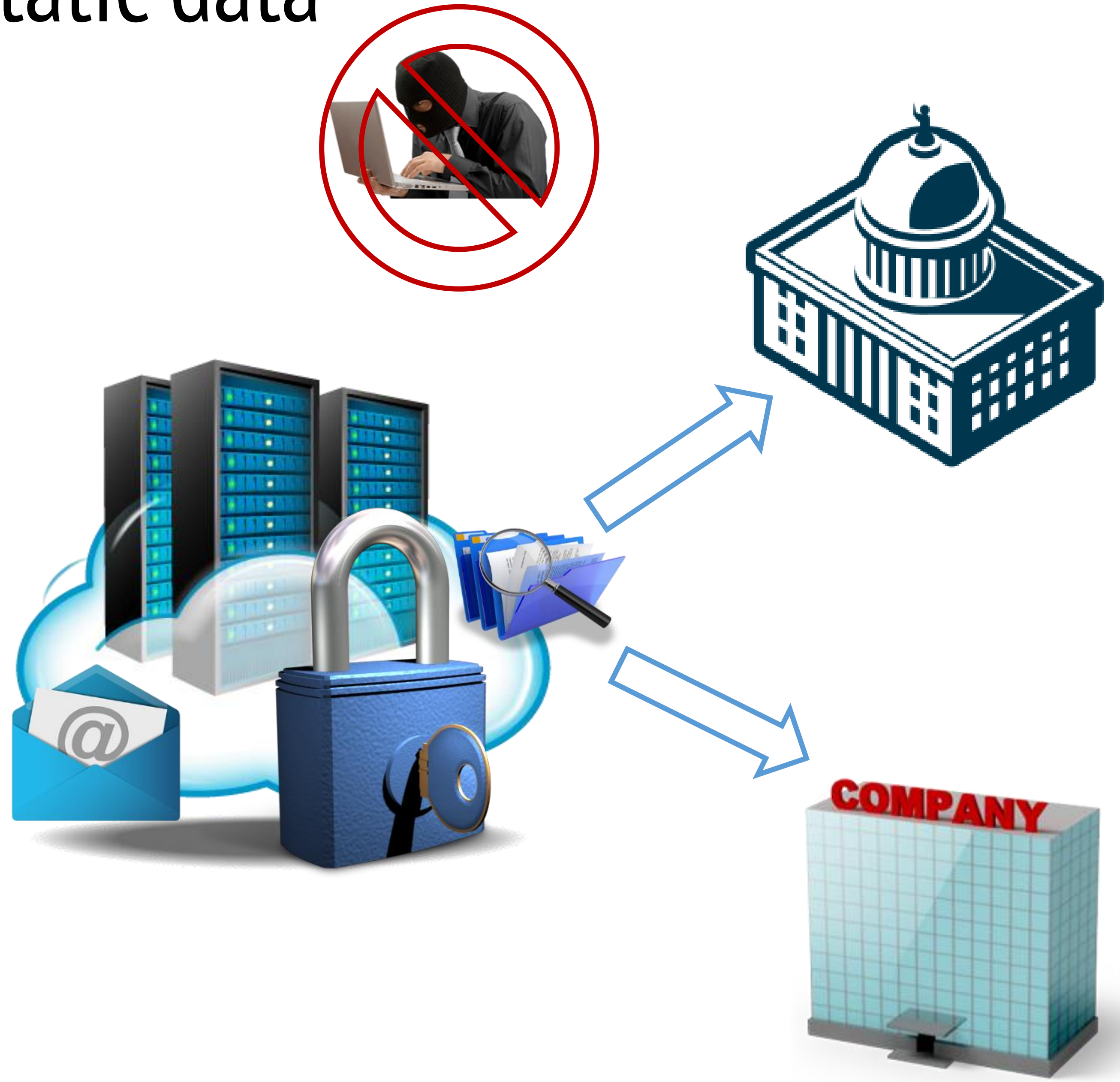
Client

Cloud

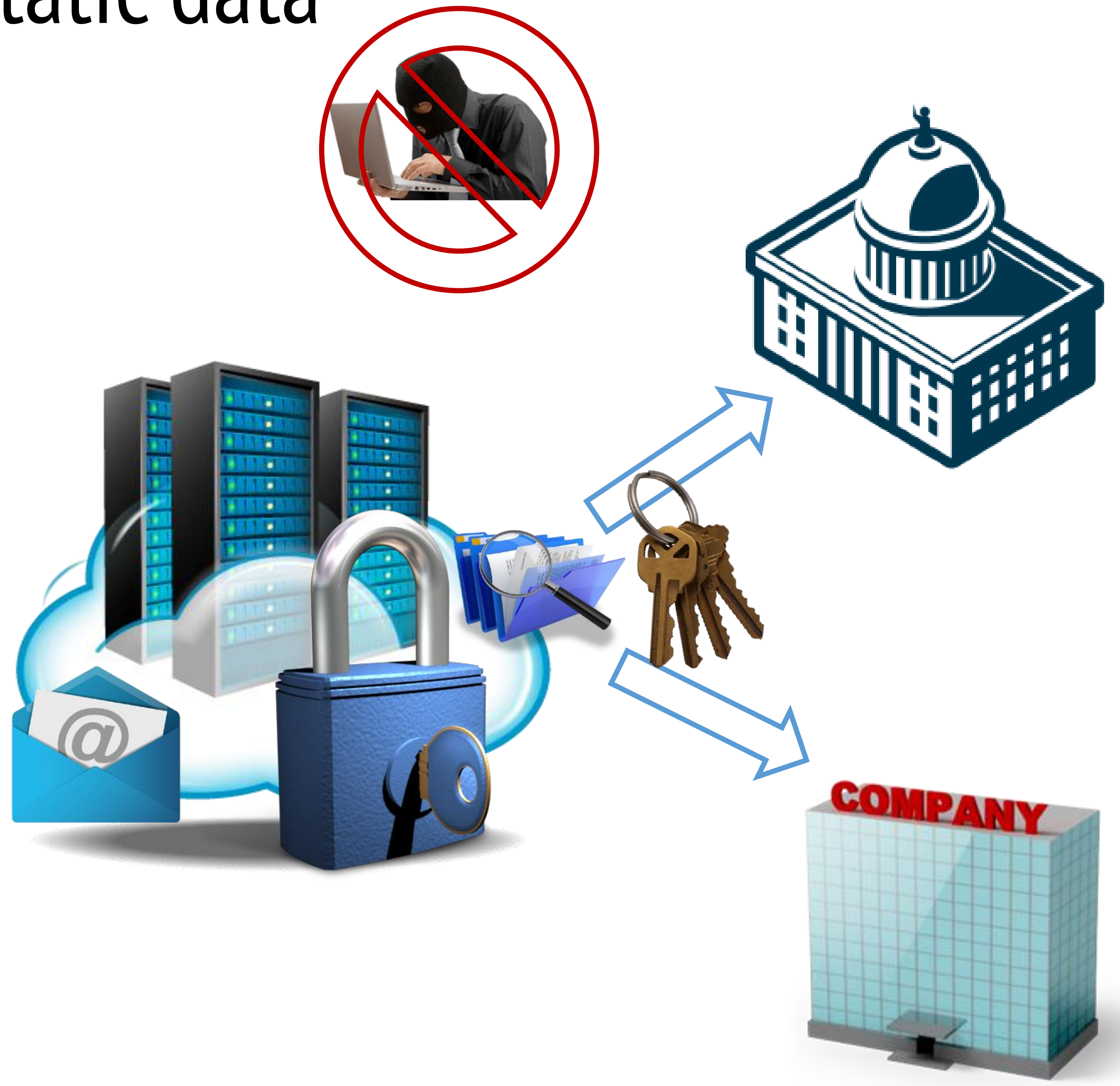
Protecting static data



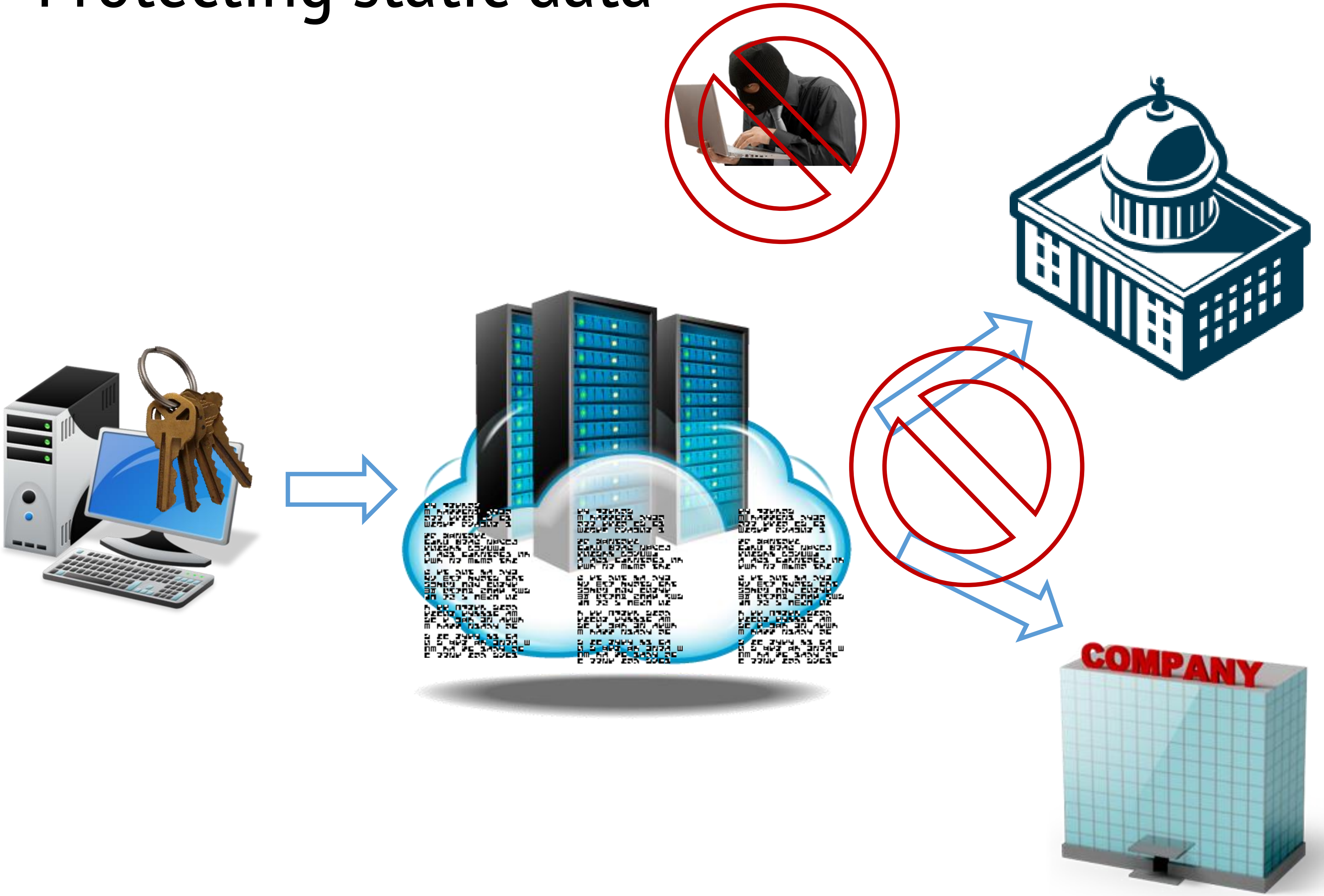
Protecting static data



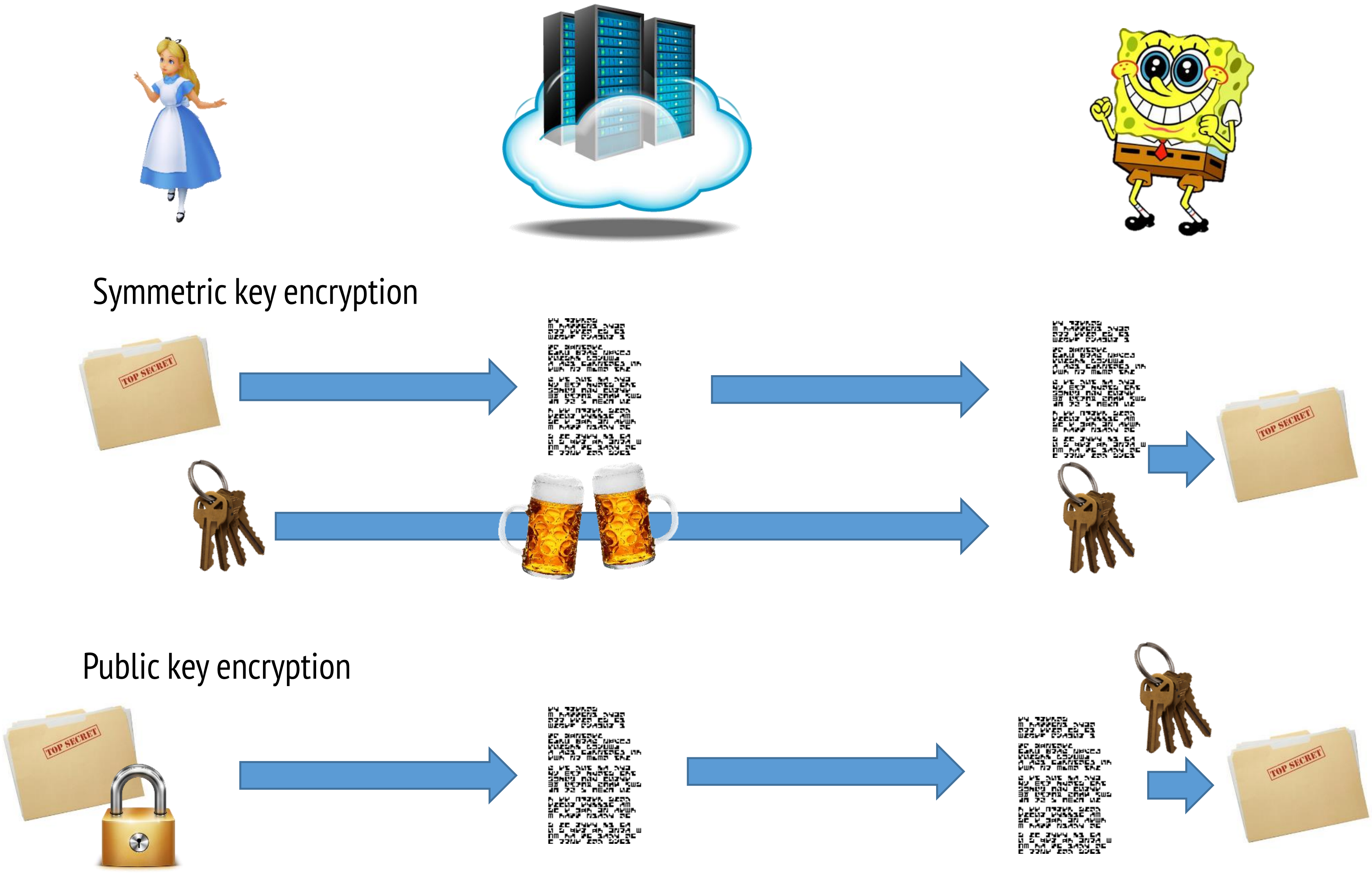
Protecting static data



Protecting static data



Protecting data from the cloud

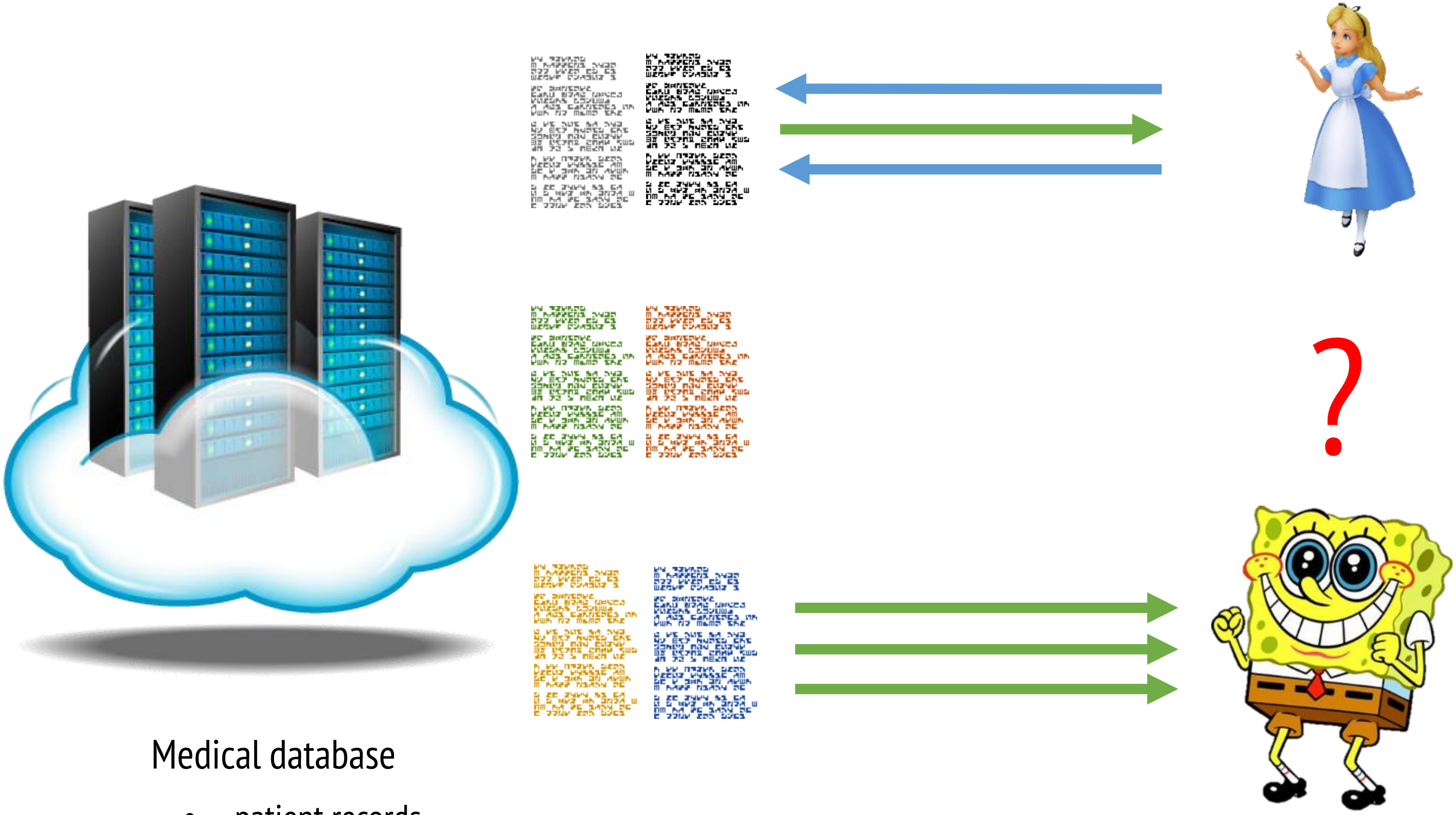


Symmetric key encryption

Public key encryption

Is encrypting data enough?

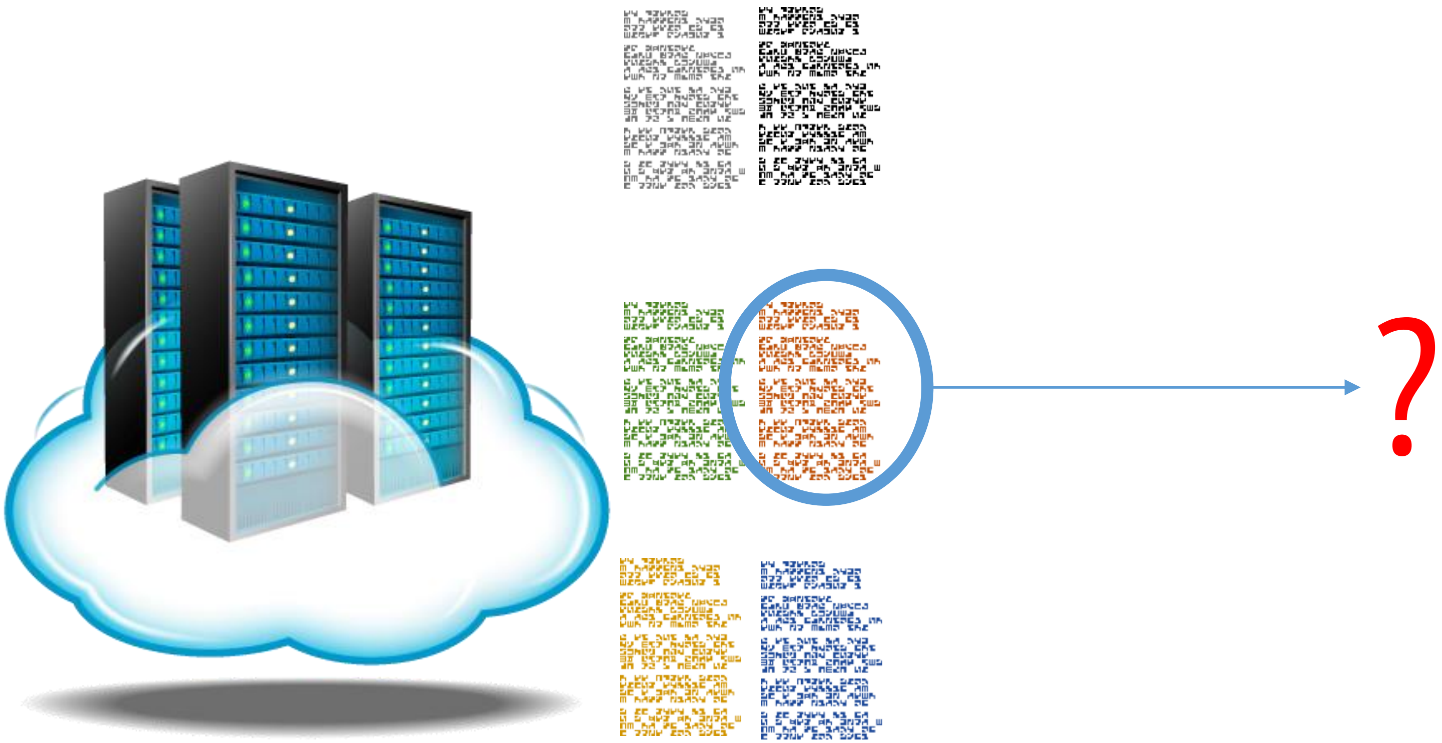
Analysing data access: who is the doctor?



Medical database

- patient records
- insurance records
- appointments

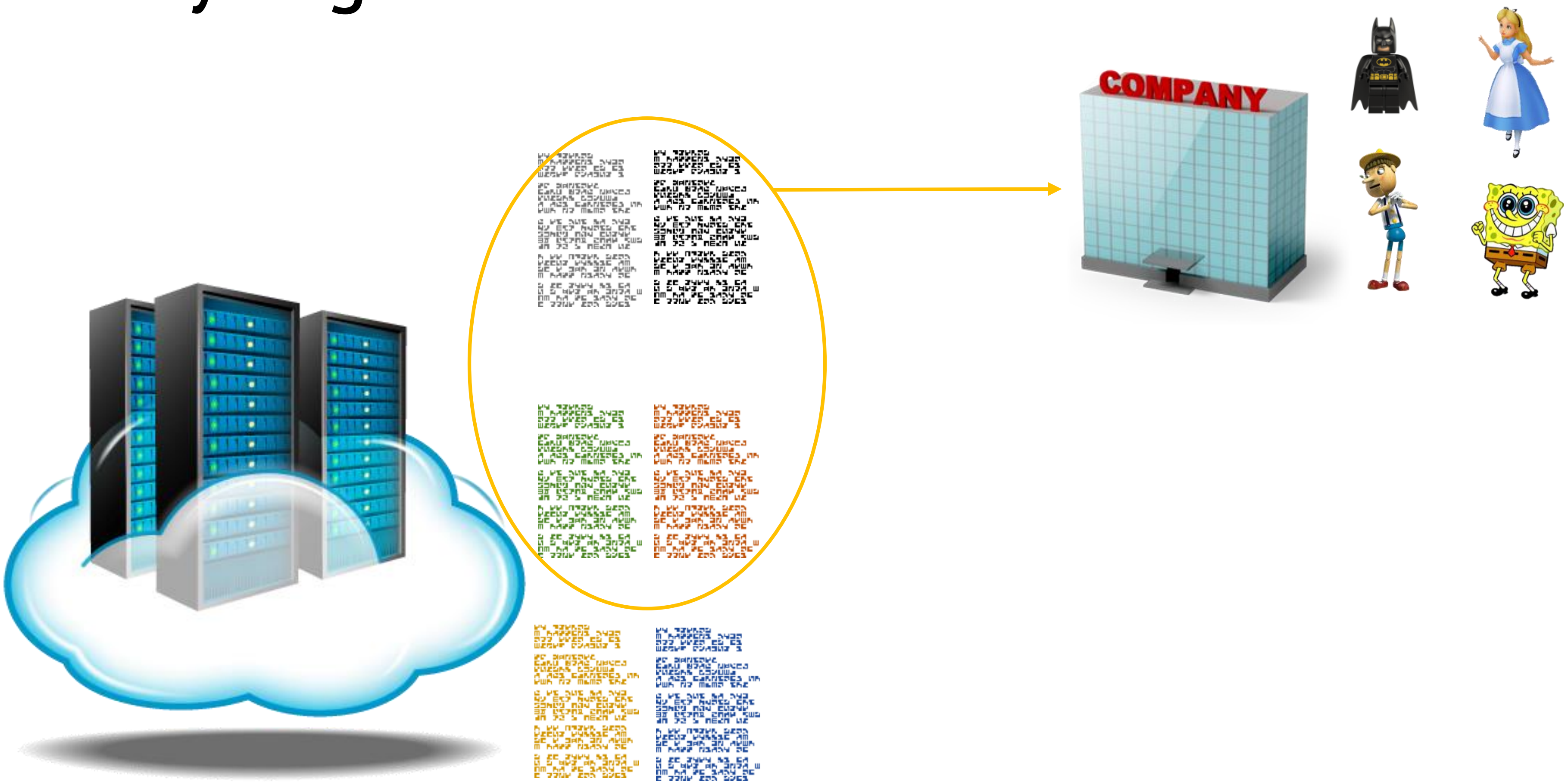
Analysing data access: who owns this ciphertext?



Medical database

- patient records
- insurance records
- appointments

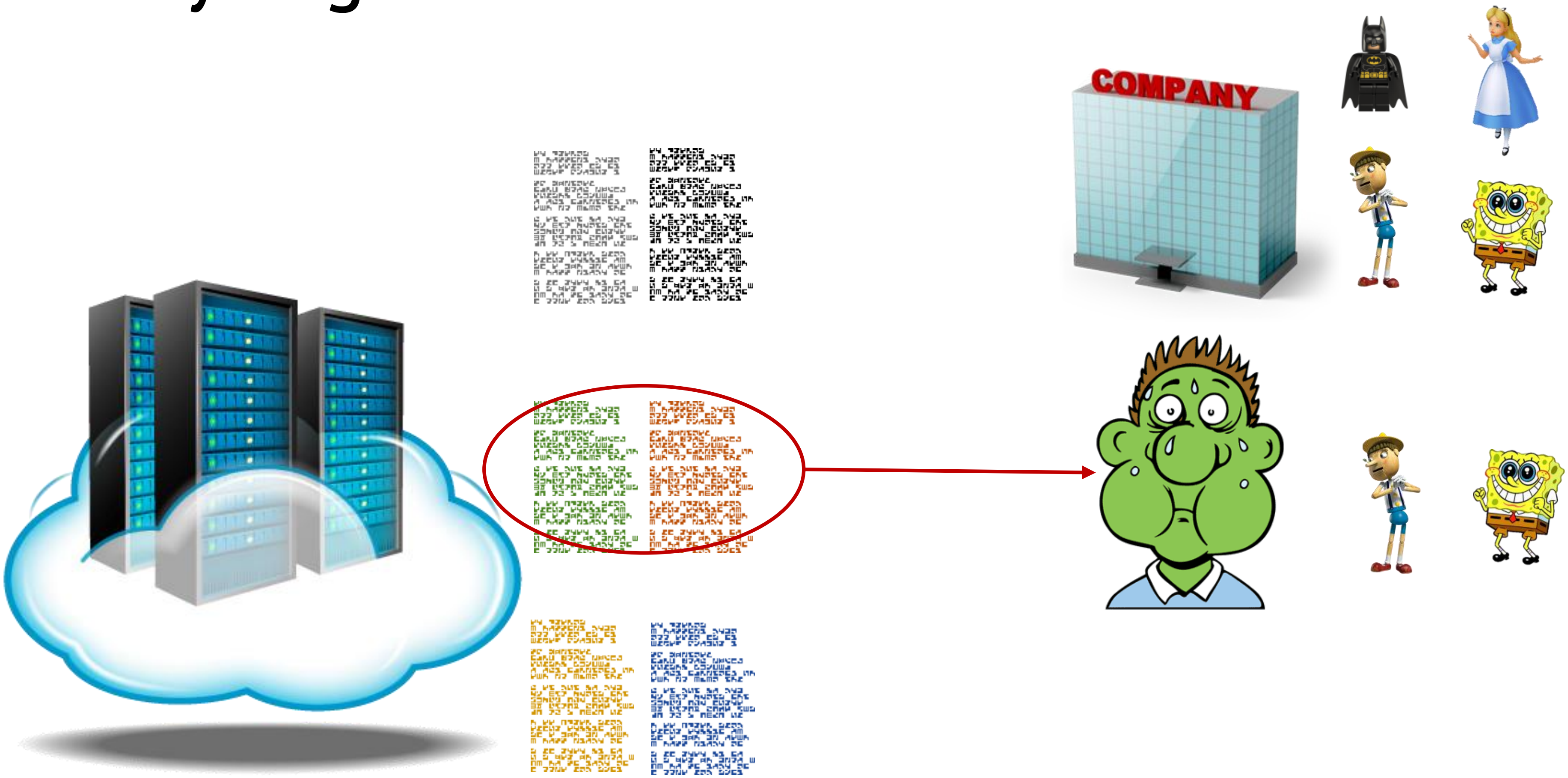
Analysing data access



Medical database

- patient records
- insurance records
- appointments

Analysing data access



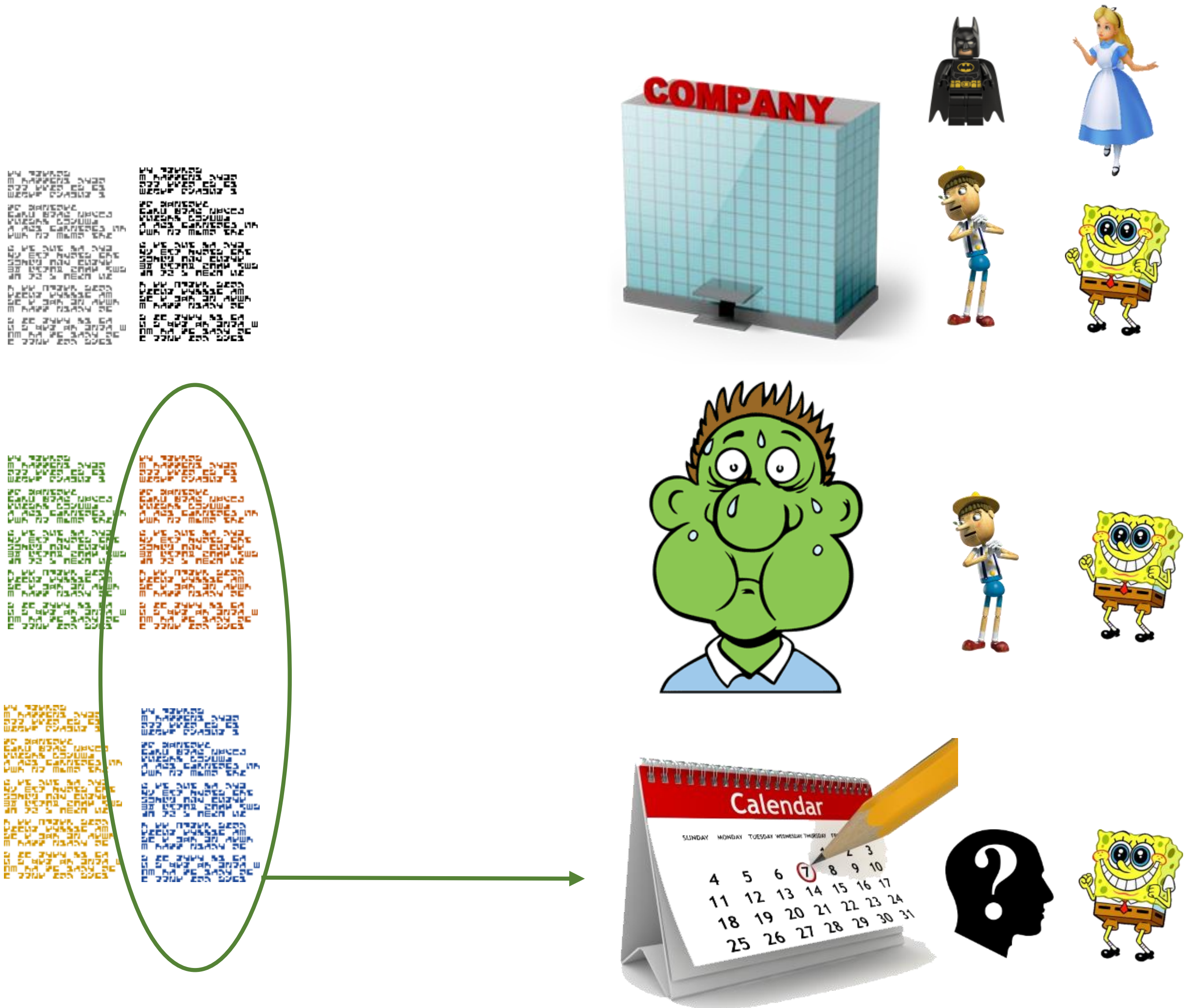
Medical database

- patient records
- insurance records
- appointments

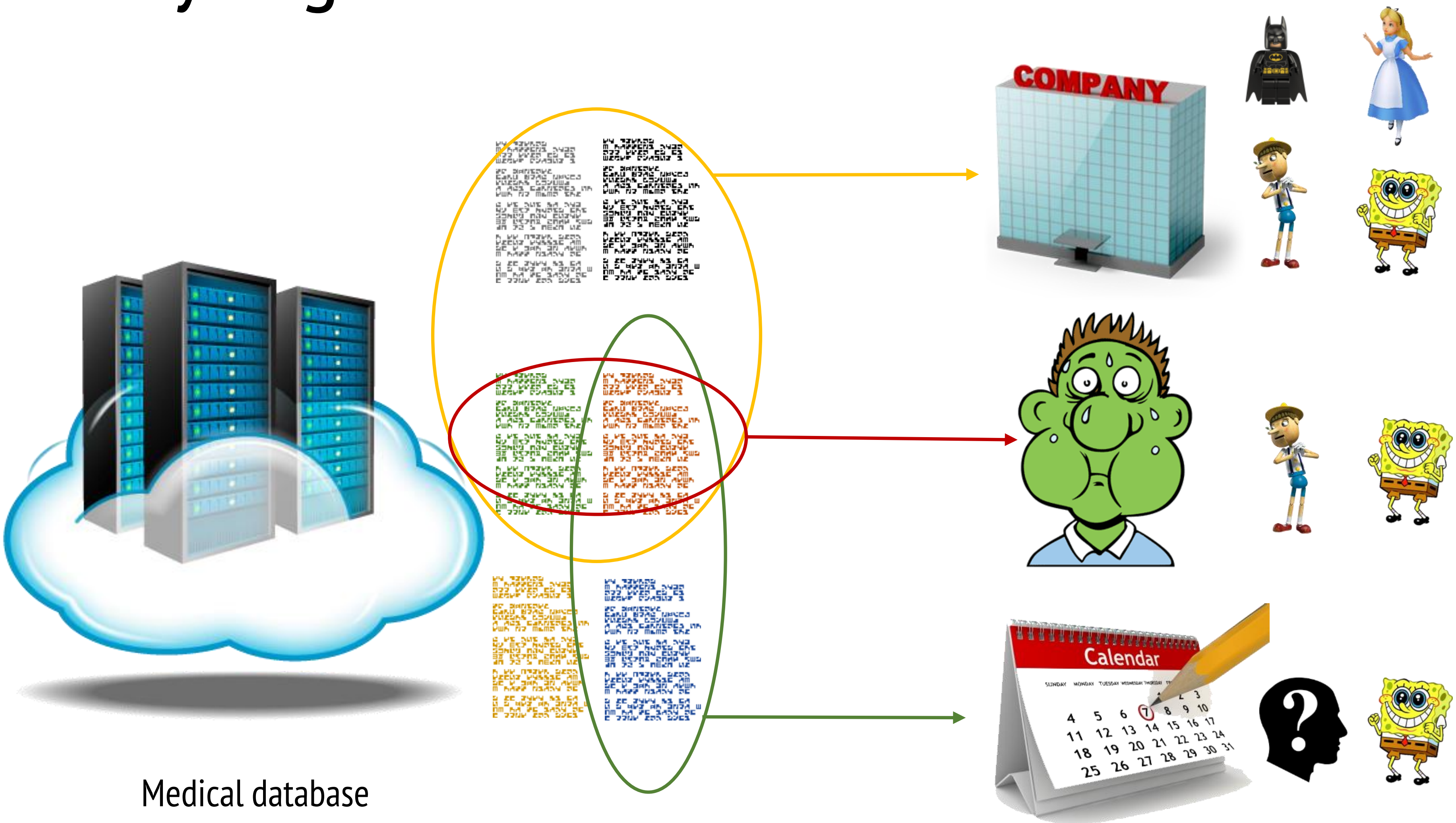
Analysing data access

Medical database

- patient records
- insurance records
- appointments



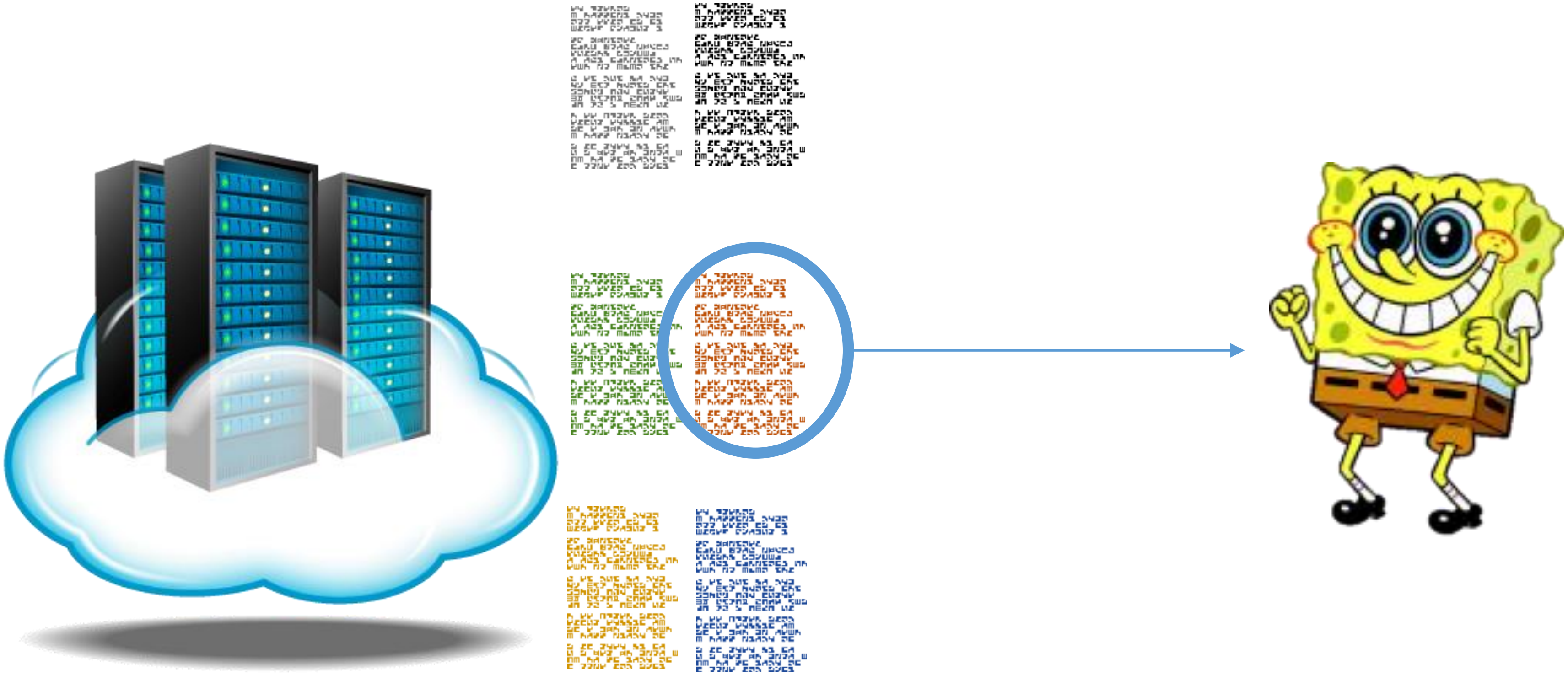
Analysing data access



Medical database

- patient records
- insurance records
- appointments

Analysing data access

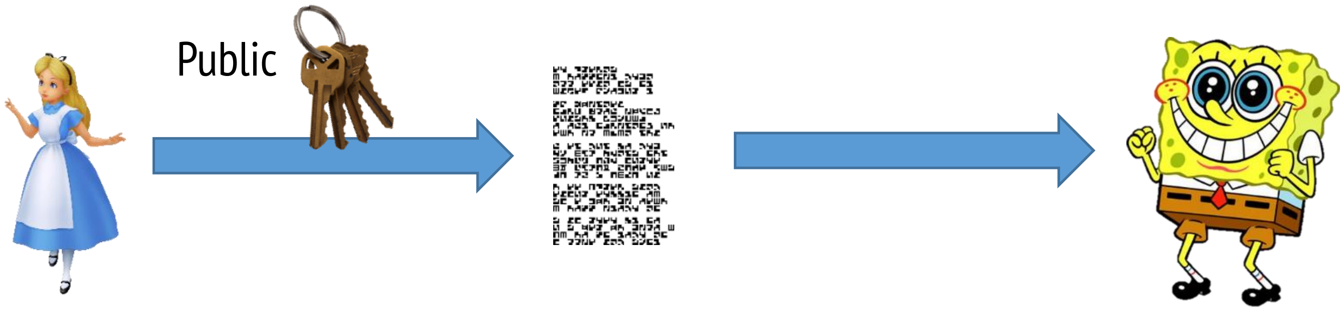


Medical database

- patient records
- insurance records
- appointments

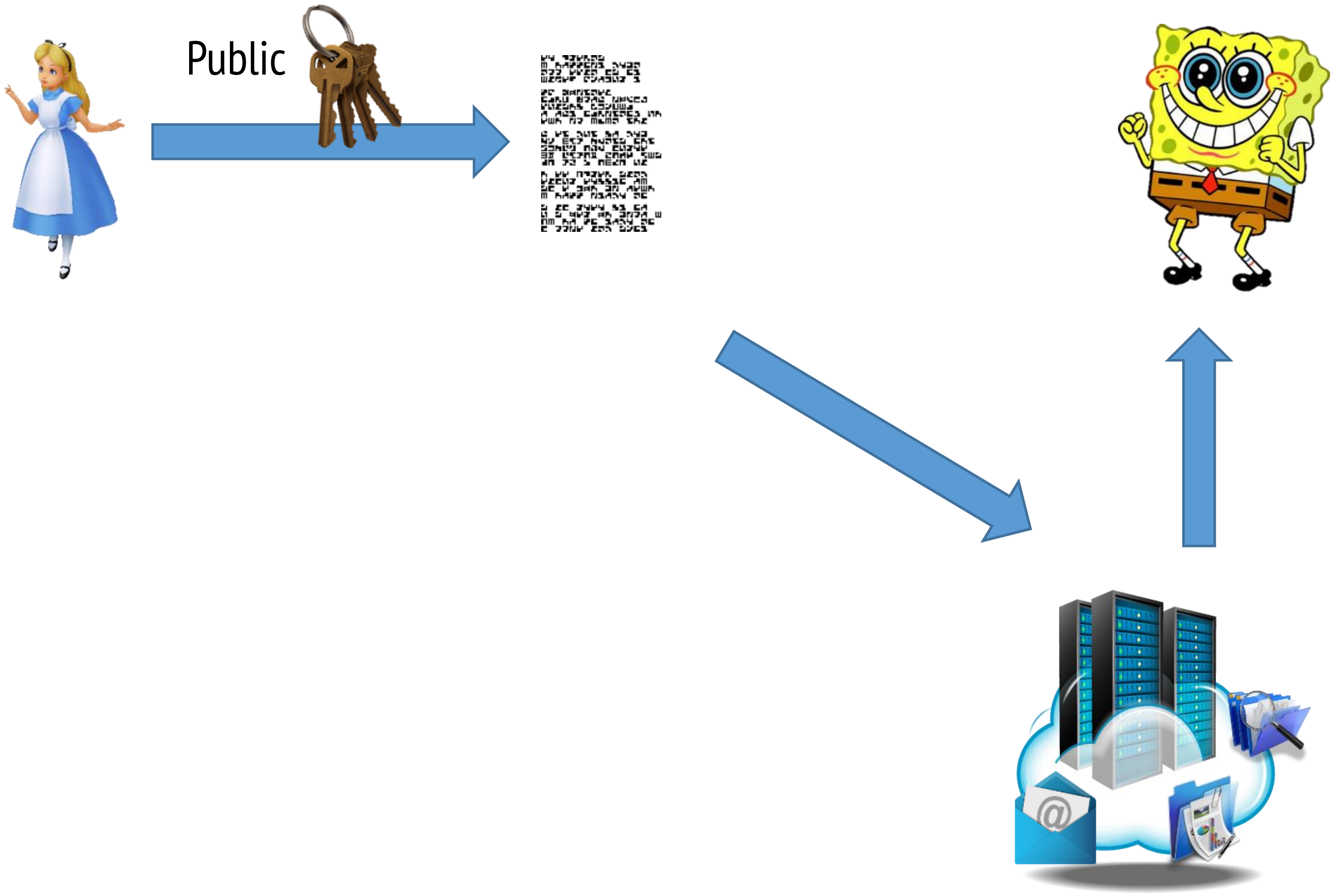
Cryptography issues

Anyone can use a public key to encrypt

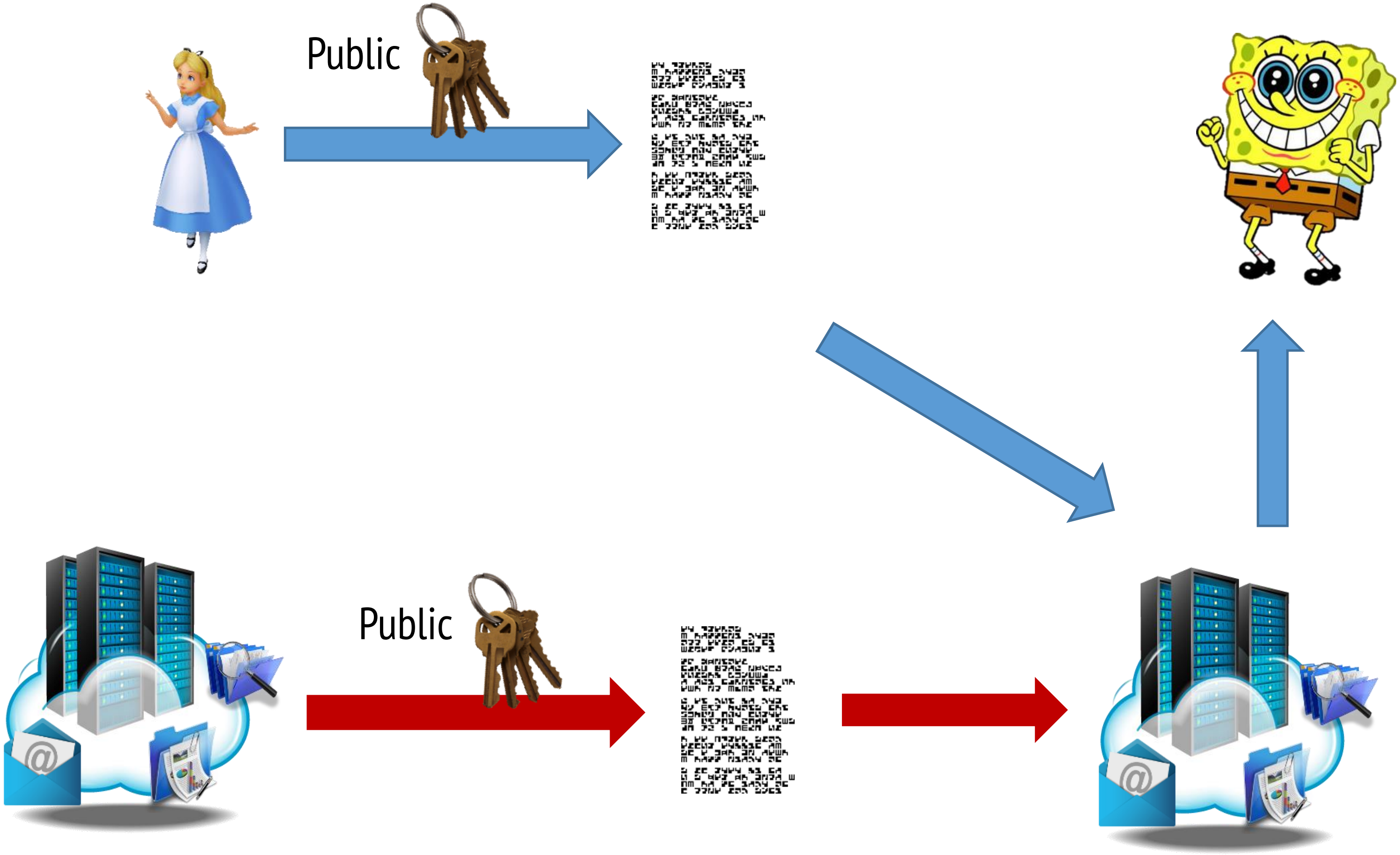


Public

Cryptography issues



Cryptography issues



Cryptography issues

- When protecting metadata, using public key crypto gives you a larger surface of attack.
- Symmetric crypto doesn’t have this problem **and** is more efficient.
- Symmetric keys are difficult to share.
- Design schemes based on **symmetric keys** and use **simple public key exchange protocols** to share them.

Just using encryption is not enough

✓**Content security** – the data is encrypted

✓**Metadata security** – ownership information, timestamps, access rights, ciphertext length, etc.

✓**Access pattern security** – when is the data accessed, who accesses the data, how is the data accessed, etc.

Searchable encryption

The challenge (in general)

- Assume we’re using Gmail to communicate (with a browser).
- Assume we’re using PGP to encrypt email (in browser).

- Can we decrypt email on the fly?
- Can we search through our emails?
- Who performs the search? Is it optimal?

The challenge (in general)

- Assume we're using Gmail to communicate (with a browser).
- Assume we're using PGP to encrypt email (in browser).

- Can we decrypt email on the fly?
  - YES
- Can we search through our emails?
  - Just the ones we decrypted
- Who performs the search? Is it optimal?
  - The client, in browser. Searching on the server would be the optimal choice

Solution?

- Assume we’re using Gmail to communicate (with a browser).
- Assume we’re using PGP to encrypt email (in browser).

- Generate a searchable index
- Store the index encrypted in the cloud

Solution?

- Assume we're using Gmail to communicate (with a browser).
- Assume we're using PGP to encrypt email (in browser).

- Generate a searchable index
- Store the index encrypted in the cloud

- Client has to download index every time
- Client still does the search, but it's much faster and can be done over all emails.

Solution?

- Assume we're using Gmail to communicate (with a browser).
- Assume we're using PGP to encrypt email (in browser).

- Generate a searchable index
- Store the index encrypted in the cloud

- ~~Client has to download index every time~~
- ~~Client still does the search, but it's much faster and can be done over all emails.~~
- THE SERVER SHOULD DO THE SEARCH! (no download, no computational effort)

Searchable Encryption

Searching

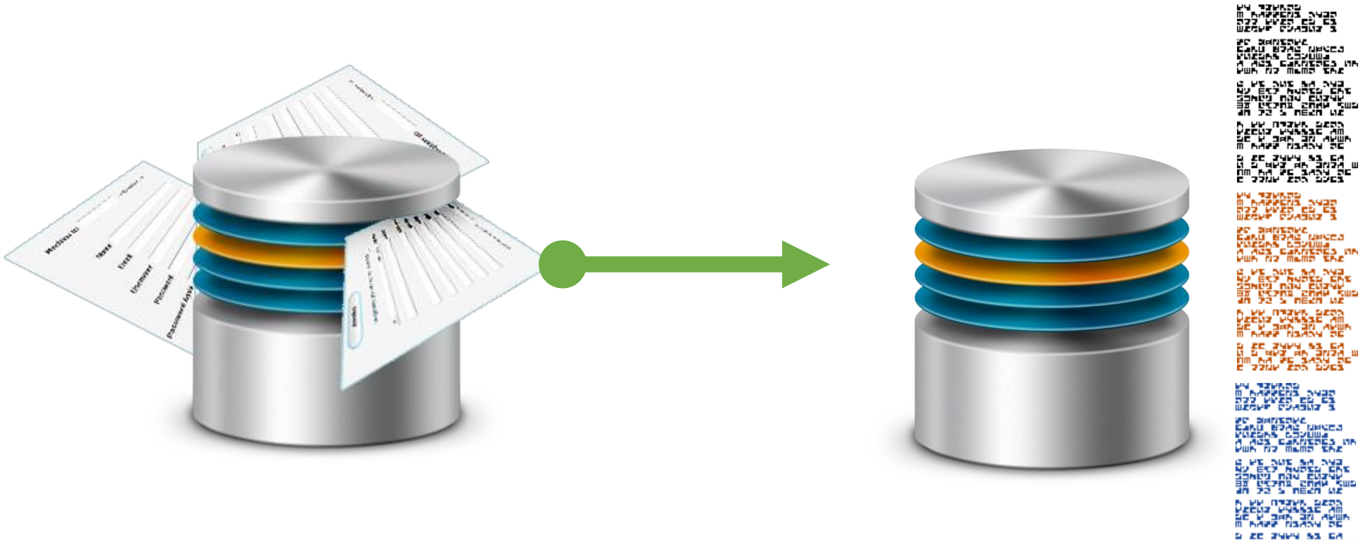
```
For each document in the database:
  For each word in document:
    if word = ‘top-secret’

print document-id
```

Encrypting databases

word → ENCRYPTED 'word' ⇔ WORD

document-id → ENCRYPTED 'document-id' ⇔ DOCUMENT-ID

document →



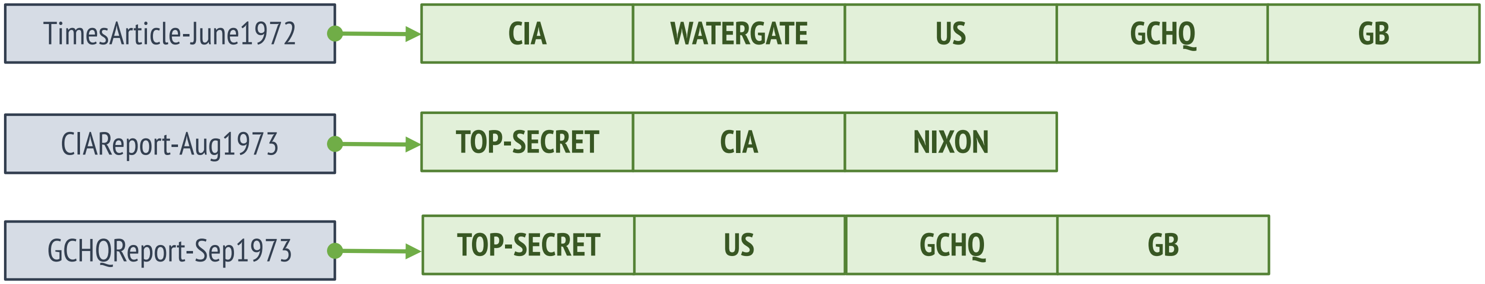
Database

Encrypted database

Searchable Encryption

KEYWORDS:	TOP-SECRET	CIA	WATERGATE	NIXON	US	GCHQ	GB

Forward index



TimesArticle-June1972	CIA	WATERGATE	US	GCHQ	GB
CIAReport-Aug1973	TOP-SECRET	CIA	NIXON		
GCHQReport-Sep1973	TOP-SECRET	US	GCHQ	GB	

Efficiency of the index

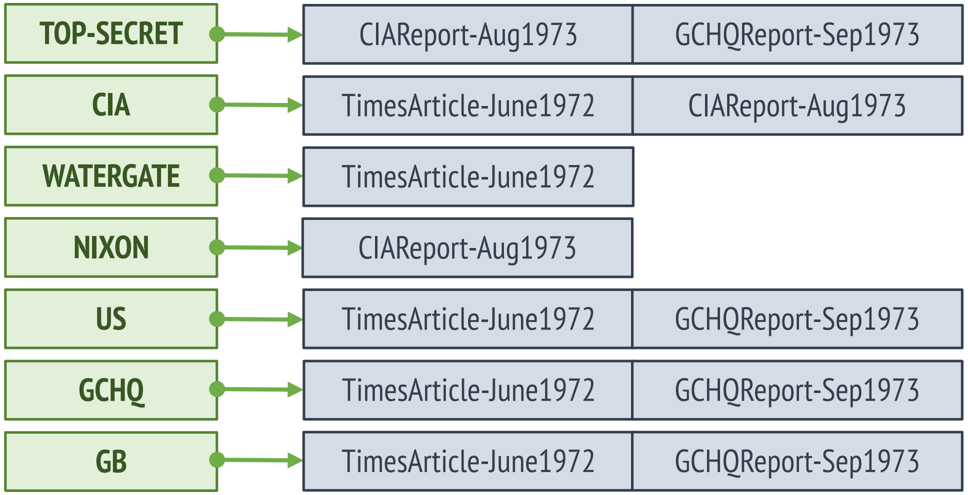
Number of **documents** increases => **time** increases

Number of **keywords** increases => **time** increases

Searchable Encryption

KEYWORDS:							
	TOP-SECRET	CIA	WATERGATE	NIXON	US	GCHQ	GB

Inverted index



TOP-SECRET	CIAReport-Aug1973	GCHQReport-Sep1973
CIA	TimesArticle-June1972	CIAReport-Aug1973
WATERGATE	TimesArticle-June1972	
NIXON	CIAReport-Aug1973	
US	TimesArticle-June1972	GCHQReport-Sep1973
GCHQ	TimesArticle-June1972	GCHQReport-Sep1973
GB	TimesArticle-June1972	GCHQReport-Sep1973

Efficiency of the index

Number of **keywords** increases => **time** increases

What do we want to protect?

What we search for

KEYWORDS:	TOP-SECRET	CIA	WATERGATE	...

What is the result of the search query

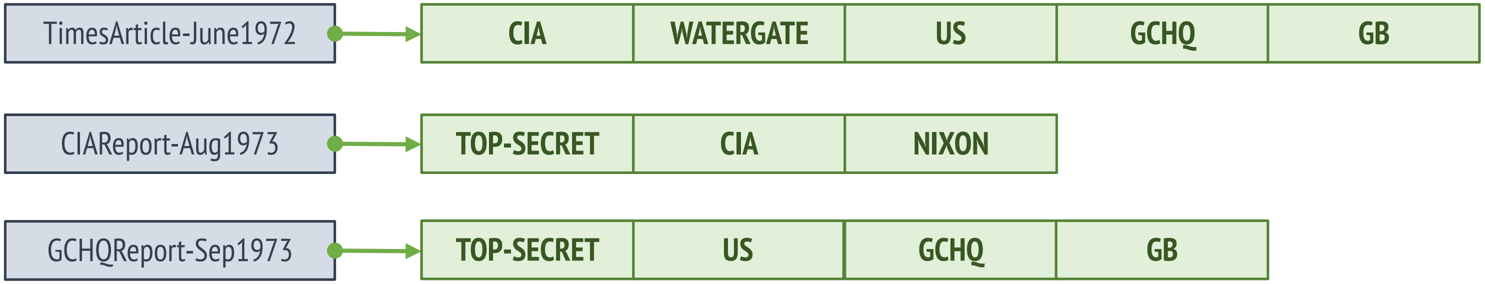
DOCUMENT NAMES:	CIAReport-Aug1973	GCHQReport-Sep1973

How often we search for something

1:	TOP-SECRET
2:	CIA
	...
n:	TOP-SECRET

Padding

Forward index



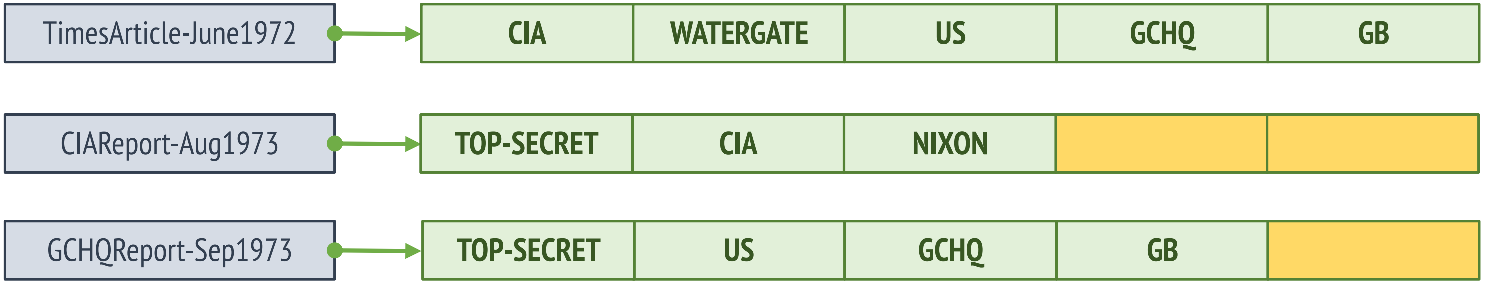
TimesArticle-June1972 → CIA | WATERGATE | US | GCHQ | GB

CIAReport-Aug1973 → TOP-SECRET | CIA | NIXON

GCHQReport-Sep1973 → TOP-SECRET | US | GCHQ | GB

Padding

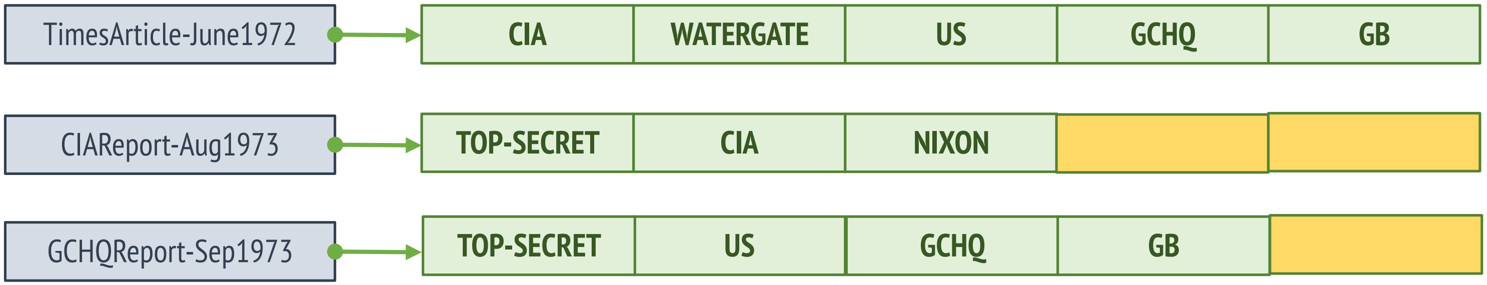
Forward index



TimesArticle-June1972	CIA	WATERGATE	US	GCHQ	GB
CIAReport-Aug1973	TOP-SECRET	CIA	NIXON	🟨	🟨
GCHQReport-Sep1973	TOP-SECRET	US	GCHQ	GB	🟨

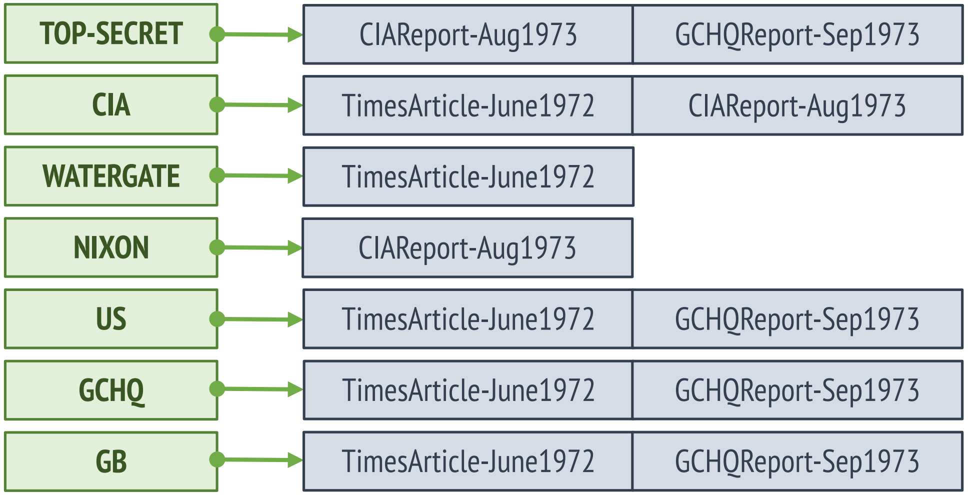
Padding

Forward index



TimesArticle-June1972	CIA	WATERGATE	US	GCHQ	GB
CIAReport-Aug1973	TOP-SECRET	CIA	NIXON	🟨	🟨
GCHQReport-Sep1973	TOP-SECRET	US	GCHQ	GB	🟨

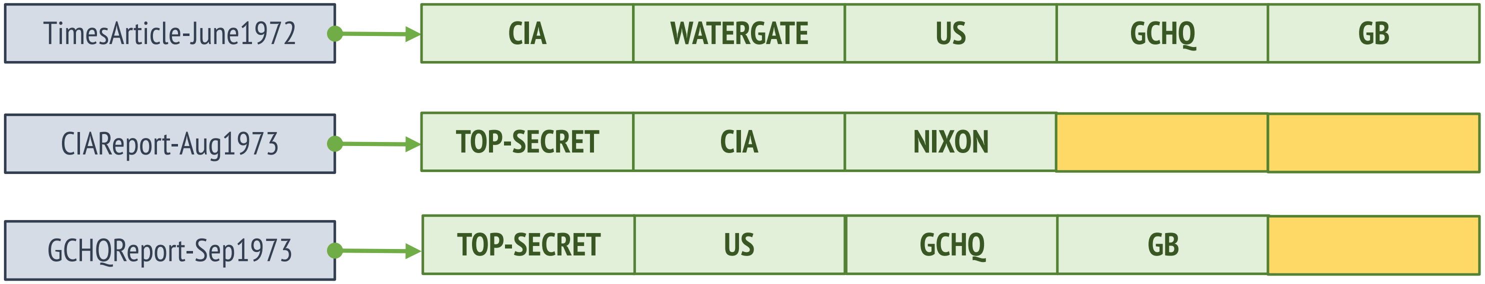
Inverted index



TOP-SECRET	CIAReport-Aug1973	GCHQReport-Sep1973
CIA	TimesArticle-June1972	CIAReport-Aug1973
WATERGATE	TimesArticle-June1972	
NIXON	CIAReport-Aug1973	
US	TimesArticle-June1972	GCHQReport-Sep1973
GCHQ	TimesArticle-June1972	GCHQReport-Sep1973
GB	TimesArticle-June1972	GCHQReport-Sep1973

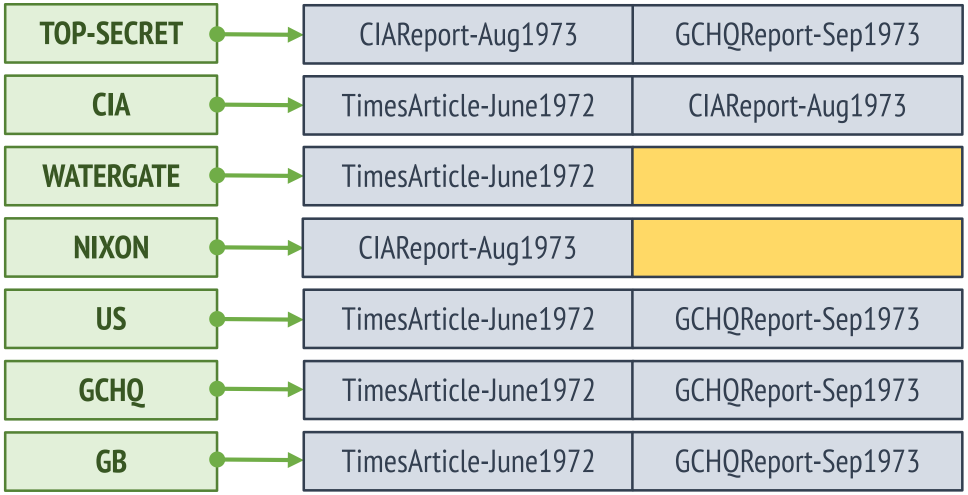
Padding

Forward index



TimesArticle-June1972	CIA	WATERGATE	US	GCHQ	GB
CIAReport-Aug1973	TOP-SECRET	CIA	NIXON	🟨	🟨
GCHQReport-Sep1973	TOP-SECRET	US	GCHQ	GB	🟨

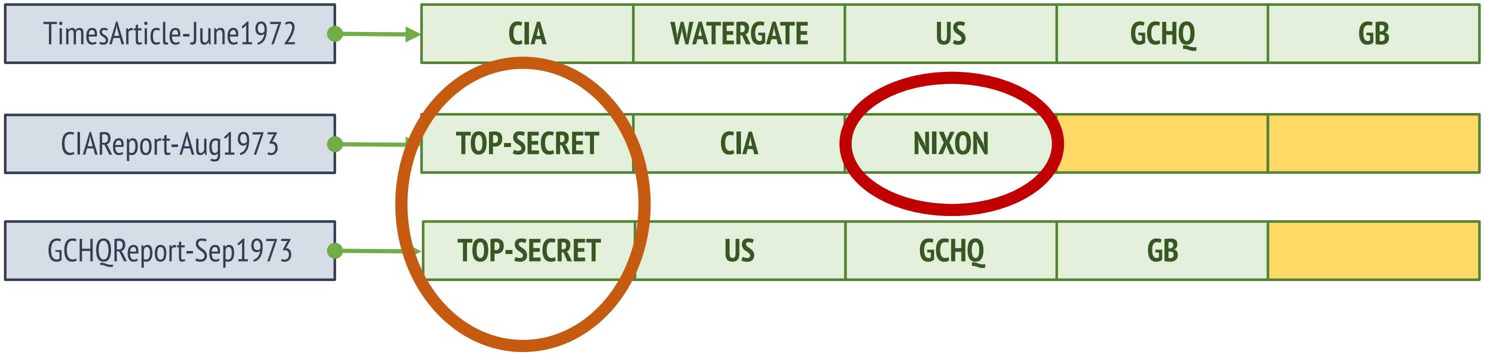
Inverted index



TOP-SECRET	CIAReport-Aug1973	GCHQReport-Sep1973
CIA	TimesArticle-June1972	CIAReport-Aug1973
WATERGATE	TimesArticle-June1972	🟨
NIXON	CIAReport-Aug1973	🟨
US	TimesArticle-June1972	GCHQReport-Sep1973
GCHQ	TimesArticle-June1972	GCHQReport-Sep1973
GB	TimesArticle-June1972	GCHQReport-Sep1973

Intersections

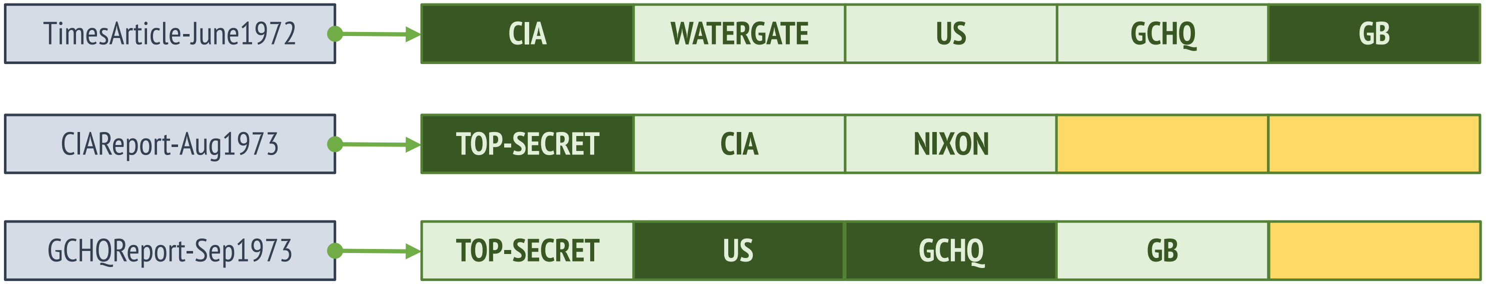
Forward index



TimesArticle-June1972	CIA	WATERGATE	US	GCHQ	GB
CIAReport-Aug1973	TOP-SECRET	CIA	NIXON		
GCHQReport-Sep1973	TOP-SECRET	US	GCHQ	GB	

Intersections

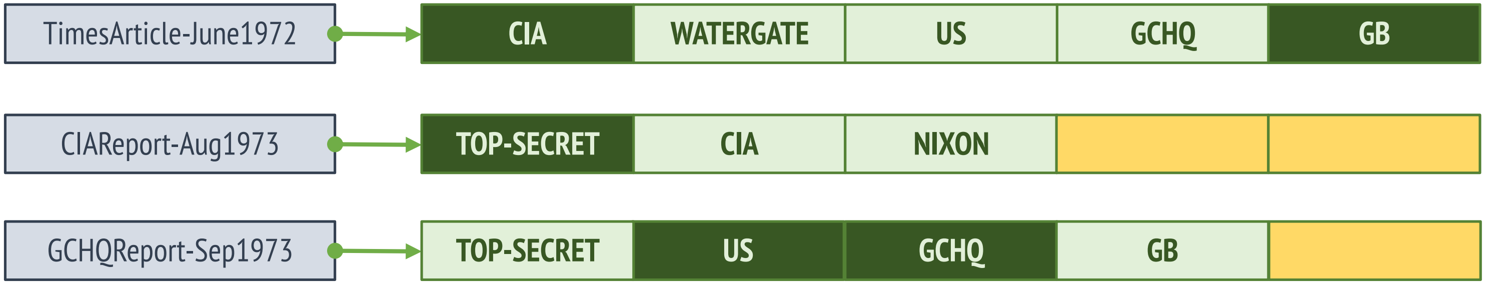
Forward index



TimesArticle-June1972	CIA	WATERGATE	US	GCHQ	GB
CIAReport-Aug1973	TOP-SECRET	CIA	NIXON		
GCHQReport-Sep1973	TOP-SECRET	US	GCHQ	GB	

Intersections

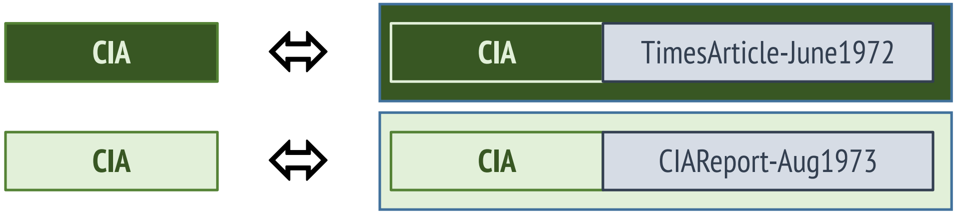
Forward index


TimesArticle-June1972 → CIA | WATERGATE | US | GCHQ | GB

CIAReport-Aug1973 → TOP-SECRET | CIA | NIXON

GCHQReport-Sep1973 → TOP-SECRET | US | GCHQ | GB

We want


CIA ⇔ CIA | TimesArticle-June1972

CIA ⇔ CIA | CIAReport-Aug1973

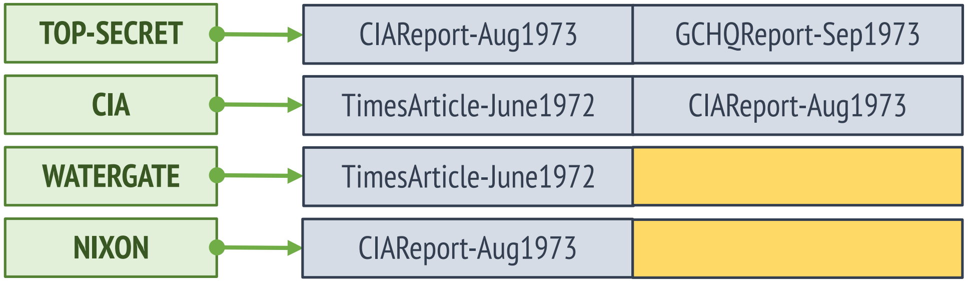
Server the computation

1. Client work needs to be as low as possible.

2. Server needs to do most of the search work.

Secure searching

Inverted index:

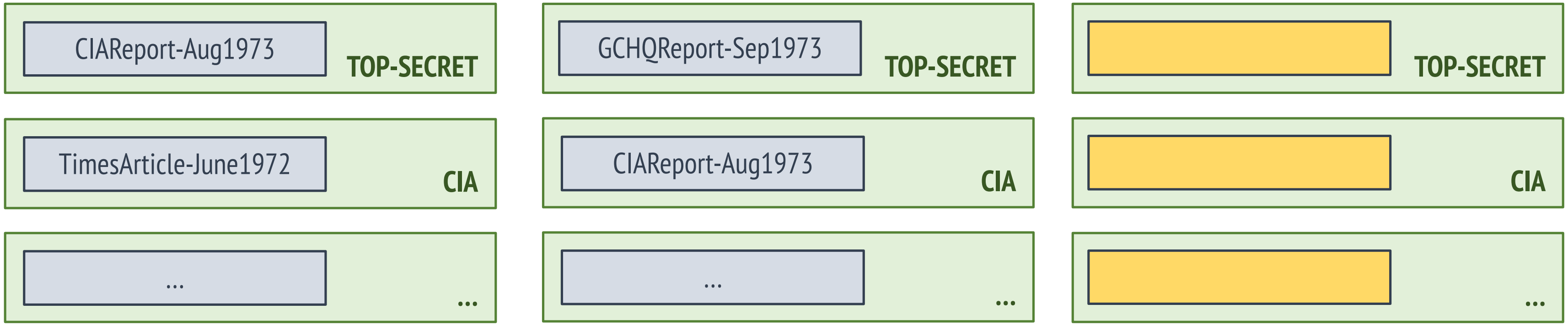


TOP-SECRET	CIAReport-Aug1973	GCHQReport-Sep1973
CIA	TimesArticle-June1972	CIAReport-Aug1973
WATERGATE	TimesArticle-June1972	
NIXON	CIAReport-Aug1973	

Symmetric key searchable encryption index:

ENC. DOC. NAMES: CIAReport-Aug1973 GCHQReport-Sep1973 TimesArticle-June1972

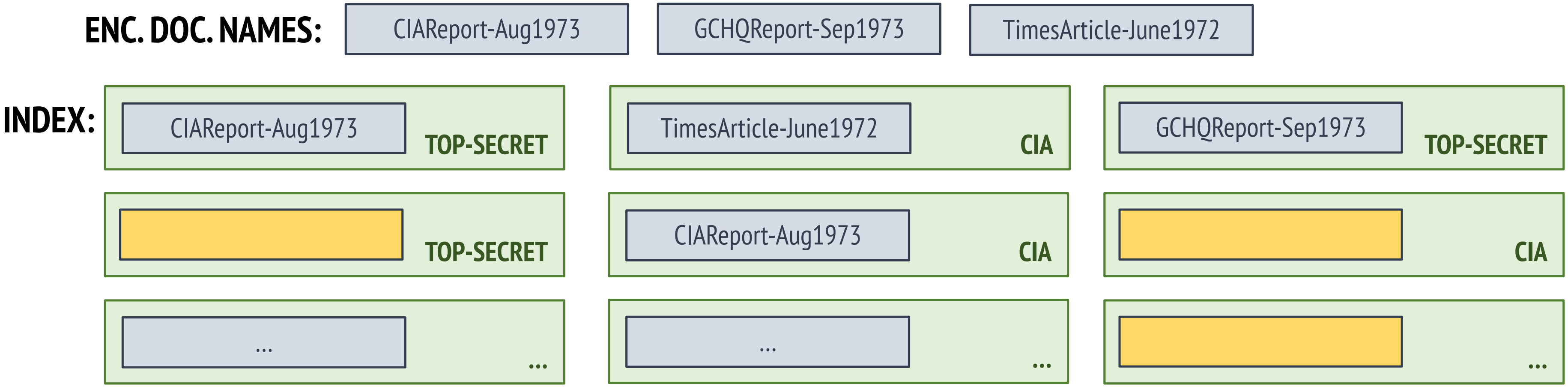
INDEX:



CIAReport-Aug1973 TOP-SECRET	GCHQReport-Sep1973 TOP-SECRET	TOP-SECRET
TimesArticle-June1972 CIA	CIAReport-Aug1973 CIA	CIA
... ...	... ...	...

Secure searching

Server has:



ENC. DOC. NAMES: CIAReport-Aug1973 GCHQReport-Sep1973 TimesArticle-June1972

INDEX:

CIAReport-Aug1973 TOP-SECRET	TimesArticle-June1972 CIA	GCHQReport-Sep1973 TOP-SECRET
🟨 TOP-SECRET	CIAReport-Aug1973 CIA	🟨 CIA
... ...	... ...	🟨 ...

Search term:

TOP-SECRET

Secure searching

Server has:

ENC. DOC. NAMES: CIAReport-Aug1973 GCHQReport-Sep1973 TimesArticle-June1972

INDEX:

CIAReport-Aug1973 **TOP-SECRET**	TimesArticle-June1972 **CIA**	GCHQReport-Sep1973 **TOP-SECRET**
🟨 **TOP-SECRET**	CIAReport-Aug1973 **CIA**	🟨 **CIA**
... **...**	... **...**	🟨 **...**

Search term:

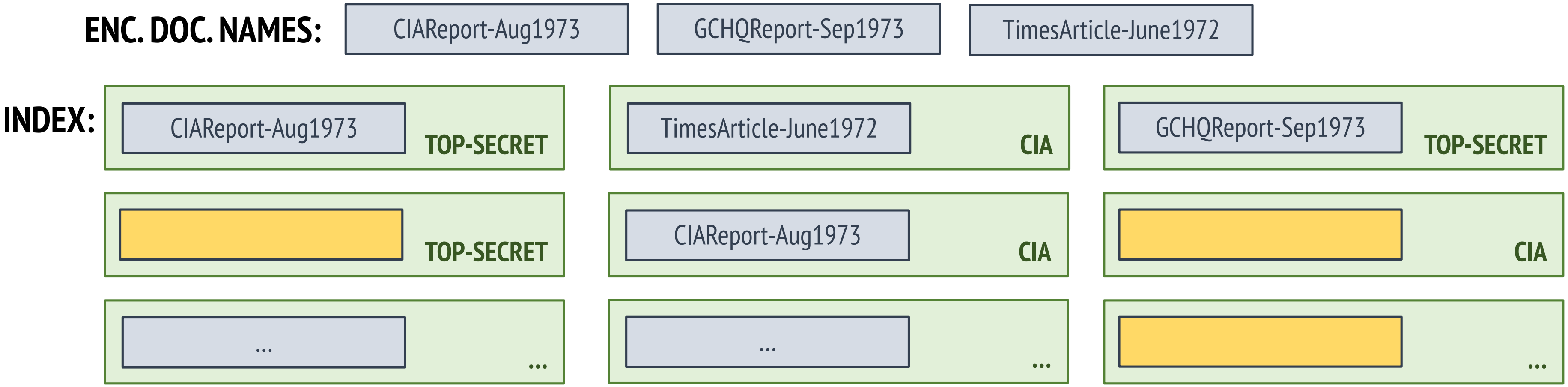
TOP-SECRET

Server computation:

CIAReport-Aug1973 GCHQReport-Sep1973 TimesArticle-June1972

Secure searching

Server has:



ENC. DOC. NAMES: CIAReport-Aug1973 | GCHQReport-Sep1973 | TimesArticle-June1972

INDEX:

CIAReport-Aug1973 TOP-SECRET

TimesArticle-June1972 CIA

GCHQReport-Sep1973 TOP-SECRET

TOP-SECRET

CIAReport-Aug1973 CIA

CIA

… …

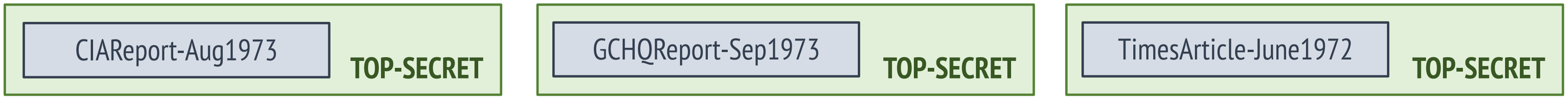
… …

…

Search term:

TOP-SECRET

Server computation:



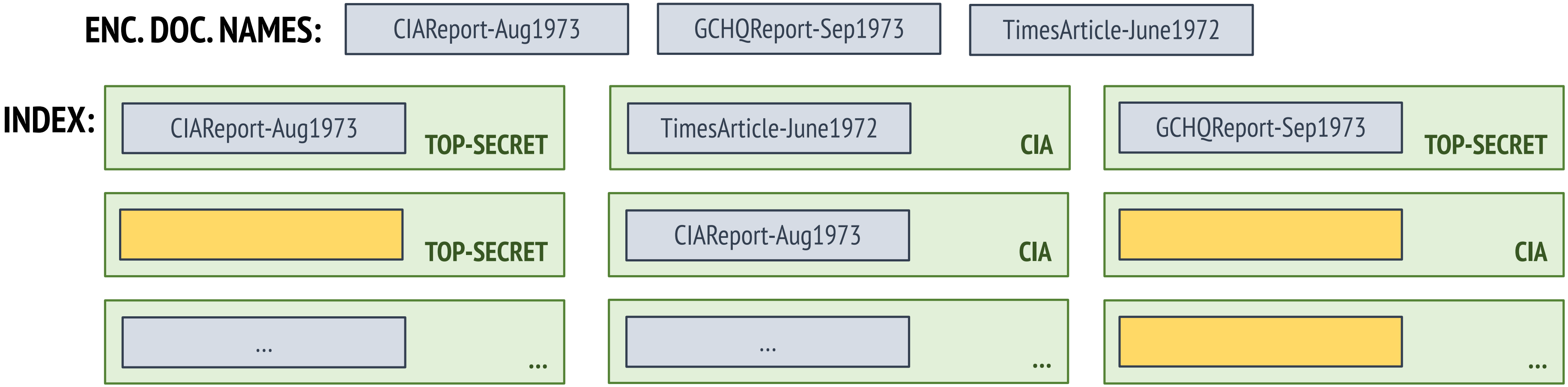
CIAReport-Aug1973 TOP-SECRET

GCHQReport-Sep1973 TOP-SECRET

TimesArticle-June1972 TOP-SECRET

Secure searching

Server has:



ENC. DOC. NAMES:

CIAReport-Aug1973 | GCHQReport-Sep1973 | TimesArticle-June1972

INDEX:

CIAReport-Aug1973 TOP-SECRET | TimesArticle-June1972 CIA | GCHQReport-Sep1973 TOP-SECRET

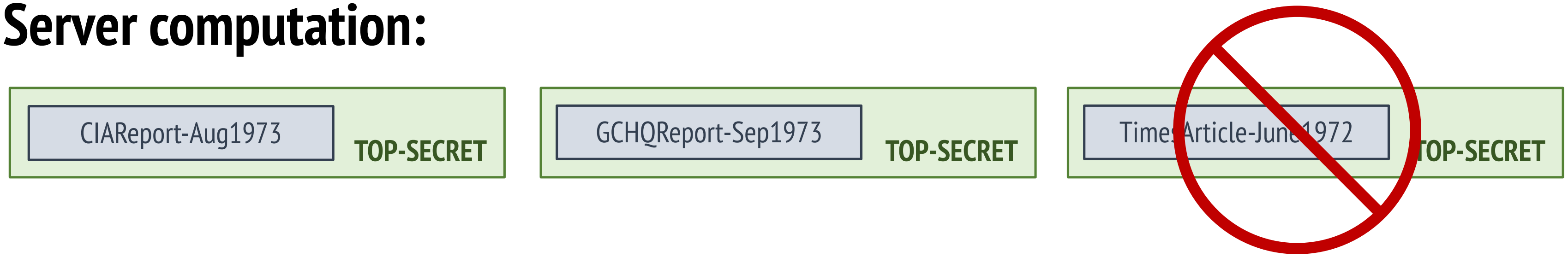
🟨 TOP-SECRET | CIAReport-Aug1973 CIA | 🟨 CIA

... ... | ... ... | 🟨 ...

Search term:

TOP-SECRET

Server computation:



CIAReport-Aug1973 TOP-SECRET | GCHQReport-Sep1973 TOP-SECRET | TimesArticle-June1972 TOP-SECRET

Secure searching

Server has:

ENC. DOC. NAMES: CIAReport-Aug1973 GCHQReport-Sep1973 TimesArticle-June1972

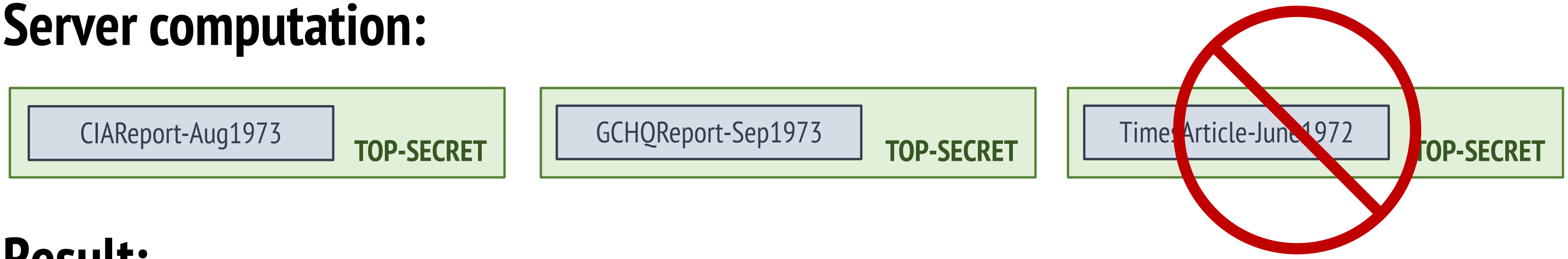
INDEX:

CIAReport-Aug1973	TOP-SECRET	TimesArticle-June1972	CIA	GCHQReport-Sep1973	TOP-SECRET
🟨	TOP-SECRET	CIAReport-Aug1973	CIA	🟨	CIA
...	...	...	...	🟨	...

Search term:

TOP-SECRET

Server computation:



CIAReport-Aug1973 TOP-SECRET

GCHQReport-Sep1973 TOP-SECRET

TimesArticle-June1972 TOP-SECRET

Result:

CIAReport-Aug1973 GCHQReport-Sep1973

Performance

Example 1 - OXT:

[Cash-Jarecki-Jutla-Krawczyk-Rosu-Steiner13]

- Encrypted database size: 13GB
- DB Contents: 1.5 million emails & attachments
- **Avg. search time: less than 500ms**

Example 2 – 2Lev:

[Cash-Jaeger-Jarecki-Jutla-Krawczyk-Steiner-Rosu14]

- Encrypted database size: 900GBs
- Setup time: 16 hours
- **Avg. query time: less than 200ms**

Searchable encryption limitations

- Encrypted search term is deterministic
- Only search patterns are hidden
- Setting up the index requires a significant amount of time
- Most schemes do not support index extensions

DISCLAIMER :-)

- Searchable encryption **solves** problems related to the security of the search index.

- Searchable encryption **does not solve** problems related to the security of subsequent data retrieval.

  Even though the response to the search query has been done in a privacy preserving manner, the server can still learn what the result of the query was by simply observing what the client does next, e.g. **monitor the emails the client is going to access/download**.

Oblivious RAM

Oblivious RAM (ORAM)

- A cryptographic primitive originally designed to prevent reverse engineering by hiding access to memory.

- It has since repurposed for use in client-server scenarios with the purpose of hiding the ways in which data is accessed from the server.

ORAM security requirements

Hide **DATA CONTENTS** and:

1. Hide **which** data is accessed (e.g. My DSS course)
2. Hide **when** data was last accessed (e.g. 5mins ago)
3. Hide **how** data is accessed (i.e. read or write access)
4. Hide **how frequently** data is accessed (e.g. every day at 12pm)
5. Hide **access pattern** (e.g. sequential, random)

ORAM

- Uses symmetric encryption (e.g. AES) to encrypt small data structures (e.g. data ‘buckets’).

- Replaces read and write operations (i.e. download and upload) with a generic **access** operation which contains both a read and a write operation.

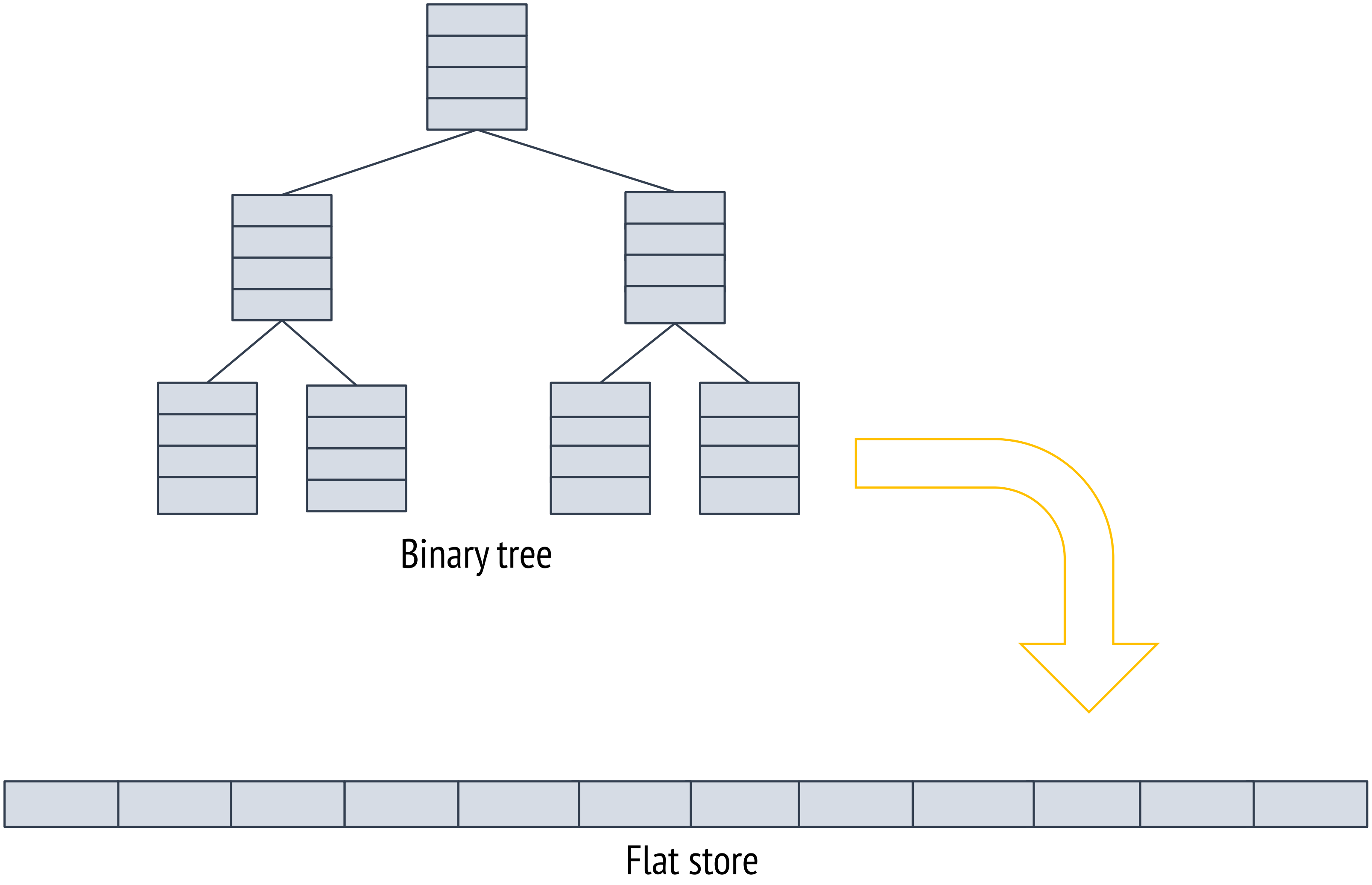
- The **access operation** has a significant overhead in order to disguise the exact data being accessed.

ORAM components

- Client stores an AES key
- Client stores a map
- Client stores a stash, i.e., a local cache structure

- Server storage is structured as a binary tree.
- On disk the three is stored as a flat data structure

Flat binary tree?



PathORAM

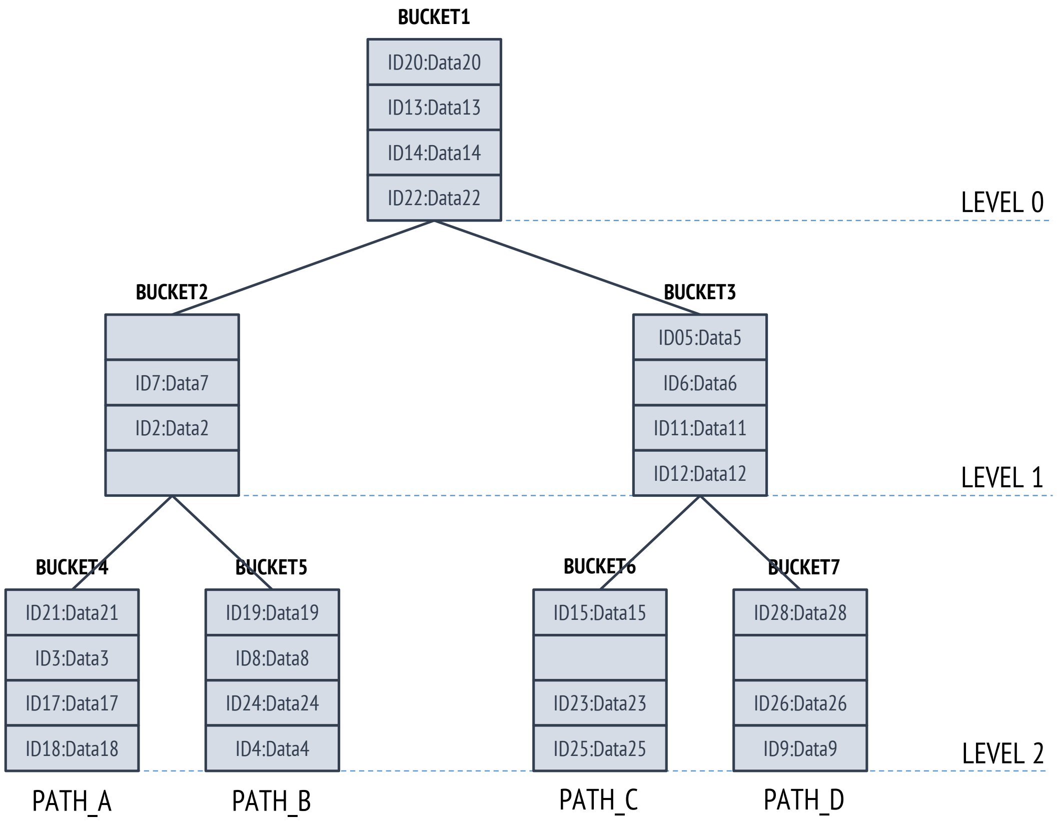
[Stefanov-van Dijk-Shi-Chan-Fletcher-Ren-Yu-Devadas13]

MAP

ID1	PATH_B
ID2	PATH_B
ID3	PATH_A
...	...
ID27	PATH_D
ID10	PATH_A
ID16	PATH_C
...	...

STASH

- ID1:Data1
- ID27:Data27
- ID10:Data10
- ID16:Data16



BUCKET1
- ID20:Data20
- ID13:Data13
- ID14:Data14
- ID22:Data22

LEVEL 0

BUCKET2
-
- ID7:Data7
- ID2:Data2
-

BUCKET3
- ID05:Data5
- ID6:Data6
- ID11:Data11
- ID12:Data12

LEVEL 1

BUCKET4
- ID21:Data21
- ID3:Data3
- ID17:Data17
- ID18:Data18

BUCKET5
- ID19:Data19
- ID8:Data8
- ID24:Data24
- ID4:Data4

BUCKET6
- ID15:Data15
-
- ID23:Data23
- ID25:Data25

BUCKET7
- ID28:Data28
-
- ID26:Data26
- ID9:Data9

LEVEL 2

PATH_A PATH_B PATH_C PATH_D

PathORAM access

READ:

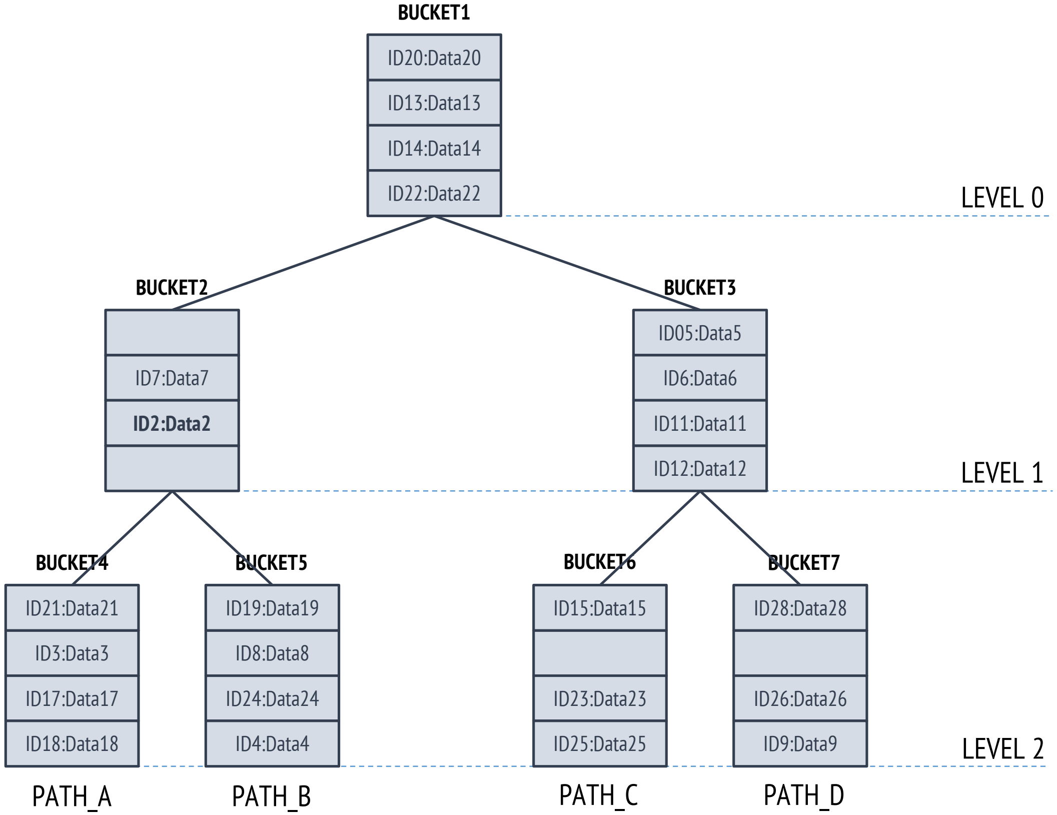
ID2:empty

MAP

ID2 → PATH_B

REQUEST:

PATH_B



PathORAM access

READ:

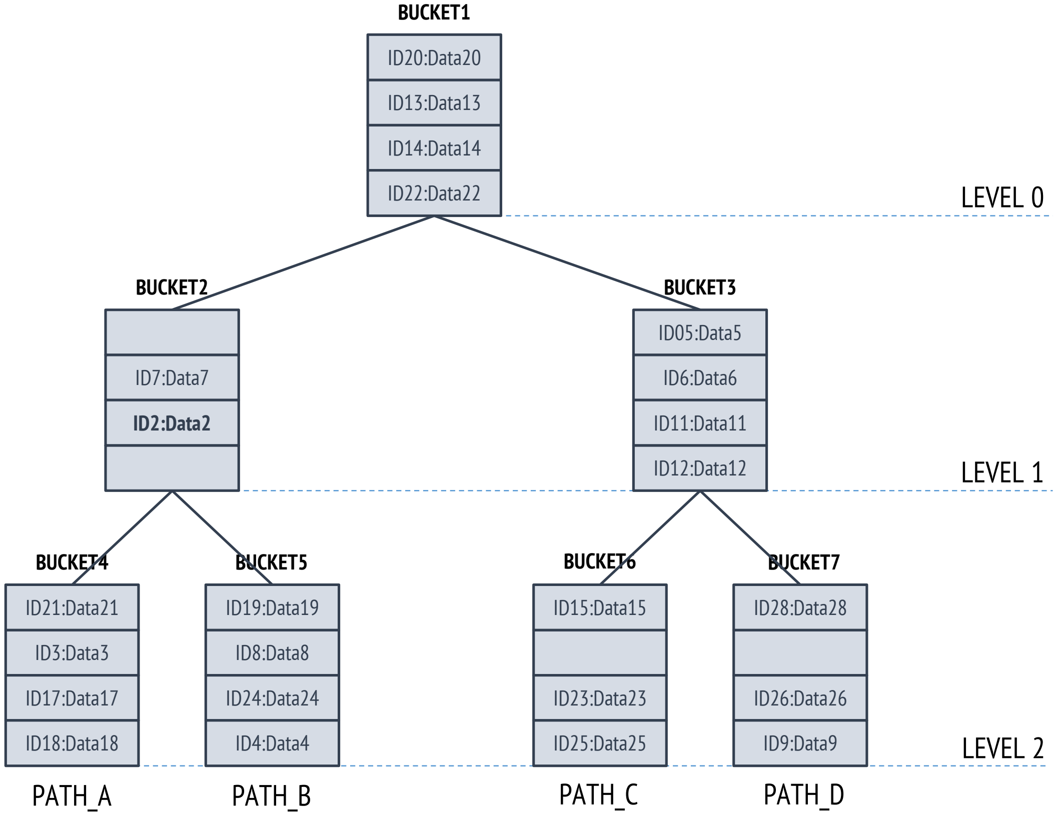
ID2:empty

MAP

ID2 → PATH_B

REQUEST:

PATH_B



BUCKET1

ID20:Data20
ID13:Data13
ID14:Data14
ID22:Data22

BUCKET2

ID7:Data7
ID2:Data2

BUCKET3

ID05:Data5
ID6:Data6
ID11:Data11
ID12:Data12

BUCKET4

ID21:Data21
ID3:Data3
ID17:Data17
ID18:Data18

BUCKET5

ID19:Data19
ID8:Data8
ID24:Data24
ID4:Data4

BUCKET6

ID15:Data15
ID23:Data23
ID25:Data25

BUCKET7

ID28:Data28
ID26:Data26
ID9:Data9

LEVEL 0
LEVEL 1
LEVEL 2

PATH_A
PATH_B
PATH_C
PATH_D

RECEIVE:

PATH_B											
Bucket5				Bucket2				Bucket1			
ID4:Data4	ID24:Data24	ID8:Data8	ID19:Data19		**ID2:Data2**	ID7:Data7		ID22:Data22	ID14:Data14	ID13:Data13	ID20:Data20

PathORAM access

READ:

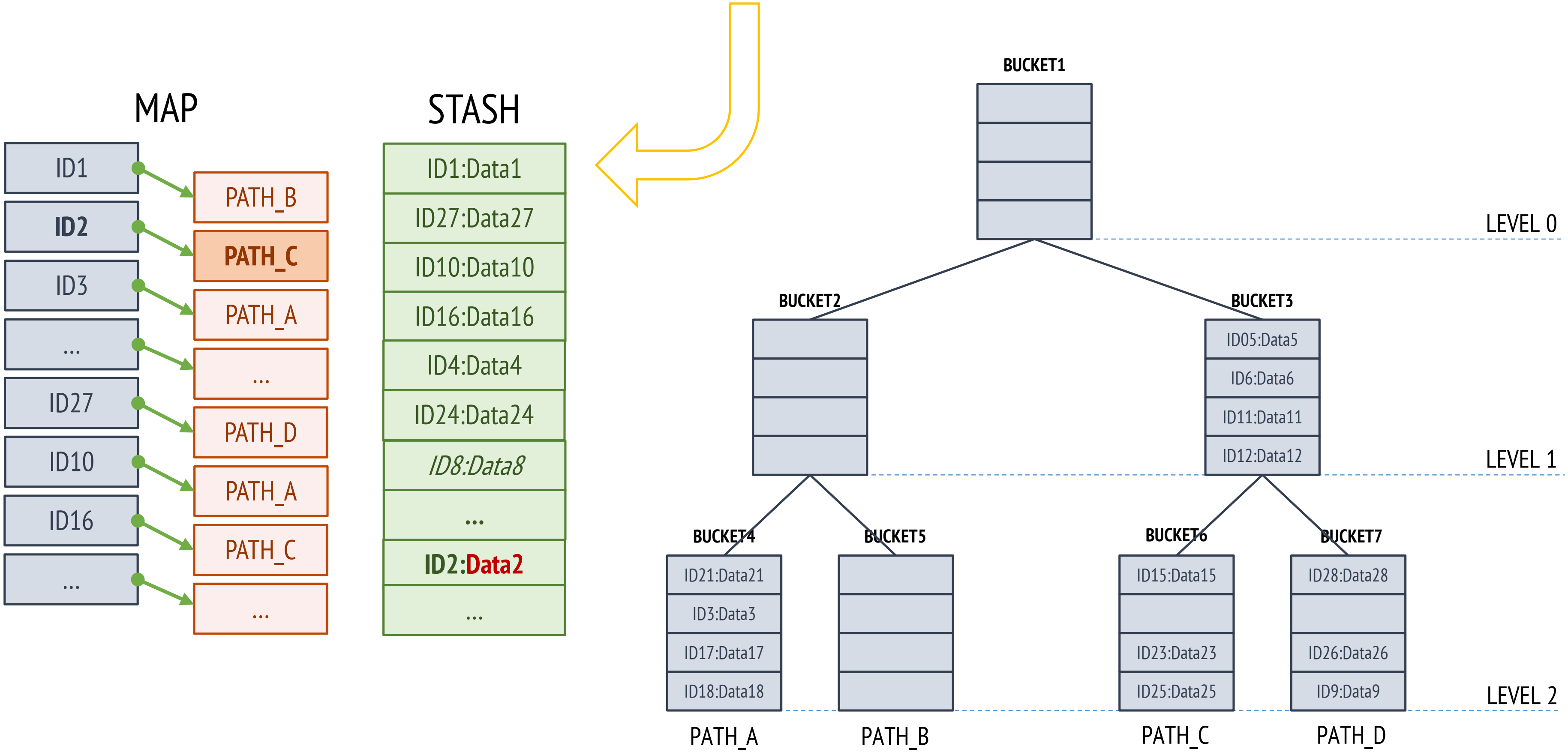
ID2:empty

REQUEST:

PATH_B

RECEIVE:

PATH_B											
Bucket5				Bucket2				Bucket1			
ID4:Data4	ID24:Data24	ID8:Data8	ID19:Data19		**ID2:Data2**	ID7:Data7		ID22:Data22	ID14:Data14	ID13:Data13	ID20:Data20



MAP

ID1	PATH_B
ID2	**PATH_C**
ID3	PATH_A
...	...
ID27	PATH_D
ID10	PATH_A
ID16	PATH_C
...	...

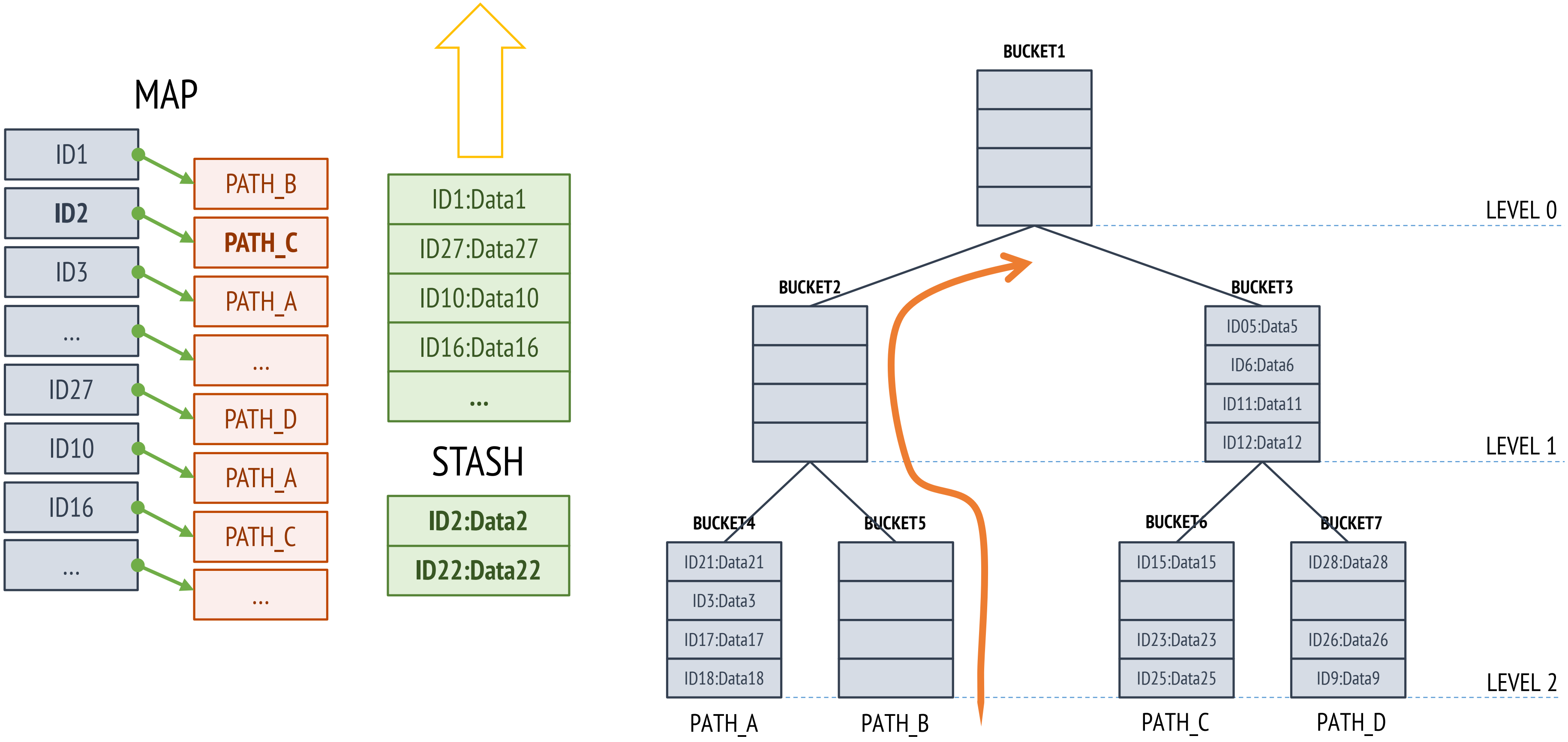
STASH

- ID1:Data1
- ID27:Data27
- ID10:Data10
- ID16:Data16
- ID4:Data4
- ID24:Data24
- *ID8:Data8*
- ...
- **ID2:Data2**
- ...

PathORAM access

WRITE:

PATH_B											
Bucket5				Bucket2				Bucket1			
ID1:Data1	ID24:Data24	ID19:Data19	ID8:Data8	ID7:Data7	**ID10:Data10**	ID20:Data20	ID13:Data13	**ID27:Data27**	ID8:Data8	ID4:Data4	**ID16:Data16**



MAP

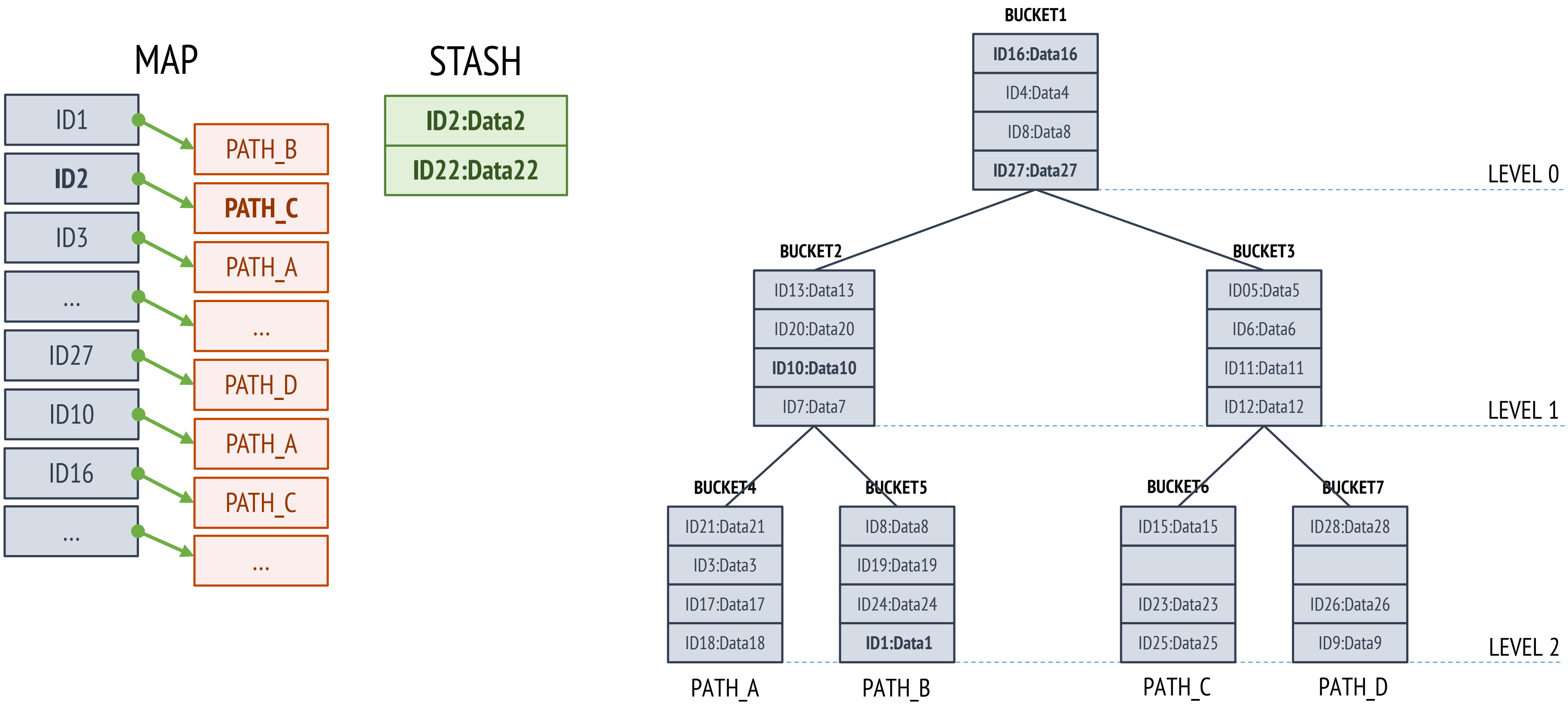
ID1	PATH_B
ID2	**PATH_C**
ID3	PATH_A
…	…
ID27	PATH_D
ID10	PATH_A
ID16	PATH_C
…	…

ID1:Data1
ID27:Data27
ID10:Data10
ID16:Data16
…

STASH

ID2:Data2
ID22:Data22

PathORAM structure



MAP

ID1	PATH_B
ID2	**PATH_C**
ID3	PATH_A
...	...
ID27	PATH_D
ID10	PATH_A
ID16	PATH_C
...	...

STASH

ID2:Data2
ID22:Data22

BUCKET1
ID16:Data16
ID4:Data4
ID8:Data8
ID27:Data27

LEVEL 0

BUCKET2
ID13:Data13
ID20:Data20
ID10:Data10
ID7:Data7

BUCKET3
ID05:Data5
ID6:Data6
ID11:Data11
ID12:Data12

LEVEL 1

BUCKET4
ID21:Data21
ID3:Data3
ID17:Data17
ID18:Data18

BUCKET5
ID8:Data8
ID19:Data19
ID24:Data24
ID1:Data1

BUCKET6
ID15:Data15

ID23:Data23
ID25:Data25

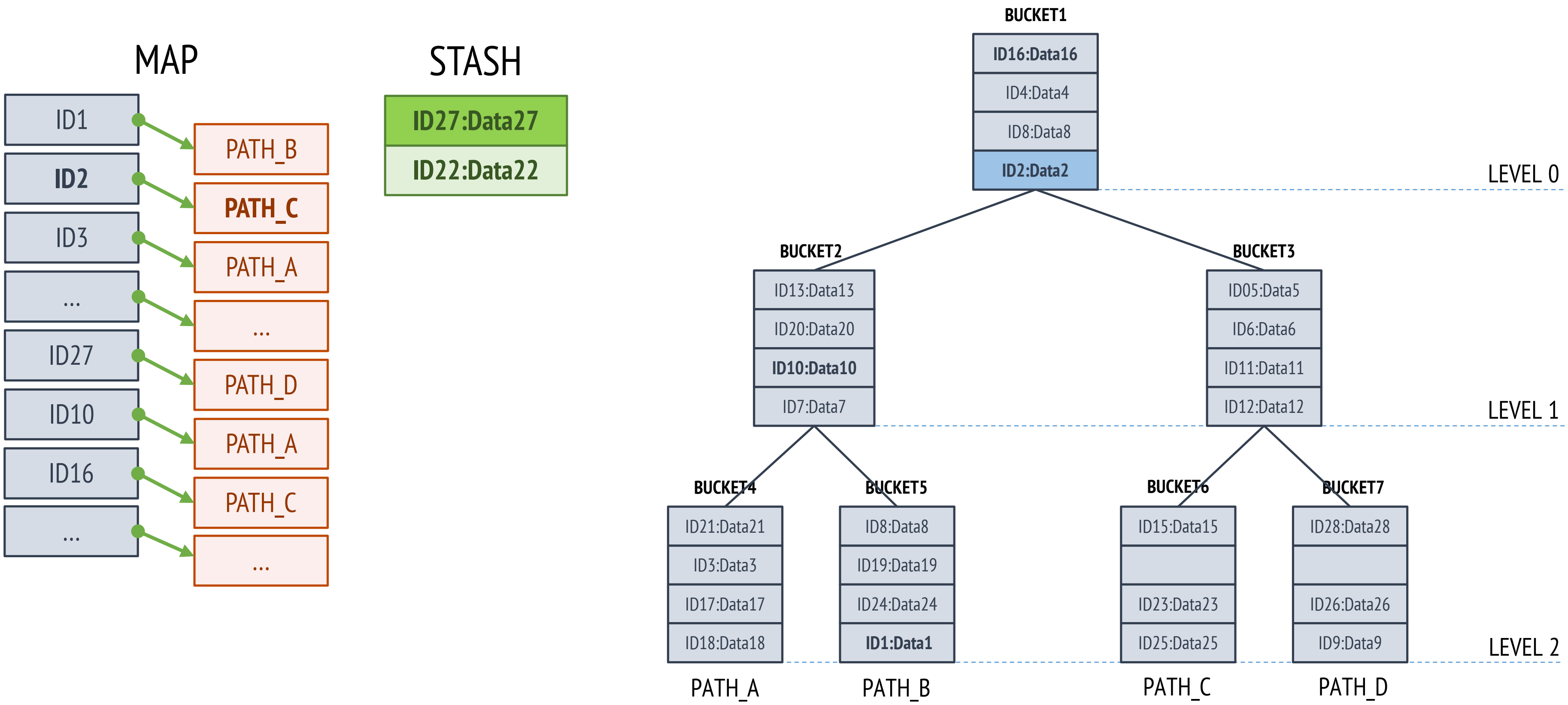
BUCKET7
ID28:Data28

ID26:Data26
ID9:Data9

LEVEL 2

PATH_A
PATH_B
PATH_C
PATH_D

PathORAM structure (alternative)



MAP

ID1	PATH_B
ID2	**PATH_C**
ID3	PATH_A
...	...
ID27	PATH_D
ID10	PATH_A
ID16	PATH_C
...	...

STASH

ID27:Data27
ID22:Data22

BUCKET1
ID16:Data16
ID4:Data4
ID8:Data8
ID2:Data2

LEVEL 0

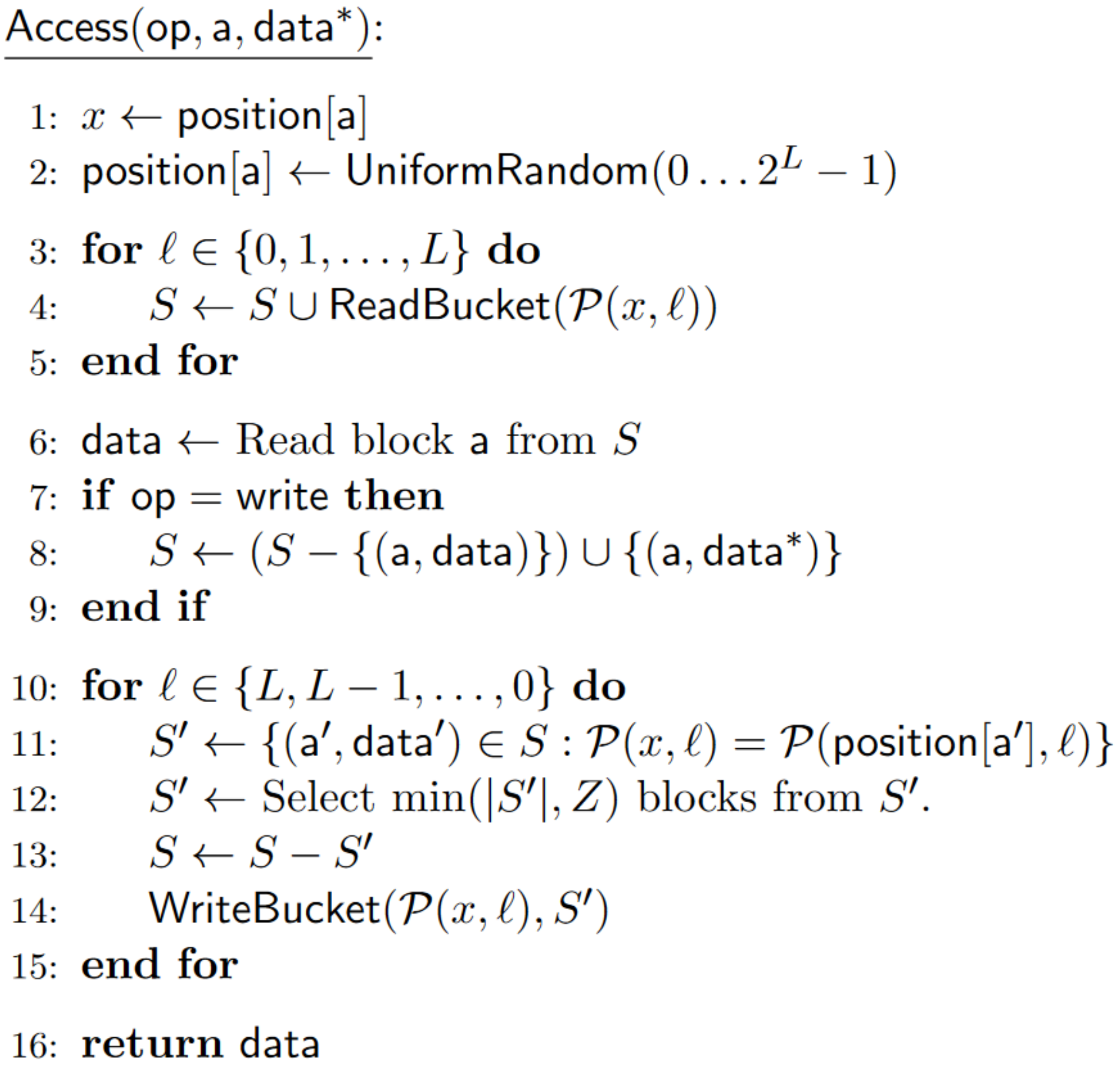
BUCKET2	**BUCKET3**
ID13:Data13	ID05:Data5
ID20:Data20	ID6:Data6
ID10:Data10	ID11:Data11
ID7:Data7	ID12:Data12

LEVEL 1

BUCKET4	**BUCKET5**	**BUCKET6**	**BUCKET7**
ID21:Data21	ID8:Data8	ID15:Data15	ID28:Data28
ID3:Data3	ID19:Data19		
ID17:Data17	ID24:Data24	ID23:Data23	ID26:Data26
ID18:Data18	**ID1:Data1**	ID25:Data25	ID9:Data9
PATH_A	PATH_B	PATH_C	PATH_D

LEVEL 2

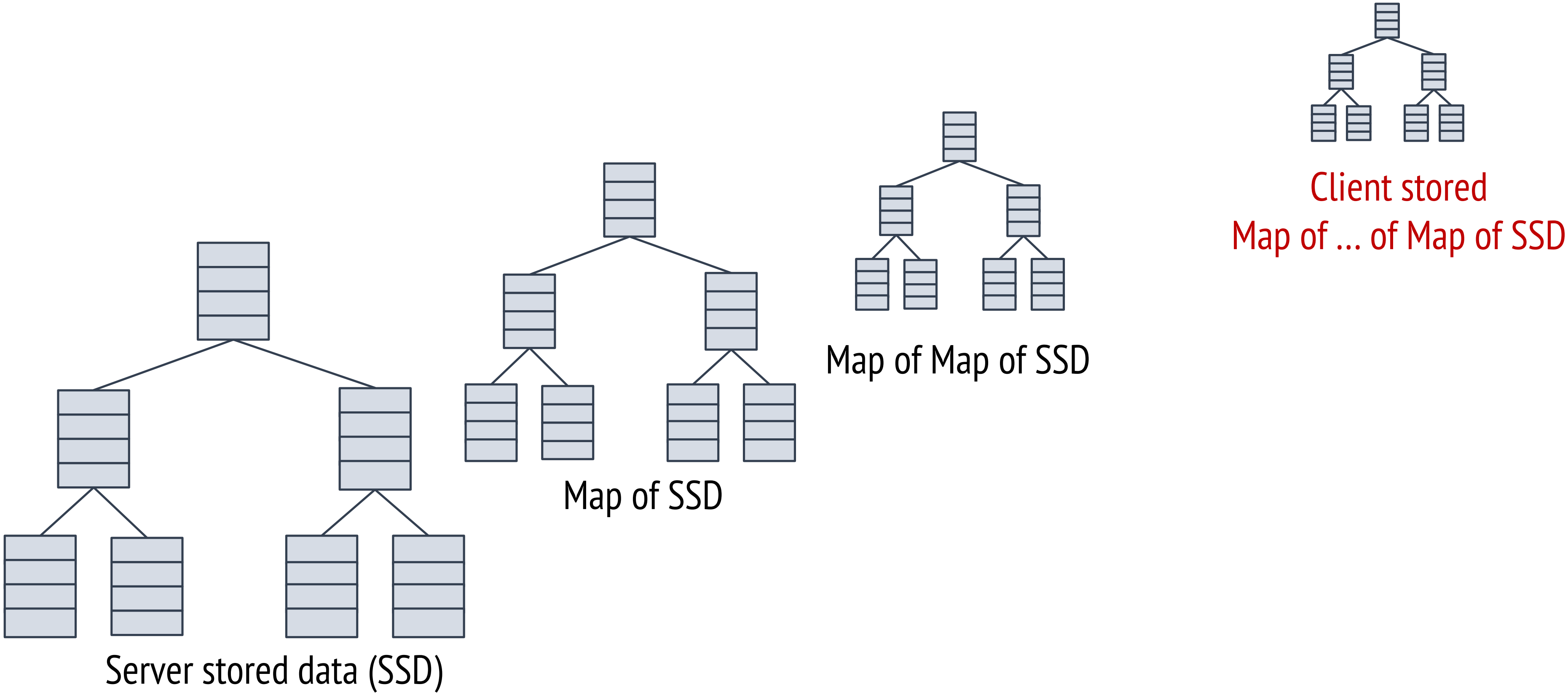
Algorithm



Access(op, a, data*):

```
$x \leftarrow$ position[a]
position[a] $\leftarrow$ UniformRandom($0 \ldots 2^L - 1$)
for $\ell \in \{0, 1, \ldots, L\}$ do
    $S \leftarrow S \cup$ ReadBucket($\mathcal{P}(x, \ell)$)
end for
data $\leftarrow$ Read block a from $S$
if op = write then
    $S \leftarrow (S - \{(\text{a}, \text{data})\}) \cup \{(\text{a}, \text{data}^*)\}$
end if
for $\ell \in \{L, L-1, \ldots, 0\}$ do
    $S' \leftarrow \{(\text{a}', \text{data}') \in S : \mathcal{P}(x, \ell) = \mathcal{P}(\text{position}[\text{a}'], \ell)\}$
    $S' \leftarrow$ Select $\min(|S'|, Z)$ blocks from $S'$.
    $S \leftarrow S - S'$
    WriteBucket($\mathcal{P}(x, \ell), S'$)
end for
return data
```

Recursive ORAM



Client stored
Map of … of Map of SSD

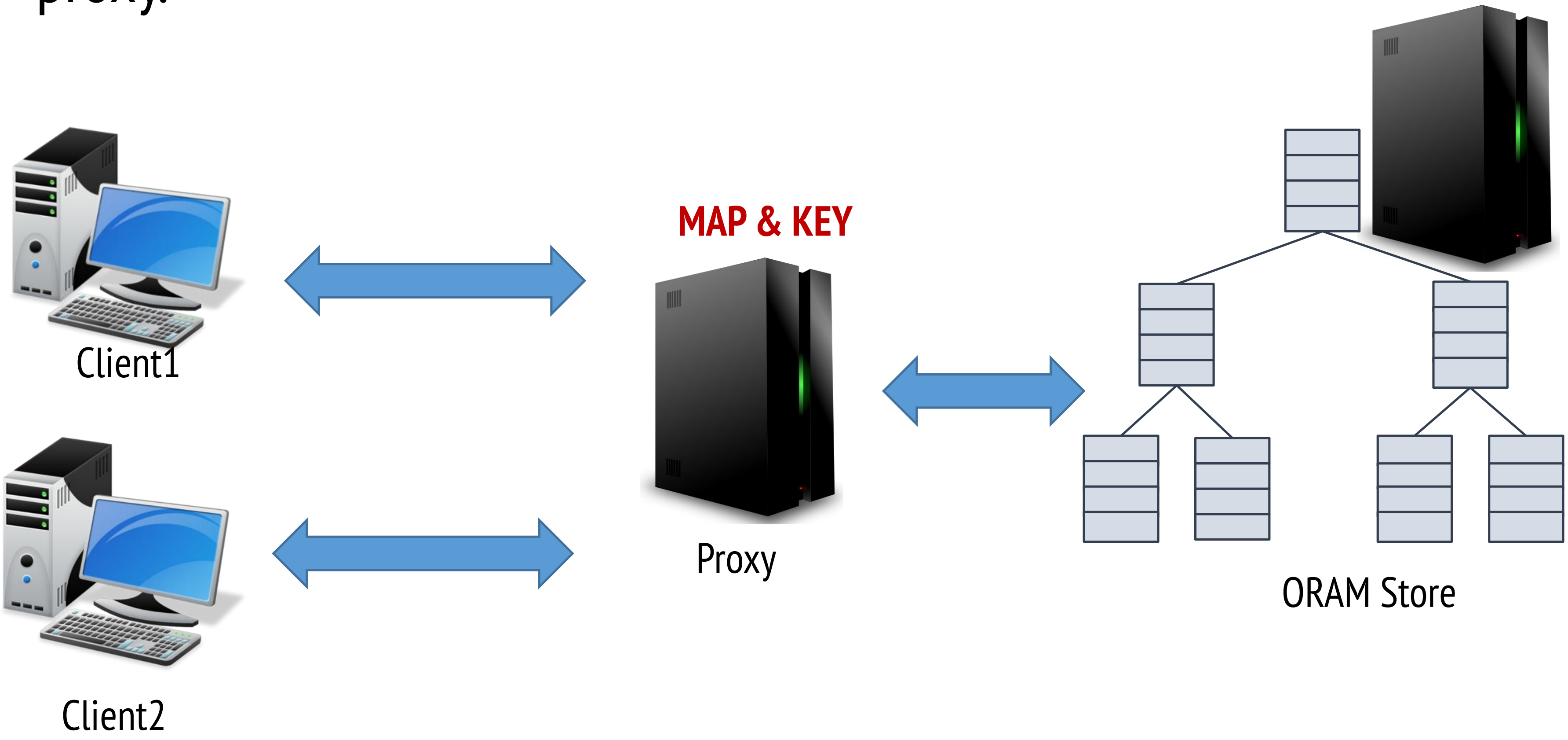
Map of Map of SSD

Map of SSD

Server stored data (SSD)

Other Limitations

PathORAM is limited to a single user. If multiple users require access to the store (server), access must be done through a proxy.



Other Limitations

If multiple users access the store timing attacks can be leveraged by the server with respect to

1. Proxy data CACHING
2. Proxy duplicating requests (e.g. Client1 and Client2 request same data)
3. Volume of data (e.g. Client1 wants more data than Client2)

PathORAM performance

Example

Assuming a 128GB database with:

- S = 64KB block size
- Z = 5 blocks per bucket
- L = 20 levels

`SecretDocument.txt`
a 1MB document stored in the database

PathORAM performance

Example

Assuming a 128GB database with:

- S = 64KB block size
- Z = 5 blocks per bucket
- L = 20 levels

What are the bandwidth requirements to access this document?

`SecretDocument.txt`

a 1MB document stored in the database

PathORAM performance

Example

Assuming a 128GB database with:

- S = 64KB block size
- Z = 5 blocks per bucket
- L = 20 levels

1MB = 1024KB
Block per document N:
N = 1024KB/64KB (size of the block) = 16

`SecretDocument.txt`
a 1MB document stored in the database

PathORAM performance

Example

Assuming a 128GB database with:

- S = 64KB block size
- Z = 5 blocks per bucket
- L = 20 levels

`SecretDocument.txt`
a 1MB document stored in the database

1MB = 1024KB
Block per document N:
N = 1024KB/64KB (size of the block) = 16

To send/receive ONE document
PathORAM requires: N*S*Z*L = 100MB

ORAM applications

- Personal health records
- Credit score systems
- GENOME related research
- As a private information retrieval (PIR) protocol

ORAM vs. Searchable Encryption

ORAM

- Provides anonymous access to data blocks
- Used as a private information retrieval (PIR) protocol
- Fully protects access patterns and data contents
- Requires a considerable overheads

Searchable encryption

- Enables users to securely search a precomputed index
- Used to efficiently **locate** data in databases
- Protects search terms and search results
- Only protects search patterns