FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

> TECHNIK

MASTERSTUDIUM:
> IT-SECURITY

Searchitect

Ein Developerframework für Searchable Encryption

Ines Mathias Tausig Silvie Schmidt
Competence Centre for IT-Security
FH Campus Wien

https://www.searchitect.eu

Netidee

Searchitect was generously funded by the the *Netidee* program of the *Internet Privatstiftung Austria (IPA)* and inspired by its *Privacy by design* category.

Issues

- Data is stored online, i.e. trust in cloud providers?
- How secure is your data?
- What about privacy?

Idea

- Letś encrypt our data!
- **Problem:** how can we search in our data then?

Searchable Encryption

Idea: Encrypted data should be searchable.

- User should be able to search for a certain *keyword* - without having to decrypt the data.
- The server may not learn anything about the data.

Problem: There are no usable schemes or frameworks for developing.

Searchitect

Enter **Searchitect**, a developer framework for searchable encryption.
Letś take a closer look at *Searchable Encryption*

Types of SE I

Full Domain search

- 2000: first SE scheme by Dawn Song et al.

It encrypted the documents in a specific way, such that a search *directly* on the ciphertext was made possible.

Types of SE II

Index based schemes

- 2003: first *index based* SE scheme by Goh et al.

It decouples the search from the storage of the actual data by creating a *search index* for all documents.

```
{
   "doc1": ["lorem","ipsum","dolor","rosat", ...],
   "doc2": ["Rock","Soul","Disco","Jazz", ...],
   "doc3": ["easterhegg","hacktoberfest","defcon", ...]
   ...
}
```

Types of SE III

Inverted Index

- 2006: first *inverted index* by Curtmola et al.

For efficiency reasons, all current schemes are based on an *inverted index*.

```
{
   "privacy": ["doc42", "doc312"],
   "Rock": ["doc43","doc7","doc11","doc312", ...],
   "defcon": ["doc23", "doc7", "doc101", ...],
   ...
}
```

Scheme properties

- **Multi User**: Multiple users (who might not have access to the actual content) can query the index
- **Dynamic scheme**: An index can be updated, after its creation (files can be added and removed)
- **Asymmetric scheme**: Different keys (held by different users) can be used to create/update the index and to query it. Uses *pairing based cryptography* which still has performance issues

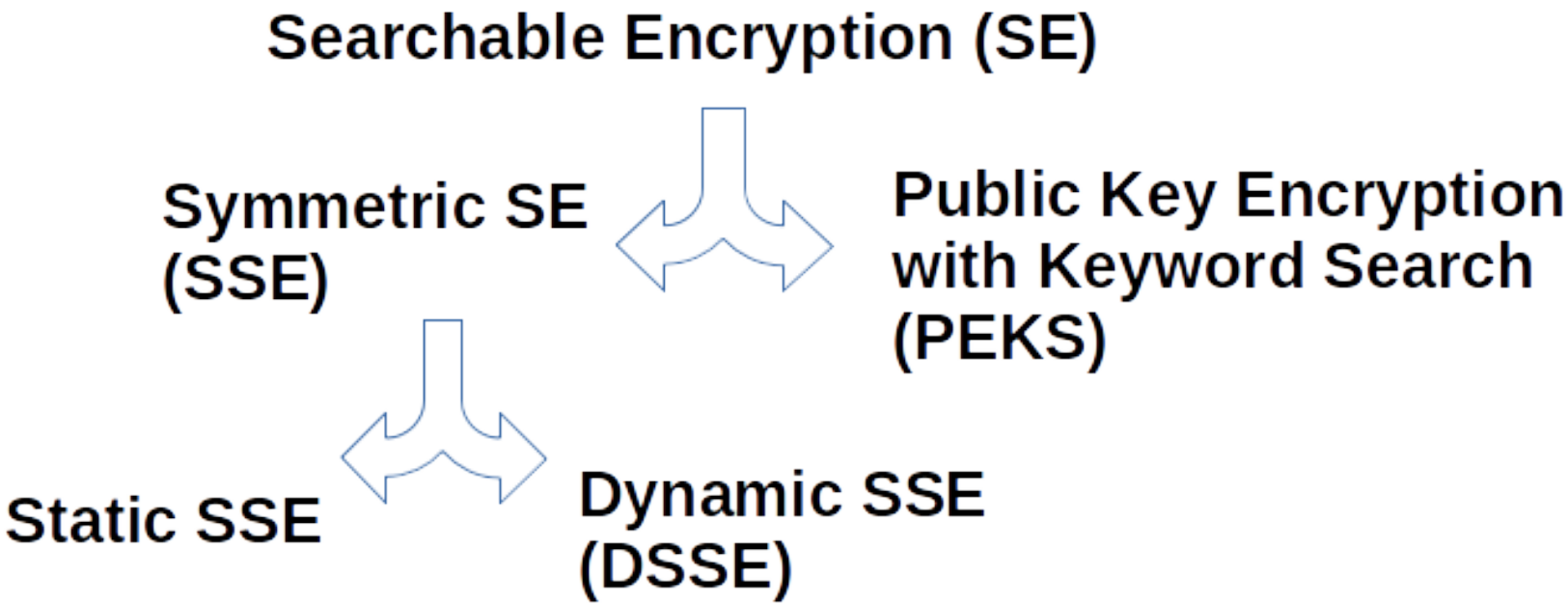
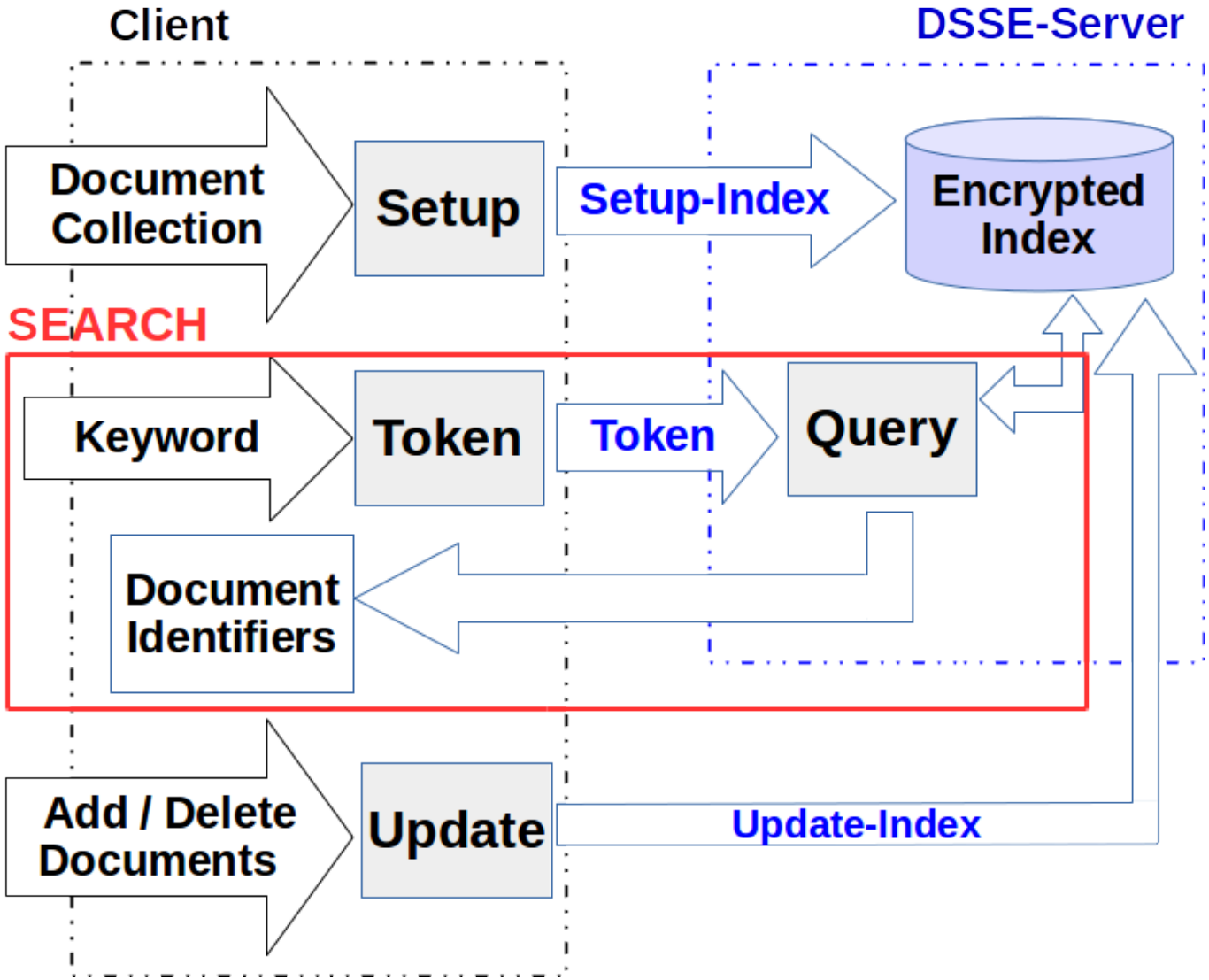


Figure: Types of SE schemes


Client
DSSE-Server
Document Collection
Setup
Setup-Index
Encrypted Index
SEARCH
Keyword
Token
Token
Query
Document Identifiers
Add / Delete Documents
Update
Update-Index

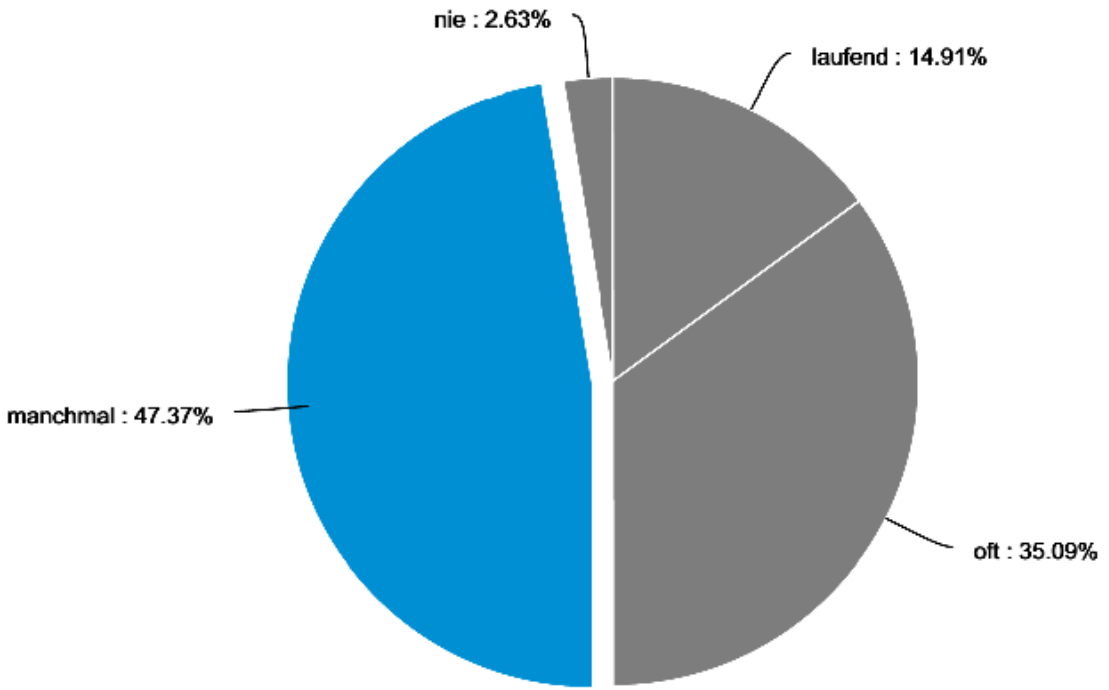
Security problems

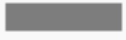
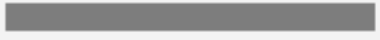
- **Hidden search**: The server should not learn, which words are queried (use encrypted search tokens)
- **Controlled search**: The server should not be able to query for words by itself (creating the search tokens requires a key unknown to the server)
- **Forward Security**: Server should not be able to tell if a newly inserted file matches a previous query

Use Cases

- Cloud storage
- Enrcypted E-Mails
- Backup Software
- Document Management

Wie häufig suchen Sie in Ihren gespeicherten Daten?

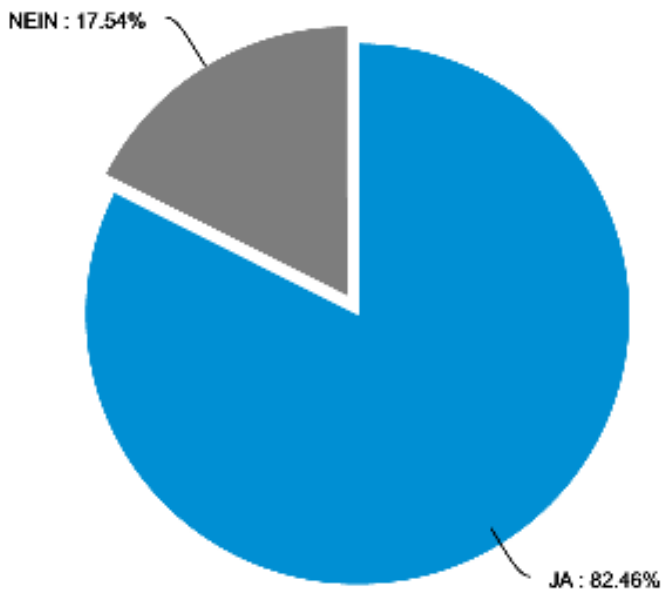


Answer	Count	Percent	20% 40%
Verwenden Sie die Suchfunktion in ihrer Backup-Software?	26	9.92%	
Verwenden Sie die Suchfunktion in ihrem Email-Client?	94	35.88%	
Verwenden Sie die Suchfunktion in ihrer File Synchronisation?	31	11.83%	
Verwenden Sie die Suchfunktion in ihrer Dokumentverwaltung?	92	35.11%	
Verwenden Sie die Suchfunktion in ihrer Festplattenverschlüsselungssoftware?	16	6.11%	
Andere Applikationen, in denen Sie die Suchfunktion verwenden:	3	1.15%	
Total	262	100 %	

Kreuzen Sie bitte Zutreffendes an: - Text Data for Andere Applikationen, in denen Sie die Suchfunktion verwenden:

08/18/2018	1604588922	Apps
08/08/2018	1604528763	Keine Suche
07/17/2018	1604464240	Browser (Verlauf)

Würden Sie eine Technologie zur Verschlüsselung von Daten die gleichzeitig eine Suchfunktion bietet nutzen?



Feasibility Problems

Schemes beginning to mature, but it's still *"Fancy crypto"* ...

- Hard to understand
- Implementations not readily available
- No standardizations

Algorithms

The schemes we recommend to use in Searchitect are **Dyn2Lev**[1] and **Sophos**[2], which are the only currently available efficient, dynamic and forward secure schemes. The former requires more storage space and bandwidth while the latter needs more computational resources.

[1] Cash et.al., 2014
[2] Bost et.al. 2016

Searchitect

A framework which hides the cryptographic complexity from the developers

- Server which processes the queries
- Client library allowing (kind of) easy integration in applications
- Plugin based architecture to integrate new schemes

Also: Stable environment for performance tests and evaluation of new schemes

Searchitect- Server side

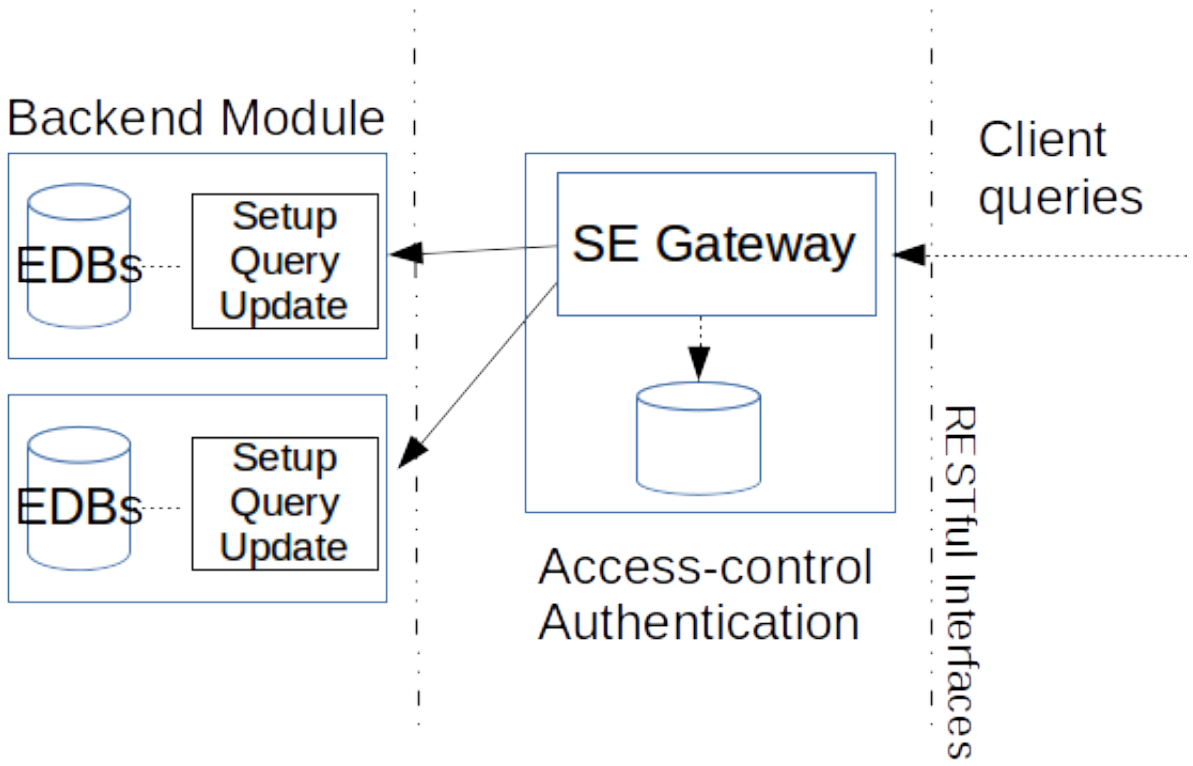


Figure: Microservices architecture

Searchitect- Client side

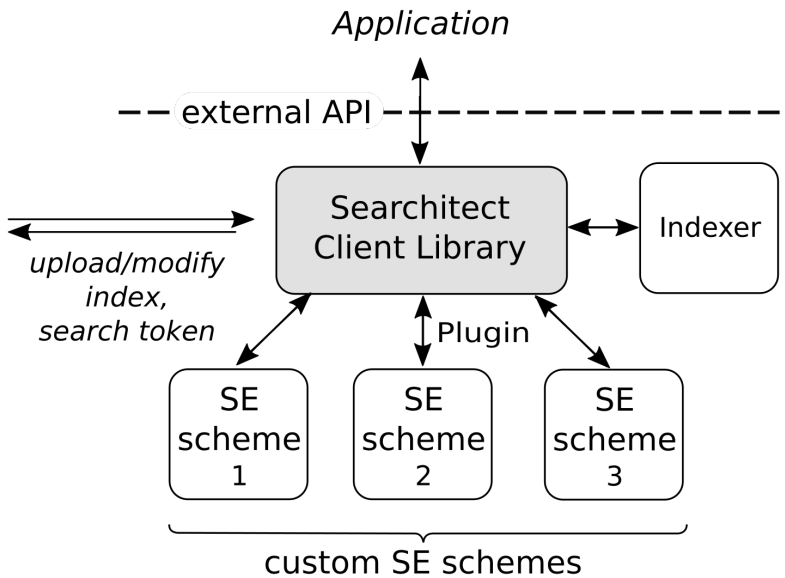


Figure: Searchitect-client

Searchitect - Openess

- Webservice API in RESTful design with JSON formatted transport data
- Integration of a new searchable encryption scheme by adopting 3 project parts.

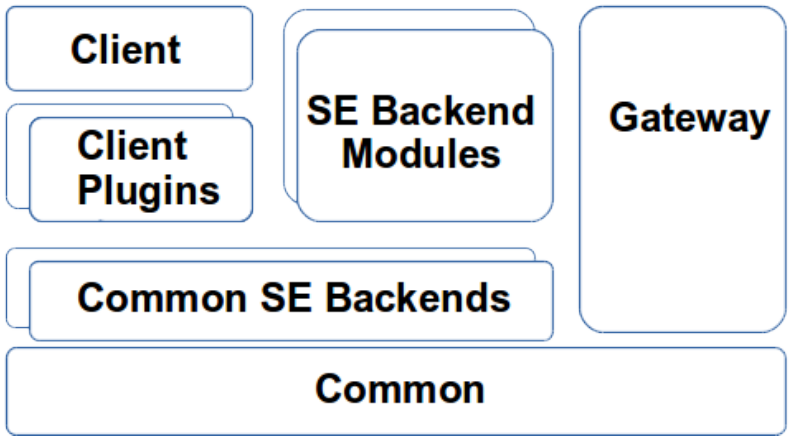


Figure: Searchitect software design

Implementation - Technologies

- Spring Boot - embedded Tomcat + easy configuration
- Docker - easy deployment
- Json Web Tokens - REST authentication
- Swagger - interface description
- RocksDB - embedded, persistent and fast key/value store
- Clusion library - Apache Lucene indexer

Start Docker container and show swagger API.

Integrated SE schemes I

Dyn2Lev [3] / Clusion Library[4]

- Static and dynamic inverted index
- Resource hiding and forward secure
- Optimization to DynRH2LevRocks

Static Variants	**DynRH2Lev**	**DynRH2LevRocks**
Storage	Memory based	persisted - RocksDB
Length of encrypted id	100 Bytes	36 Bytes
Seed of random Bytes (IV)	each time	once per keyword
Operations on	Strings	Bytes

Integrated SE schemes II

Sophos - Bost 2016

- Forward security achieved using trapdoor permutation - asymmetric primitive
- Server cannot link update tokens

Dynamic Variants	**DynRH2LevRocks**	**Sophos**
Storage	RocksDB + client state	
Resource	Hiding	Revealing
Id Encryption	2 x AES_CTR	XOR $H_K(ST_i)$
Label Derivation	Keyed hash	Trapdoor permutation
FS-Search	Bandwidth	Calculation effort

[3] Cash et al. 2014

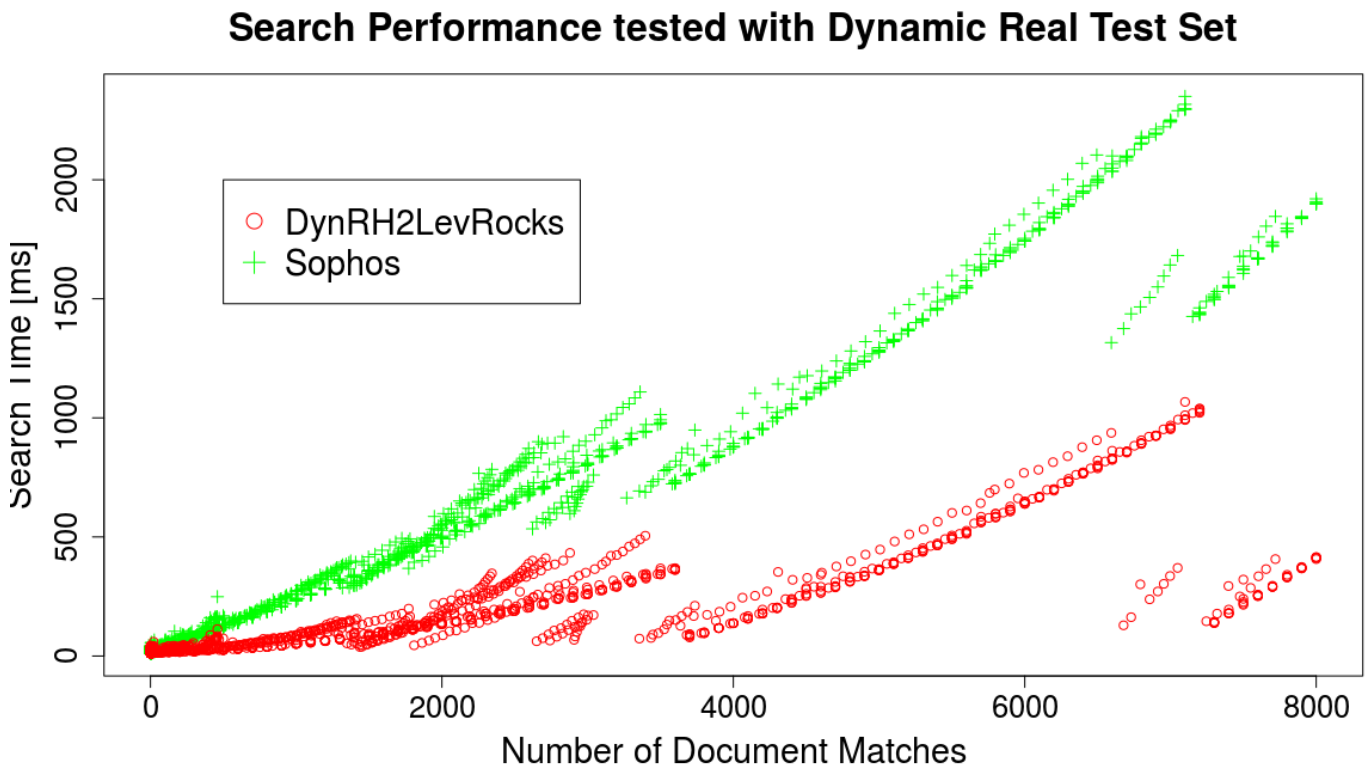
[4] Kamara and Moataz 2017

Test Simulation

Simulation use case: backup program

- **Static tests:** tests performance of Setup, Search protocol in a simulation of full data backups
- **Dynamic tests:** tests performance of Update and Search protocol and the storage space usage in a simulation of incremental data backups

Search - Sophos vs. DynRH2LevRocks



Searchitect-Integration

Idea: The encrypted search is accomplished independent from the document encryption. This allows the use of SE in applications without client/server architecture, such as backup programs.

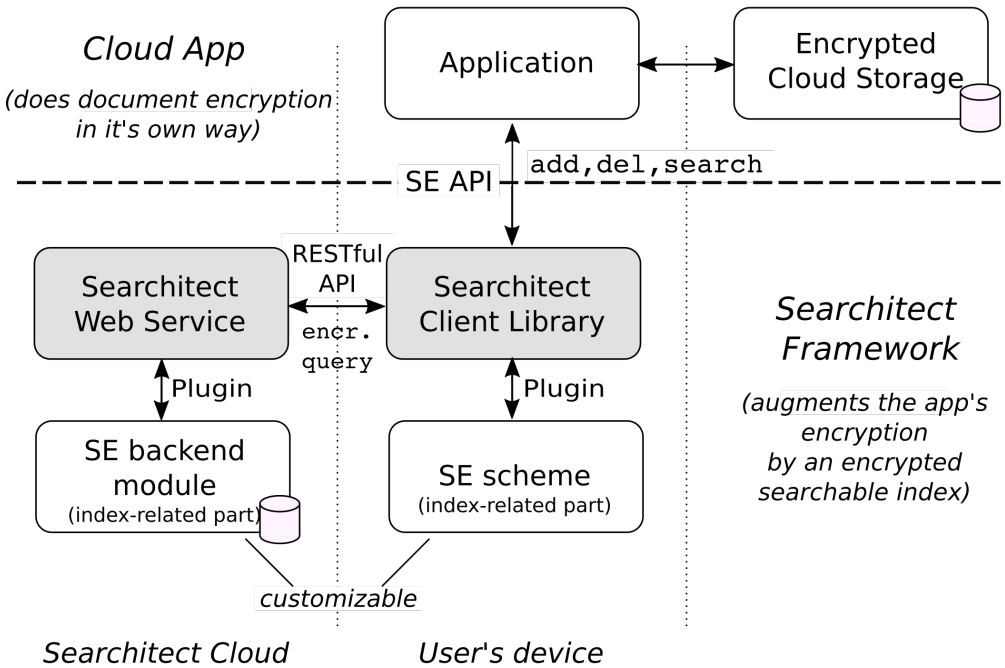


Figure: Searchitect-architecture

Example-Integration

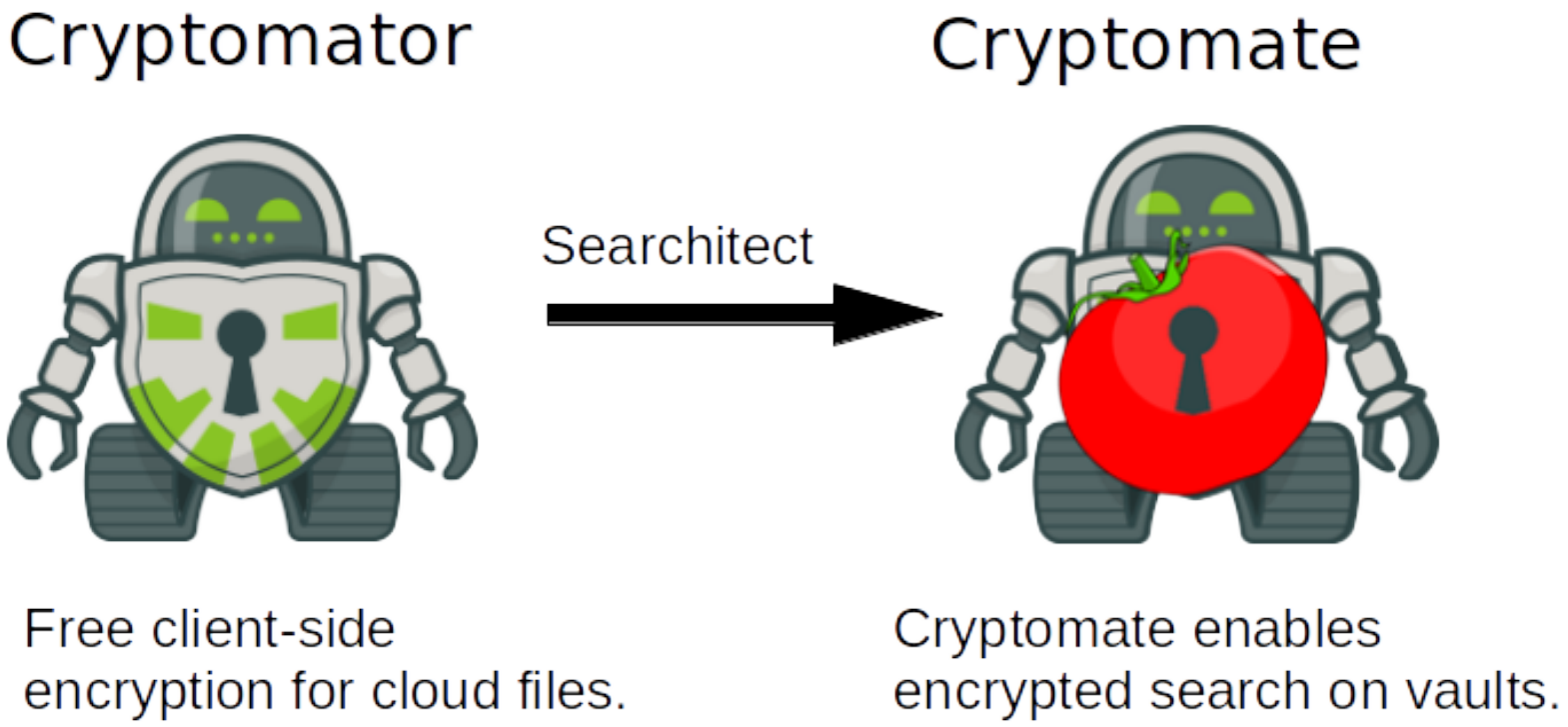


Figure: Image source: `https://cryptomator.org`

Show cryptomator search functionality.

QUESTIONS?

Contact

- https://www.searchitect.eu
- https://gitlab.com/searchitect