Searchitect - A Developer Framework for Hybrid Searchable Encryption (Position Paper)

Ulrich Haböck, Manuel Koschuch, Ines Kramer, Silvie Schmidt and Mathias Tausig
Competence Centre for IT Security, University of Applied Sciences, FH Campus Wien, Vienna, Austria

Keywords: Symmetric Searchable Encryption, Hybrid Encryption, Privacy Enhancing Technologies, Developer Framework.

Abstract: In light of the trend towards cloud-based applications, privacy enhancing technologies are becoming increasingly important. Searchable encryption (SE) allows to outsource data to the cloud in a secure way, whilst permitting search functionality on that encrypted data: the host is able to perform search queries on behalf of the user, but without having access to the encryption keys. We propose Searchitect, a developer framework which allows to enhance existing cloud-based applications with searchable encryption. Searchitect provides a ready-to-use client-server infrastructure, which is expandable by custom SE schemes, the server being a configurable webservice offering searchable encryption as a service (SEaaS). Unlike other searchable encryption frameworks our approach is hybrid: Searchitect separates the index component from the data encryption scheme, leaving the application's own specific encryption paradigm and access control untouched. In this way, we hope to ease the integration of searchable encryption into already existing cloud-based applications, requiring only the client code to be modified. Further, as searchable encryption is a very active field of research, we emphasize the experimental character of Searchitect's framework. It aims at developers keeping track of recent SE developments, providing an easy deployable solution for testing in public and private clouds.

1 INTRODUCTION

Supported by the trend towards mobile and web applications, outsourcing data to the cloud is becoming an increasingly popular solution to provide access from and to multiple devices or users. In respect of data breaches reported by the news on an almost daily basis, e.g. Anthem's 80 Million (The Guardian, 2015), Yahoo's 1 Billion (New York Times, 2016), or the more recent Equifax breach (Cision PR Newswire, 2017), securing cloud resources from unauthorized access poses a serious challenge. As a primary countermeasure, many cloud-based services encrypt user's data not only in transit but also at rest, hosted in the cloud. In general, data does not allow to perform meaningful operations on it while encrypted. Whenever a user wants to benefit from services which operate on her data, providing access to her encryption keys (or having the cloud-storage provider decrypt the data) is the only convenient solution to most current approaches. However, such a step has to be considered carefully: on the one hand with respect to the provider's capability to meet certain security requirements (i.e., server trust), and on the other hand concerning privacy issues in case of key leakage or third-party access.

Searchable encryption (SE) is a privacy enhancing technology which allows a server to search on encrypted data without having access to plaintext information or the encryption key itself. The server knows *what* it does, i.e. processing an encrypted search query over encrypted data (mostly an index) resulting in an encrypted outcome, but *does not learn* anything valuable about neither the query's nor the document's *content*. Since the pioneering work of Song, Wagner and Perrig (Song et al., 2000), dozens of SE schemes have been designed trying to balance usability, security, efficiency, and scalability, see (Bösch et al., 2014) or the more recent position paper (Asghar et al., 2017) for a comparative overview. In spite of the special attention received by academic research, reception of these schemes in real world applications is still poor. All SE schemes providing multi user access are at least of linear complexity in the number of encrypted items, hence being impracticable at large scale. Also *Searchable Symmetric Encryption (SSE)* schemes, which cover the single user setting, vary largely with respect to efficiency and compact-

Haböck, U., Koschuch, M., Kramer, I., Schmidt, S. and Tausig, M.
Searchitect - A Developer Framework for Hybrid Searchable Encryption (Position Paper).
DOI: 10.5220/0006789402910298
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 291-298
ISBN: 978-989-758-296-7

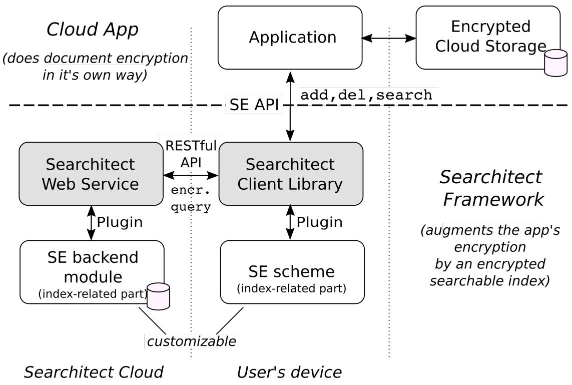
Cloud App
(does document encryption in it's own way)
Application
Encrypted Cloud Storage
add,del,search
SE API
Searchitect Web Service
RESTful API
encr. query
Searchitect Client Library
Searchitect Framework
(augments the app's encryption by an encrypted searchable index)
Plugin
Plugin
SE backend module (index-related part)
SE scheme (index-related part)
customizable
Searchitect Cloud
User's device

Figure 1: The Searchitect framework, integrated in an arbitrary end-to-end secured cloud application. Searchitect acts as an abstraction layer between the application and (the index-related components of) the custom SE scheme. In this hybrid way the cloud application is complemented by search functionality, leaving it's own way to handle document encryption untouched.

ness. Furthermore, many published schemes are in a rather experimental state and should be used with caution: although supplied by rigorous security proofs, the threat model considered is often too restrictive to capture all consequences of information leakage as shown by (Zhang et al., 2016), (Liu et al., 2014), (Naveed et al., 2015), (Cash et al., 2015), (Liu et al., 2014), and (Grubbs et al., 2016). From the viewpoint of software developers, the situation is even less satisfactory. Among sufficiently versatile schemes with proven security, only few publications report concrete performance numbers or make their prototype code accessible to the public. To the best of our knowlegde, the only open source software libraries implementing state of the art single user schemes are

- the Clusion Library (Encrypted Systems Lab, 2017), an SSE Java library which provides variations of the schemes from (Cash et al., 2014) and (Kamara and Moataz, 2017), including a distributed version of the former based on MapReduce/HADOOP for Amazon AWS,
- the Open SSE library (Bost, 2017), a C++ library which implements forward and backward secure SSE schemes from (Bost, 2016) and (Bost et al., 2017).

Altough there exist experimental SE frameworks such as Mylar (Popa et al., 2014a), CryptDB (Popa et al., 2011a), or CloudCryptoSearch (Ferreira and Domingos, 2013), they either aim at the integration of searchable encryption into specific environments, are not designed to be flexible in their SE backend, or do not provide comprehensive documentation. In this position paper, we propose *Searchitect*, a developer framework which provides a ready-to-use client-server architecture for index-based searchable symmetric encryption.

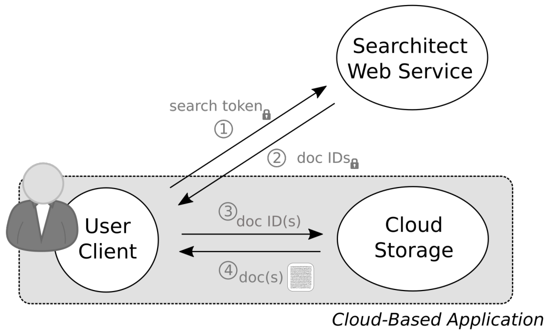

Figure 2: Searchitect's system model for searches. The search over encrypted indexes is separated from the application's own way to encrypt and host the data collection. Searchitect's web service can be either hosted independently or outsourced to our Searchitect cloud.

1. As a *hybrid* solution, cf. Figure 1 and 2, Searchitect integrates search functionality by not interfering with the application's own way to handle end-to-end encryption and access control of plaintext documents. Instead, Searchitect *complements* the application by a customizable infrastructure for *searchable index encryption*, the server-side part of which can (but not necessarily has to) be entirely separated from the application's cloud storage. This allows to apply our framework to environments with restricted access to server-side code, e.g. a cloud backup (or any other) program that just utilizes (arbitrary) cloud storages. Whenever such separation is desired, Searchitect's server-side part can be outsourced to our proposed *Searchitect cloud*, offering search over encrypted data as a service.
2. As an *extensible* solution, Searchitect is not restricted to any specific index-based schemes. Instead, we foster developers to implement schemes of their own choice and provide testing and evaluation features within our framework.
3. As an *open source* solution, client and server-side code will be made available under an *Open Source Initiative* (OSI[1]) approved open source license. Searchitect will be supplemented by a comprehensive documentation, including guidelines and implementation examples to serve the needs of application developers, who do not have specific knowledge of cryptography.

Due to the experimental state of searchable encryption in general, we strongly emphasize that our (as

[1] https://opensource.org/

any other) SE framework should be used in prototype software projects only. Until current reasearch has converged, Searchitect might help developers keeping track of recent SE developments, providing an easily deployable solution for testing in public and private clouds.

The remainder of this paper is organized as follows. In Section 2 we shortly discuss related frameworks and clarify how our approach differs from them. Section 3 surveys fundamental notions and properties of searchable encryption. Section 4 forms the core of our proposal: it provides a detailed description of the Searchitect framework, including implementation details. Finally, we shortly discuss application use cases and future work in our summary Section 5.

2 RELATED WORK

CryptDB (Popa et al., 2011a), (Popa et al., 2011b), is a multi-user system that provides an SQL-aware encryption layer on top of SQL databases. To enable SQL queries over encrypted data, the authors combine several cryptosystems, each covering a different set of SQL requirements. In particular, keyword search is achieved by the word-based SSE scheme of (Song et al., 2000). However, CryptDB is not under active development anymore and considered to be broken: Due to the deterministic character of CryptDB's encryption (Naveed et al., 2015) were able to recover more than 60% of patient records from electronic medical databases based on static frequency analysis.

Mylar (Popa et al., 2014a), (Popa et al., 2014b), is an experimental framework that allows almost seamless integration of SSE into web applications that are built on Meteor (Meteor Development Group Inc., 2014), a full-stack JavaScript development platform for web and mobile applications. As an experimental framework, Mylar is open source and is well documented, including implementation examples. However, Mylar uses its own multikey SE scheme which is vulnerable to client-server collusion: (Grubbs et al., 2016) show that under such a collusion, the server can perform brute-force dictionary attacks on any past, present, or future search queries. However, this attack is disputed by the Mylar research team, (Popa et al., 2014b).

CloudCryptoSearch (Ferreira and Domingos, 2013) is a less recognized encryption middle-ware for cloud-based data storage. Though being generic in its architecture, CloudCryptoSearch comes with its own SE scheme composed of homomorphic encryption techniques combined with dynamic indexing mechanisms. The follow-up frameworks IES-CBIR (Ferreira et al., 2015) and MIE (Ferreira et al., 2017) concentrate on searchable encryption of images. However, these three frameworks lack rigorous security proofs of their encryption schemes, as well as a thorough documentation that helps to ease the integration into software projects.

All above mentioned frameworks follow the canonical system design, in which searches are served by the same (cloud) entity that holds the encrypted data collection. Searchitect's paradigm is different, as explained in Section 1, separating the search over encrypted indexes entirely from the application's own way to encrypt and access plaintext documents. This allows an easy integration of searchable encryption into environments with restricted access to server-side source code, for example a cloud backup/storage program that only utilizes cloud storages, cf. Section 5. Further, Searchitect is designed to be extensible in its cryptographic backend, and its web service comes with several logging and measurement features, cf. Section 4.1. In this way we hope to support developers with testing and evaluating SSE schemes of their choice under real-world circumstances.

3 BASICS OF SEARCHABLE ENCRYPTION

Searchable encryption (SE) is a cryptographic technique that allows to outsource searching over a private document base to an untrusted proxy in such a way that as little information as possible is leaked. In its most restrictive setting a single user, the *owner* of the document base, is allowed to *write* the outsourced document storage, perform (encrypted) search queries and access requested documents (all of which we summarize by *read*). Depending on the *access profile* of the scheme, defining which entity is allowed to write and/or read, the scheme uses either symmetric or public key primitives, dividing SE in the two classes

- Searchable Symmetric Encryption (SSE), and
- Public Key Encryption with Keyword Search (PEKS).

As in (Bösch et al., 2014), we distinguish between the basic access profiles

- single writer / single reader (S/S),
- single writer / multiple readers (S/M),
- multiple writer / single reader (M/S), and
- multiple writers / multiple readers (M/M).

Access profiles are interpreted with respect to the number of involved keys rather than users: A single user (S/S) scheme provides a single key for the generation of search tokens, even if the latter is distributed among several clients and/or users (with all the associated consequences for security). A multi-reader scheme provides several keys for generating search tokens, one for each user, supplemented by efficient mechanism for key revokation. Multi-writer schemes are typically PEKS schemes, distributing ownership among several users which are allowed to modify the document base. Searchitect concentrates on symmetric schemes, covering the S/S and S/M scenarios, although not necessarily restricted to those. However, and as mentioned in Section 1, Searchitect's approach is hybrid. It separates the encrypted search structure (i.e. the index) from encryption and access mangement of the plaintext database, which is done by the application utilizing our framework in its own favored way.

3.1 Common SE Functions

Although some SE schemes such as (Song et al., 2000), (Shen et al., 2009) or (Boneh et al., 2004) encode searchable information into the ciphertext itself, most of the schemes are *index-based*. Based on a set of chosen keywords $\mathcal{W} = \{w_1, \dots, w_n\}$, the document base $\mathcal{D}$ is indexed, either in a forward or backward manner. Using the owner's, i.e. writer's, key, this information is mapped to an 'encrypted' index $\mathcal{I}$, hiding information on the keywords in a still searchable way. Generally, every index-based SE scheme provides at least the following functionality.

- For initialisation, or setup:
  - $\mathsf{Keygen}(1^\lambda)$: A probabilistic algorithm that, given a security parameter λ, generates the key material needed by a user, in symmetric S/S and S/M schemes a single key k.
  - $\mathsf{BuildIndex}(k, \mathcal{D}, \mathcal{W})$: builds the encrypted index $\mathcal{I}$ over the entire document base $\mathcal{D}$ with respect to the keywords in $\mathcal{W}$.
- After setup, during operational phase:
  - $\mathsf{Trapdoor}(k, w)$: Creates an (encrypted) search query q for a keyword w. This is done locally on the user's client, using her key k, and subsequently forwarded to the server.
  - $\mathsf{Search}(q, \mathcal{I})$: The search algorithm for an encrypted trapdoor q over the (encrypted) Index $\mathcal{I}$. This procedure returns encrypted representations of the matching document identifiers.

Some SE schemes are *static*, i.e. they do not allow to modify the encrypted index $\mathcal{I}$ in an efficient manner. Such schemes force the owner of the document base to rebuild the encrypted index from scratch after every change and are therefore impractical for most real-world applications. *Dynamic* SE schemes, i.e. those which allow for an efficient modification of $\mathcal{I}$ after setup, also provide

  - $\mathsf{UpdateIndex}(k, \mathsf{op}, id_D, \{w_i\})$: The update function with respect to a certain scheme-specific set of operations. For example, $\mathsf{op} = \mathsf{add}/\mathsf{del}$ adds/removes a document D with identifier id_D and keywords $\{w_i\}$ to/from the encrypted index.

3.2 Security of SE Schemes

Informally speaking, an SE scheme is secure if it reveals no sensitive information to the untrusted server, keeping plaintext content of the encrypted documents and search queries private. However, trading off efficiency against security almost all practical schemes leak information to the server: Either the number of documents matching a query (i.e. the *size pattern*) or other partial information on the *access pattern* (i.e. the ids of the matching documents) which might be used to infer the user's *search pattern*. Hiding access and search pattern is a subtle issue. Although there exist several formal notions to reflect trapdoor, index and update security, many of them do not capture all threats that result from the minimal information leakage on the above mentioned patterns. Hence the majority of schemes backed by security proofs are nevertheless vulnerable to file injection attacks (Zhang et al., 2016), or statistical inference by counting attacks (Naveed et al., 2015),(Cash et al., 2015), or such attacks which exploit prior knowledge on the user's search habit (Liu et al., 2014).

Dynamic schemes are particularly vulnerable to file injection if they are not forward and backward secure. *Forward security* means that previously sent search token cannot be applied to newly added data, and *backward security* prohibits the application of newly generated search queries to former states of the document collection. At the time of writing, (Chang and Mitzenmacher, 2005), (Bost, 2016), (Bost et al., 2016) and (Stefanov et al., 2014) are the only published forward secure schemes, and only (Bost et al., 2017) supports both forward and backward security.

4 SEARCHITECT FRAMEWORK

Searchitect is a complementary infrastructure to provide search functionality to existing encrypted web-applications. Operating on indexes only, Searchitect separates the search functionality entirely from the

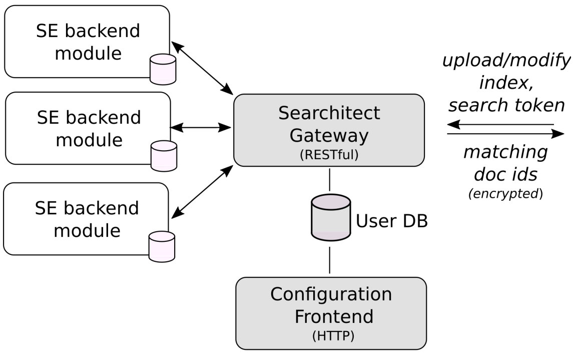
SE backend module
SE backend module
SE backend module
Searchitect Gateway (RESTful)
upload/modify index, search token
matching doc ids (encrypted)
User DB
Configuration Frontend (HTTP)

Figure 3: The Searchitect server comprises a RESTful gateway, which forwards authenticated requests to its backend searchable encryption modules.

document storage itself, and serves as an abstraction layer for the index part of arbitrary SE schemes (cf. Figure 1). Searchitect consists of two main components:

- The *Searchitect server*, offering a RESTful webservice which handles the search on the outsourced index, and
- the *Searchitect client*, which is integrated into the client software on the user's device, and which provides the interface for search queries, index setup and modification.

4.1 The Searchitect Server

Searchitect's webservice, cf. Figure 3, hosts several customized *Searchitect instances*, i.e. SE backend modules comprised of the scheme specific algorithms and the encrypted indexes. Besides its web-based configuration frontend it provides a RESTful gateway for the client-server interaction, which handles client authentication and forwarding of the client requests to their respective backend. Configuration and management of a Searchitect instance is done by the owner via the web frontend: An owner might choose to apply several copies of the same SE backend module to experiment with load balancing, or select different SE schemes to compare their performance. Besides a certain set of pre-defined SE schemes, as described in Section 4.4, Searchitect allows to use user-specific SE modules, and provides a simple access management (user addition and revocation) both for single and multi-user schemes. The Searchitect server offers logging of server/client activities at different levels of granularity to facilitate administration and evaluation:

- Basic: Reports access from clients, including the type of operation (search / update) and size of the encrypted index over time.

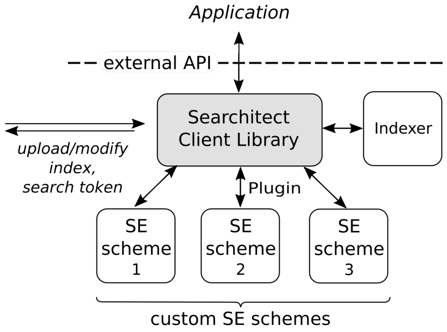

Figure 4: The Searchitect client serves as an abstraction layer to the custom cryptographic SE schemes.

- Advanced: Extends the basic log by timing measurements of server side update and search request, including the communication cost.
- Complete: Supplements the server side measurements by timing measurements collected from the clients. This mode allows to identify equivalent requests over separate SE instances to compare different SE schemes, if desired.

Besides that, the Searchitect server may also be used for key distribution of various multiuser SSE schemes, see Section 4.3.

4.2 The Searchitect Client

The Searchitect client, as shown in Figure 4, is a thin abstraction layer between the application and cryptographic SE schemes. It provides a simple generic API, which covers basic setup, search and edit commands (Table 1), and handles the communication between the application, its SE plugins, and the Searchitect webservice. While the client undertakes all non-specific tasks such as user authentication, logging, indexing plaintext documents and caching changes, the SE plugins are responsible for the actual cryptographic operations. They implement the scheme-specific algorithms such as BuildIndex, UpdateIndex and Trapdoor, and translate the generic SE commands into appropriate procedures.

The Searchitect client holds a password-locked master key, which is encrypted in a semantically secure way to prevent offline dictionary attacks. The master key is used to unlock the actual key material for the SE scheme(s) and to authenticate at the server. When performing a search, the communication flow is as follows: The application hands over a search query to the Searchitect client, which utilizes its chosen backend SE-plugin to transform the query via Trapdoor into a search token. If already authenticated, the Client sends the search token along with its

Table 1: API commands of the Searchitect client (draft).

`get/setURL()`	retrieves or sets the URL of the Searchitect webservice.
`setPwd()`	sets the user's client password, which is used to secure locally stored key material.
`setLog()`	controls the client's logging functions, including distribution of collected log data to the server.
`add/rmDoc(), updateDoc()`	adds, removes or updates the index locally, by handing over one or more plaintext documents.
`pushChanges()`	uploads the locally gathered changes to the Searchitect server.
`searchIndex(query)`	The actual search function. Creates the keyed search token, sends it to the server and extracts the document identifiers from its response.

(previously received) access token to the Searchitect webservice. The server processes the request (again, using the selected SE backend) and returns the reply to the client. The client uses its key material to extract the plaintext list of matching document ids from the encrypted search reply, handing it over to the application. Subquent steps are outside the scope of the Searchitect framework. E.g., the application might select some, or all of the document ids to receive either metadata or the full documents from the cloud storage and decrypt them with a key that is only known to the application.

4.3 Security Considerations

To prevent passive eavesdropping and active attacks on the client-server channel, communication between the client and the server needs to be secured. We propose to use Transport Layer Security (TLS) with server certificate to establish a secure and authenticated channel. Subsequent client authentication is done inside that secured channel, using a zero-knowlege protocol such as HMQV (Krawczyk, 2005) and the user's master key as client secret. A PKI-free alternative is to perform a TLS-SRP handshake, (Wu, 1998), using the user's password, whitened by the master key, as SRP's client credential. Whitening is essential to prevent a curious server from dictionary attacks on the SRP client verifier.

Symmetric multi-reader schemes require key distribution. As any other configuration of the Searchitect instance, this can be done via Searchitect's web frontend: for instance, the owner's browser receives the user's HMQV public key (or SRP verifier) from the Searchitect server, and uses this public key to (EC)-ElGamal-encrypt the symmetric key to be delivered. This encrypted key is then returned to the Searchitect server, which hands it over to the client when logged in.

One may expand any single user SE scheme to a multi-user setting, distributing the same trapdoor key among all users. Once a user is revoked by the owner of the Searchitect instance, the server blocks further access from that user. As all other client-server communication remains secure, a revoked user cannot learn anything about the requests of the others. However, such a construction should not be confused with a real single-writer/multi-reader (S/M) scheme: it does not prevent a malicious server colluding with a user, revoked or not, henceforth decrypting the search token of all others by a dictionary attack.

4.4 Implementation

As can be seen in Figures 3 and 4, the implementation will consist of at least 4 separate parts:

- The gateway service, which handles client requests and proxies the search requests to its backends.
- The server backends, implementing one SE scheme each. They are responsible for actually storing the search index and acting upon it.
- The client library, acting as a facade for the application developers exposing the functionality described in Table 1.
- The client backends, which are called by the client library to create the actual trapdoors for the server. The SE implementations have to match those of the server backends.

Our first server and client prototypes will be in Java, backed by the Clusion Library's implementation of a forward-secure variant of Dyn2Lev (Cash et al., 2014). Extensions to support the schemes from (Bost, 2017) are desirable and translations of the client code to other languages like C, C++ or Python are planned thereafter. It should be noted that the modular construction of Searchitect and the HTTP based communication will allow an interaction between implementations done in different programming languages on the server and the client.

To support an easy deployment and testing of the server part, the microservices architecture will be provided in a containerized solution using Docker[2]technology. The source code will be made available to the public under an OSI approved open source license.

5 SUMMARY AND OUTLOOK

The main point of Searchitect is to be easily integrated into software products, without requiring a deeper understanding of the chosen SE schemes by the application developer or complex modifications of already existing programs. Therefore Searchitect focuses on:

1. a comprehesible and easy to use API and documentation for developers,
2. an expandable framework to new schemes, and
3. an easily deployable and testable solution for private and public clouds.

Our first steps will be, as already discussed in more detail in Section 4.4, the development of the gateway service and a dummy SE scheme for testing purposes. Subsequently, we will implement the entire server and client architecture, as well as a fully functional SE scheme in order to provide a proof-of-concept for evaluation. Searchitect's approach is generic, and so is its scope of application. However, due to its hybrid design, the typical use cases we are considering are:

- cloud backup programs, allowing the restoration of a single file,
- cloud storage programs, allowing retrieval of a specific file on devices with a constrained network connection,
- mail user agents, allowing searching for a specific message while keeping the e-mails encrypted on the IMAP server,
- stacked filesystem encryption tools, also allowing retrieval of a specific file on devices with a constrained network connection.

We once more emphasize the experimental character of our framework. As searchable encryption is a very active field of research, Searchitect (as any other SE framework) should be used in prototype software projects only.

ACKNOWLEDGEMENTS

Part of our work on the Searchitect project is funded by *Netidee*[3] (*Internet Privatstiftung Austria*), grant no. 2099.

REFERENCES

Asghar, S. C. M. R., Galbraith, S. D., and Russello, G. (2017). Secure and practical searchable encryption: A position paper. In *Australasian Conference on Information Security and Privacy (ACISP 2017)*, LNCS 10342, pages 266 – 281. Springer Verlag.

Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G. (2004). Public key encryption with keyword search. In *EUROCRYPT 2004*, LNCS 3027, pages 506 – 522. Springer Verlag.

Bösch, C., Hartel, P., Jonker, W., and Peter, A. (2014). A survey of provably secure searchable encryption. In *ACM Computing Surveys, 47(2)*, pages 18:1–18:51. ACM.

Bost, R. (2016). $\Sigma o\phi o\varsigma$ – Forward secure searchable encryption. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1143 – 1154. ACM.

Bost, R. (2017). Open symmetric searchable encryption. https://github.com/OpenSSE.

Bost, R., Fouque, P.-A., and Pointcheval, D. (2016). Verifiable dynamic symmetric searchable encryption optimality and forward security. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1143 – 1154. ACM.

Bost, R., Minaud, B., and Ohrimenko, O. (2017). Forward and backward private searchable encryption from constrained cryptographic primitives. In *24rd ACM Conference on Computer and Communications Security (CCS'17)*, pages 1465 – 1482. ACM.

Cash, D., Grubbs, P., Perry, J., and Ristenpart, T. (2015). Leakage-abuse attacks against searchable encryption. In *22rd ACM Conference on Computer and Communications Security (CCS'15)*, pages 668 – 679. ACM.

Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Rou, M.-C., and Steiner, M. (2014). Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security (NDSS) Symposium 2014*, pages 1 – 16. Internet Society.

Chang, Y.-C. and Mitzenmacher, M. (2005). Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the Third International Conference on Applied Cryptography and Network Security (ACNS'05)*, pages 7442 – 455. Springer Verlag.

Cision PR Newswire (2017). Equifax announces cybersecurity incident involving consumer information. Sep. 07, 2017. https://www.prnewswire.com/news-releases/equifax-announces-cybersecurity-incident-involving-consumer-information-300515960.html.

[2] https://www.docker.com

[3] https://www.netidee.at

Encrypted Systems Lab (2017). Clusion v0.2.0. https://github.com/encryptedsystems/Clusion.

Ferreira, B. and Domingos, H. (2013). Searching private data in a cloud encrypted domain. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval (OAIR'13)*, pages 165 – 172. Centre de Hautes Etudes Interantionales D'Informatique Documentaire (CID).

Ferreira, B., Leitão, J., and Domingos, H. (2017). Multimodal indexable encryption for mobile cloud-based applications. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 213 – 224. IEEE.

Ferreira, B., Rodrigues, J., Leitão, J., and Domingos, H. (2015). Privacy-preserving content-based image retrieval in the cloud. In *IEEE 34th Symposium on Reliable Distributed Systems*, pages 11 – 20. IEEE.

Grubbs, P., McPherson, R., Naveed, M., Ristenpart, T., and Shmatikov, V. (2016). Breaking web applications built on top of encrypted data. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1353 – 1364. ACM.

Kamara, S. and Moataz, T. (2017). Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *EUROCRYPT 2017, Part III*, LNCS 10212, pages 94 – 124. IACR.

Krawczyk, H. (2005). Hmqv: A high-performance secure diffie-hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, LNCS 3621, pages 546 – 566. Springer.

Liu, C., Zhu, L., Wang, M., and Tan, Y.-A. (2014). Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176 – 188.

Meteor Development Group Inc. (2014). Meteor: The fastest way to build javascript apps. https://www.meteor.com.

Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference attacks on property-preserving encrypted databases. In *22rd ACM Conference on Computer and Communications Security (CCS'15)*, pages 644 – 655. ACM.

New York Times (2016). Yahoo says 1 billion user accounts were hacked. Dec. 14, 2016. https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html.

Popa, R., Redfield, C., Zeldovich, N., and Balakrishnan, H. (2011a). CryptDB: Protecting confidentiality with encrypted query processing. In *Twenty-Third ACM Symposium on Operating Systems Principles (SOSP'11)*. ACM.

Popa, R., Stark, E., Helfer, J., Valdez, S., Zeldovich, N., Kaashoek, M., and Balakrishnan, H. (2014a). Building web applications on top of encrypted data using Mylar. In *11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*, pages 157 – 172. USENIX Association.

Popa, R. A., Redfield, C., Tu, S., Balakrishnan, H., Kaashoek, F., Madden, S., Zeldovich, N., and Burrow, A. (2011b). CryptDB. https://css.csail.mit. edu/cryptdb.

Popa, R. A., Stark, E., Valdez, S., Helfer, J., Zeldovich, N., Kaashoek, F., and Balakrishnan, H. (2014b). Mylar research platform. https://css.csail.mit.edu/mylar/.

Shen, E., Shi, E., and Waters, B. (2009). Predicate privacy in encryption systems. In *Sixth IACR Thery of Cryptography Conference (TCC 2009)*, LNCS 5444, pages 457 – 473. Springer Verlag.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy (SP'00)*, pages 44 – 55. IEEE.

Stefanov, E., Papamanthou, C., and Shi, E. (2014). Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security (NDSS) Symposium 2014*. Internet Society.

The Guardian (2015). Massive anthem health insurance hack exposes millions of customers' details. Feb. 05, 2015. https://www.theguardian.com/us-news/2015/feb/05/millions-of-customers-health-insurance-details-stolen-in-anthem-hack-attack.

Wu, T. (1998). The secure remote password protocol. In *Proceedings of the 1998 Internet Society Symposium on Network and Distributed Systems Security*, pages 97 – 111.

Zhang, Y., Katz, J., and Papamanthou, C. (2016). All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX 16)*, pages 707 – 720. USENIX.