FernUniversität in Hagen

Seminar report

Client side encryption in the browser and key management

as part of the master seminar
applied cryptography

David Edler

Date:
October 2022 - February 2023

Advisor:
Osmanbey Uzunkol

Contents

1 Motivation

Digital processes are taking over everyday life, often described as Ubiquitous Computing [14]. Computation and storage used to be local on the personal computer. Advancements in bandwidth, network coverage and mobile computing opened the way for the trend to store data in datacenters or more broadly in the cloud. When this data includes personal information, the protection of the data becomes a core requirement. This is especially important for health-related [11] [12], financial [4], or privacy related data.

The key question for this report is how to ensure the remotely stored data stays private and is not readable for any other party. Encryption is common to keep data safe in transit and in storage. For transit, encryption with TLS has wide adoption by browsers [5] and providers. Privacy on the network itself as such is a solved problem. In storage, encrypted databases [3] or disk encryption [9] are available. Encrypting the storage protects from attacks such as a break in with stolen disks or from data exfiltration. But none of the mentioned approaches prevent the server or cloud operator from reading the data. Because the keys for TLS or for the storage encryption are available on the server. An attacker with physical or remote access to the server can extract the key and use it to decrypt and gain access to the cleartext.

By using client-side encryption, the user can maintain control over the keys to their data and ensure its sovereignty. Popular messengers like WhatsApp [7] and iMessage [1] started to adopt end-to-end encryption recently and bring the technology to a large audience. The user has a local key, and encrypts data on the local device directly after entering. Only encrypted data is sent over the network and stored as such in the datacenter. The cloud provider has no way to access the cleartext, and the goal to keep private information private is achieved.

A paper by Johns and Dirkensen [8] explores this solution for browser based web or cloud applications.

1.1 Cloud applications

Cloud applications, also known as cloud apps or web apps, are becoming increasingly popular. These types of applications can be divided into two distinct parts:

1. Frontend running in the browser of the user. The user interface (UI) serves as the primary point of interaction between the user and the system. The most used languages for coding the UI are HTML and

JavaScript. It is mostly stateless and communicates for storage and computation to a backend service.

2. Backend service running on third party computers in the cloud. Data is sent from the frontend via APIs to a remote location, where data is processed and stored in a datacenter. Commonly used are public cloud services like AWS, Azure, or Google compute cloud. There are different ways to implement cloud storage, such as a file storage solution like Amazon S3, or a more structured approach using a database like Amazon RDS.

The requirement is to keep the data private from anyone but the user. As mentioned, it is easy to encrypt the data for the API calls on the network via TLS/SSL. The backend of a cloud application stores the data in the realm of a cloud provider. Encrypting the storage on disk or using an encrypted database is also easily available. But the requirement cannot be satisfied with any of the above. Because the encryption key is available on disk or memory of the server. See figure 1 for a visualization.

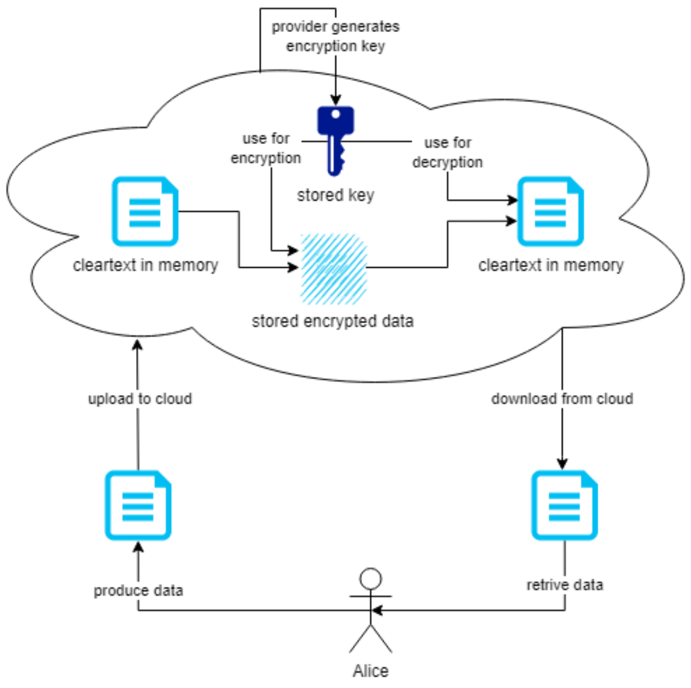

Figure 1: Storing data in and retrieving data from the cloud without end-to-end encryption. The cloud provider generates and stores the key.

There is only one way to protect data from being readable by the cloud provider. Encrypting it locally, before it leaves the user device, and store the key locally. In end-to-end encryption, the key never leaves the user realm and satisfies the requirement.

1.2 End-to-end encryption

In this scenario, the user stores the encryption key locally on a user-controlled device, as in figure 2. Only encrypted data is sent to the cloud and stored accordingly. The cloud provider cannot see the clear text or the key at any point in time.

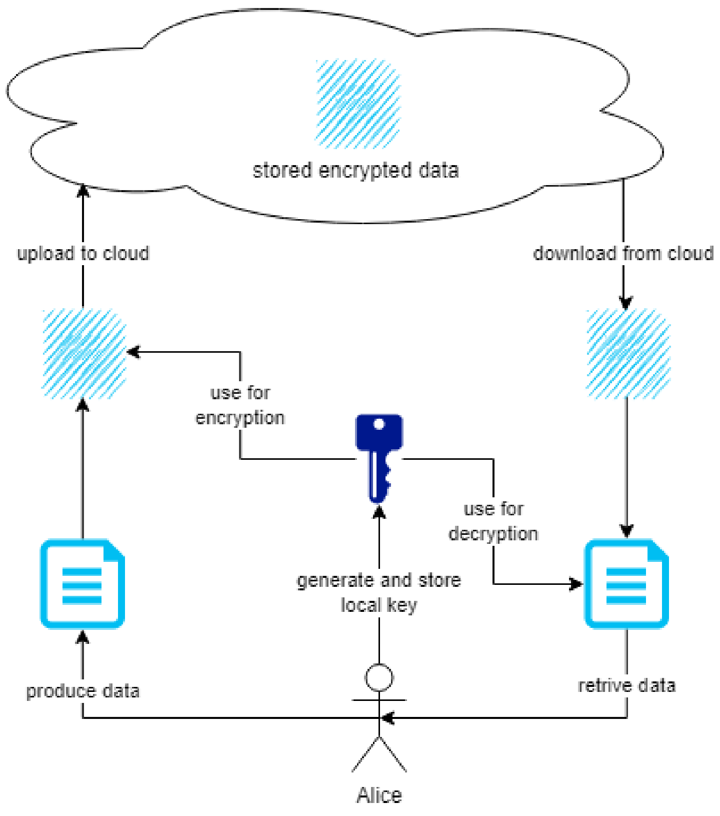

Figure 2: Using end-to-end encryption to store in and retrieve data from the cloud. The key stays with the user, and the cloud provider has no access to the stored information.

A proposed solution in the form of crypto membranes realizes end-to-end encryption for web applications. It will be explained in the next section 2.1. The challenge and a solution for operations on encrypted data will be shown in section 2.2. Problems with crypto membranes are listed in the section 2.3. The following section 2.4 will discuss more details on implementing the idea.

One core problem when dealing with end-to-end encryption is key management. Section 3 will explore solutions and architectures.

2 Proposed solution

2.1 Crypto membranes

Crypto membranes have been proposed by Johns and Dirkensen [8]. Browsers do not support end-to-end encryption out of the box today. As motivated earlier, it is very useful for cloud applications to have an easy way to implement end-to-end encryption. Crypto membranes are a proposal to bridge this gap.

Focus lies on the cloud application provider, the cloud provider, and the user. The user interacts with a cloud application and does not trust any of the other two parties. Because of this, the user also does not trust the web application code. The application runs in the browser and has fields for entering and displaying data. The crypto membrane encrypts entered information right after input. It stores the key for the encryption process locally, and the process is happening in isolation from the web application. To display data, the local key is used for decryption. Again, this happens in isolation from the application. The application and cloud provider never see the clear text or the encryption keys. See figure 3 for an overview of the system architecture. The green parts see ciphertext only and have no access to the key.

Johns and Dirkensen propose to introduce a new set of HTML Tags prefixed with crypto. A cryptoDiv, cryptoSpan, and cryptoH1 headings display encrypted information. Figure 4 illustrates usage of a CryptoDiv in web application code, which will be decrypted as described below.

New form elements called cryptoInput can receive data input. The figure 5 illustrates the usage.

The new crypto tags automatically encrypt and decrypt in an isolated sandbox. The process is mostly transparent for the user. This is where the term membrane comes from, the crypto tags are a transparent and isolated layer for secure data input and output. Browsers already support sandboxing via an iframe with a restrictive origin policy and an empty content security policy. The authors recommend implementation as a browser extension. The extension replaces the new crypto tags with an iframe on rendering. Encrypted values are passed between the web application and the iframe. In the iframe, a standard div, span, h1 or input tag is rendered.

The value of the input and the content of the display tags in the iframe

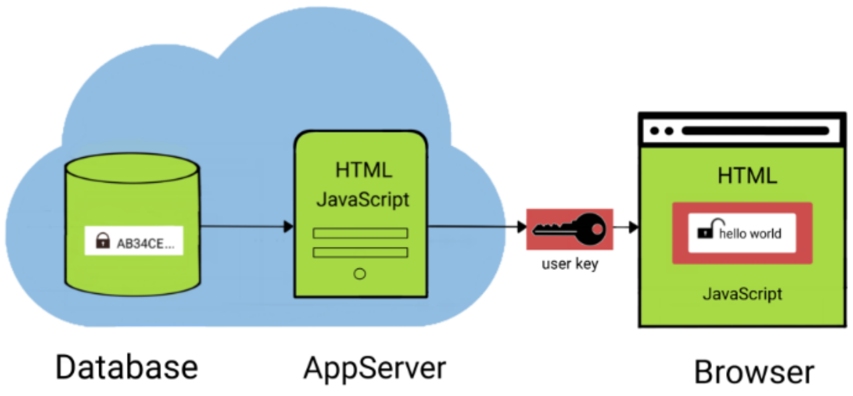

Figure 3: Crypto membranes system architecture as described by Johns and Dirkensen [8]. Clear text and encryption key in red are stored locally in an isolated environment. Areas not under user control are the web application in the browser, the application server and database in the cloud. Those parts have no access to the key or cleartext.

```
<CryptoDIV CMKeyID="123" CMAlgID="OrderPreserving">
AB34CEA23...
</CryptoDIV>
```

Figure 4: Example of a CryptoDiv to output encrypted data by Johns and Dirkensen [8]

is clear text. The encryption keys are available to the extension. The values passed to and from the web application one layer above the iframe are encrypted or decrypted on the fly, when passing the membrane in the form of the browser extension. The architecture realizes an isolated environment for entering data, encrypting the inputs and for decrypting and displaying data. Due to the isolation, only the extension needs to be trusted, but not the web application. Its code can be harmful or be manipulated. See figure 6 for a schematic view of the crypto membrane in the context of the application code executed in the browser.

The crypto tags separate the encryption/decryption from the web application. When new data is entered into a cryptoInput, the value is encrypted with a local key and the web application receives the encrypted value. This value can be sent via API to a backend and stored or retrieved later. To display encrypted data, a cryptoDiv or alike tags are used. The cloud provider also only ever sees encrypted data and does not need to be trusted.

```
<CryptoINPUT Type="text" Name="confinput" CMKeyID="345"
↪  CMAlgID="Deterministic">
```

Figure 5: Example of a CryptoInput to enter data that will be encrypted by Johns and Dirkensen [8]

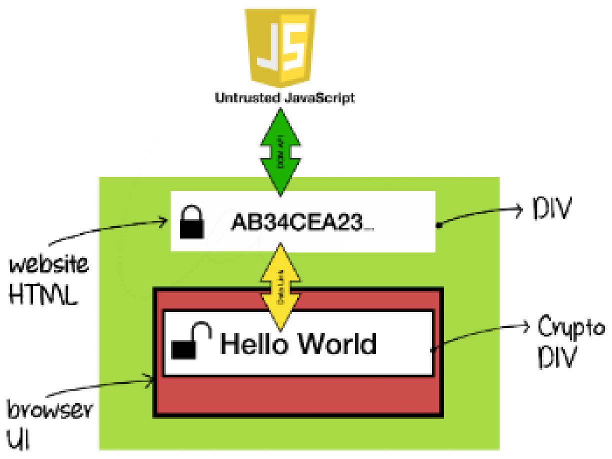

Figure 6: Crypto membrane in the browser by Johns and Dirkensen [8]. The web application contains JavaScript, which is untrusted. The untrusted JavaScript can only see encrypted values.

2.2 Fully homomorphic encryption

What is a cloud storage good for if it only holds encrypted values? If all the data must be downloaded and decrypted with the local key for every query, the system is not very useful. A solution is described as fully homomorphic encryption (FHE) using ideal lattices by Gentry [6]. The idea of FHE is that computations on the ciphertext produce an encrypted result. After decrypting, the result is equivalent to the result of the same computation on clear text. Hence, a computation can happen in the ciphertext space without the need for the computation environment to have access to the key or clear text.

In an end-to-end encrypted web application, within an FHE environment, a user can encrypt a piece of data, use it in a query and get an encrypted result. Data stored in the cloud is encrypted, and only the user has the key. With FHE the web application can perform operations in the cloud on the encrypted, remotely stored data, while the key for the data does not leave the user realm. With this approach, most common database operations in applications like selection, querying and sorting are possible on the homomorphically end-to-end encrypted values.

2.3 Problems with crypto membranes

Styling can be allowed by forwarding the styling of the crypto element to the element in the iframe. While the CSS can contain external URLs and as such allow information leaks, the empty content security policy of the iframe blocks any network calls and mitigates this type of attack.

There is a gap in styling complex inputs when rich text is entered into a cryptoTextarea. When this is later rendered with a cryptoDiv, the web application might want to style parts of the rich text in a specific way. There is no way described in the original paper in which styles for nested rich text elements can be forwarded to the iframe. Because it limits styling to the cryptoDiv itself. This problem can easily be avoided by specifying the input format for the cryptoInput tags to simple and plain text.

Users can be tricked to enter data into an unencrypted, standard HTML input field. The application can pretend to use the crypto elements, by faking their look and feel. A confirmation in a part of the browser window, that the web application cannot control, serves as a region of trust. The authors Johns and Dirkensen propose an indicator in or next to the URL bar, where the lock for https websites is located. The indicator lights up, when a crypto Input is active. A user can be sure to enter confident information in a crypto Input, by checking that this indicator is positive. While this approach works, it relies on an educated user, that does not fail to verify the indicator. For a new, not yet established standard, it is very likely users will miss the indicator or misunderstand its relevance.

None of the current interfaces for browser extensions allow manipulating the area close to the lock in the address bar. They do allow displaying an icon on the right of the address bar. The extension can place an indicator there. But those icon areas can be non-visible or hidden by the user. To implement the original advice by the paper, browser manufacturers must open the API to allow extending the lock area. But this creates an attack surface for confusing users with malicious browser extensions, which pretend a lock icon or a valid certificate where none is present. It is very unlikely browser manufacturers will give control of this area to extensions.

One more problem with crypto membranes is accessibility. While users being able to see can recognize the isolated element in the iframe easily due to its transparency, blind users will use screen readers. For screen readers, the sandboxed element in the iframe will not be treated equal to a native element on the original page. As such, it will be hard for blind users to hear the unencrypted value while still being in the flow of the rest of the page.

Another problem is the verification of the browser extension. When a user is tricked to install the wrong extension from the browser extension store, the whole encryption process is broken. A malicious extension could directly leak the unencrypted data to third parties via HTTP API calls.

2.4 Implementation of crypto membranes

The core of the extension can be slim. Because it relies on the sandboxing technique of iframes, which is already stable in all browsers, and battle tested.

The replacement of the crypto elements with the iframe is possible with standard JavaScript operations on the DOM of the rendered web application page. Johns and Dirkensen propose to find all crypto elements and iterate over them in the extension. For each found crypto element, an iframe is injected into the DOM and replaces the original element.

The authors propose an optional attribute on the crypto tags called algorithm. It gives some control over the encryption process to the web application. This is potentially dangerous, because the application can dictate a flawed algorithm or one with known attacks. For this reason, the property can only select properties of the algorithm like order preserving or deterministic.

Another optional attribute of the crypto tags is the key ID. The user must map existing local keys to the crypto tags. There might be more than one local key available. The key ID tag can solve the ambiguity. Because only once, the mapping must be established between a local key and the key ID. Then all crypto tags with the same key ID value will use the same local key. Without the key ID attribute, each crypto tag would need to be mapped individually. Or the same key would be used for all tags, which can be unwanted for some use case.

Apart from the key mapping, the extension also must manage the key lifecycle. The next chapter will discuss it in detail.

3 Key management

The keys for the decryption and encryption process need to be generated, saved, mapped to the encrypted data, backed up and eventually shared with other devices or third parties. Safe and easily usable storage of the keys on the user device is a complex problem.

Loss of keys is another challenge. If an application stores keys on a single mobile device, device loss is equivalent to data loss. Storing a copy of the key on another device or on an external storage is one way of creating resilience. A simpler solution is to print the key in an exported format. This can be a QR code, which the user can import with a mobile phone easily.

But the approach does not scale when handling many keys. One way to solve this is to have a main key, which is backed up as a printout. The keys used for encryption of data are encrypted with the main key and stored on a cloud storage along with the encrypted data. Giving access to a new device is as simple as importing the main key to that device and giving access to the cloud storage.

Another use case is to give access to a third party. The next section will explore a solution for sharing data in an end-to-end encrypted cloud environment. The principle is discussed for files, but it applies to data segments with the same key id in the context of crypto membranes as well.

3.1 File encryption and sharing

In a file use case, the user generates a unique key for every file. The file key is encrypted with a local key and stored in the cloud next to the encrypted file. As a result, the cloud provider can see the encrypted file and the encrypted key, but cannot decrypt the key or the file. The properties of end-to-end encryption are preserved. Figure 7 illustrates the process.

To share a file, Alice fetches the file key from the cloud, decrypts it with the local key and encrypts it with Bobs public key. Alice stores the newly encrypted key in the cloud. See Figure 8 for a schema of sharing the encryption key.

Alice can give Bob access to the cloud storage holding the encrypted data without restriction. Because all data and keys are encrypted. Using a different encryption key for each file allows sharing specific files. Bob can download the encrypted file and the newly encrypted key. Decrypt the file key with the private key and finally decrypt the file and read its clear text.

Public-key cryptography allows encrypting for the third party, while all parties keep their private keys secret. The public key of the third party is needed. GPG solves this with the web of trust, but this approach has limitations. There is a high entrance and acceptance barrier for new users. A public key infrastructure with trusted certificate authorities is a viable alternative.

An advantage of this process is that files need to be encrypted only once, even when sharing them with one or more parties. Assuming the shared files are larger than the keys, the new encryption of the file key for each recipient

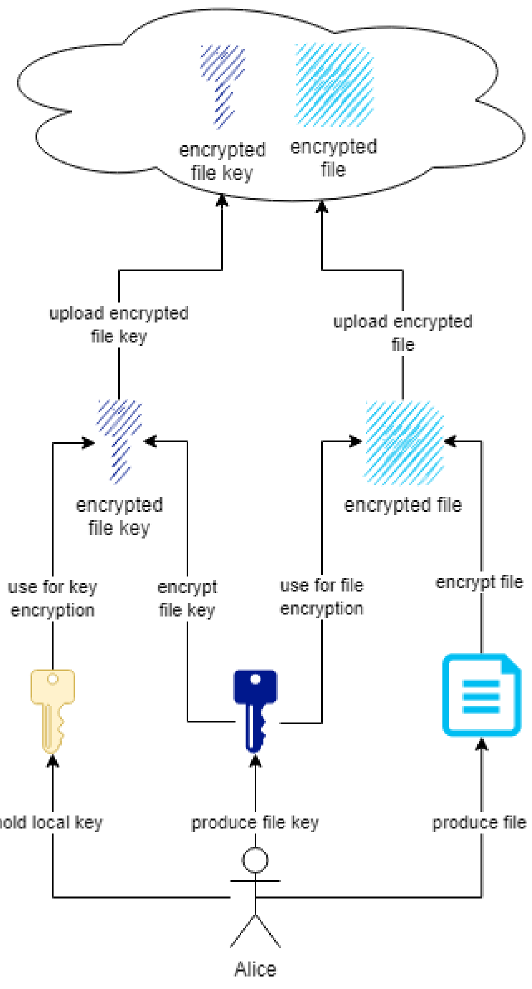

Figure 7: Storing an encrypted file and its encrypted file key in the cloud

is saving bandwidth and processing time, because sharing an encryption key for a new party is much cheaper than processing the file. It also saves storage space, because the encrypted file is the same for all recipients and needs to be stored only once.

4 Summary and outlook

We have seen that it is possible to extend browsers to support end-to-end encryption for cloud applications with crypto membranes. Furthermore, it is still possible to share or operate on the encrypted data stored remotely.

In section 2.3 several architectural and implementation problems of the

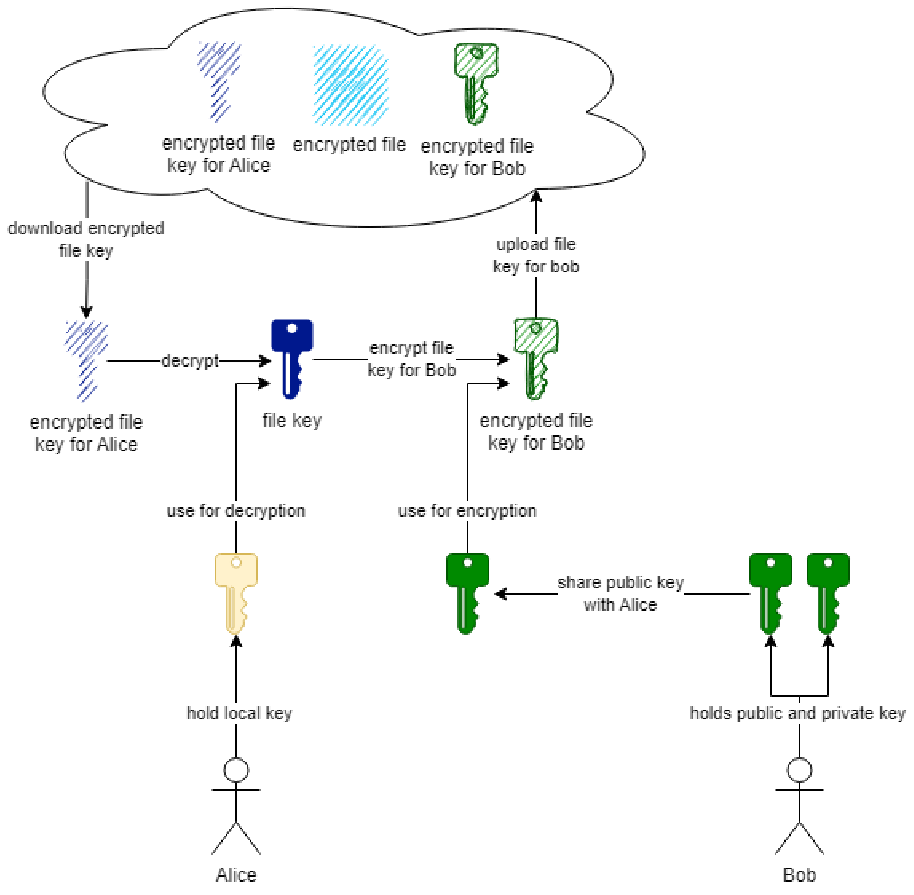
encrypted file key for Alice
encrypted file
encrypted file key for Bob
download encrypted file key
upload file key for bob
decrypt
encrypt file key for Bob
encrypted file key for Alice
file key
encrypted file key for Bob
use for decryption
use for encryption
share public key with Alice
hold local key
holds public and private key
Alice
Bob

Figure 8: To share an encrypted file, Alice fetches the encrypted file key, decrypts it with the local key, encrypts it with Bobs public key and uploads the key. The encrypted file is not involved in the sharing process.

approach have been discussed. None of the problems are blockers, they can be solved in cooperation with browser manufacturers. What is missing is a complete specification and a reference implementation of the idea.

Users should not trust application providers or cloud operators with sensitive data. Data on remote servers is likely to get stolen. An apple report [10] mentions that “Globally, the total number of data breaches more than tripled between 2013 and 2021“. There are no indicators, this trend is slowing down. End-to-end encryption gives some control to the user, so even if stolen, an attacker cannot access the clear text.

Recent attacks on LastPass [13] show that even end-to-end encryption is not a sole solution. The choice of encryption algorithms, encryption parameters, as well as the key and password length, are important to maintain security.

The biggest challenge is to make it easy for the end user to do the right choices. It is vital, that users have a correct mental modal [2] of the encryption architecture. The technology and implementation is not the biggest challenge, but to build tools that foster the understanding and as such the correct adoption of end-to-end encryption by the end user.

Enabling security membranes in browsers are a good approach to allow cloud application providers to build such a solution.

References

[1] Apple: *imessage*, 2023. https://support.apple.com/messages, (accessed: 11.02.2023).

[2] Wei Bai, Michael Pearson, Patrick Gage Kelley, and Michelle L. Mazurek: *Improving non-experts' understanding of end-to-end encryption: An exploratory study.* In *2020 IEEE European Symposium on Security and Privacy Workshops*, pages 210–219, 2020.

[3] Iqra Basharat, Farooque Azam, and Abdul Wahab Muzaffar: *Article: Database security and encryption: A survey study.* International Journal of Computer Applications, 47(12):28–34, June 2012. Full text available.

[4] S. Britto, R. Kumar, and S. Albert Rabara: *An end-to-end secure mobile payment system using public key infrastructure system.* Journal of Algorithms & Computational Technology, 6(3):395–409, 2012. https://doi.org/10.1260/1748-3018.6.3.395.

[5] Alexis Deveria: *Tls 1.2 support by all browsers*, 2023. https://caniuse.com/tls1-2, (accessed: 11.02.2023).

[6] Craig Gentry: *Fully homomorphic encryption using ideal lattices.* In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery, ISBN 9781605585062. https://doi.org/10.1145/1536414.1536440.

[7] F. inc: *Whatsapp*, 2023. https://www.whatsapp.com/, (accessed: 11.02.2023).

[8] Martin Johns and Alexandra Dirksen: *Towards enabling secure web-based cloud services using client-side encryption.* In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'20, page 67–76, New York, NY, USA, 2020. Association for Computing Machinery, ISBN 9781450380843. https://doi.org/10.1145/3411495.3421364.

[9] Louiza Khati, Nicky Mouha, and Damien Vergnaud: *Full disk encryption: Bridging theory and practice.* Cryptology ePrint Archive, Paper 2016/1114, 2016. https://eprint.iacr.org/2016/1114.

[10] Madnick: *The rising threat to consumer data in the cloud*, 2022. https://www.apple.com/newsroom/pdfs/

`The-Rising-Threat-to-Consumer-Data-in-the-Cloud.pdf`, (accessed: 11.02.2023).

[11] Gerald Ollerer and Alexander Mense: *End-to-End encryption for personal telehealth systems.* Stud Health Technol Inform, 200:140–145, 2014.

[12] G.H. Schroer: *Feasibility of end-to-end encryption using attribute based encryption in health care*, November 2016. `http://essay.utwente.nl/71375/`.

[13] Karim Toubba: *Notice of recent security incident*, 2022. `https://blog.lastpass.com/2022/12/notice-of-recent-security-incident/`, (accessed: 11.02.2023).

[14] Mark Weiser: *The computer for the 21st century.* SIGMOBILE Mob. Comput. Commun. Rev., 3(3):3–11, jul 1999, ISSN 1559-1662. `https://doi.org/10.1145/329124.329126`.

Declaration of Authorship

I declare that I have written the seminar paper independently and without unauthorized use of third parties. I have used only the sources and aids indicated, and have marked as such the passages taken verbatim or in spirit from them. The assurance of independent work also applies to any drawings, sketches or graphical representations included. The paper has not previously been submitted in the same or a similar form to the same or any other examination authority, nor has it been published. By submitting the electronic version of the final paper, I acknowledge that it can be checked for contained plagiarism with the help of a plagiarism detection service and will be stored exclusively for examination purposes.

Furthermore, I grant the department the right to evaluate the paper for its own teaching and research activities and to publish it appropriately, giving credit to the author.

Berlin, 31.01.2023

David Edler