Security Overlay for Distributed Encrypted Containers

Florian Patzer[1], Andreas Jakoby[2], Thomas Kresken[1], Wilmuth Müller[1]
[1]Fraunhofer Institute of Optronics, System Technologies and Image Exploitation – IOSB
Karlsruhe, Germany
[2]Bauhaus Universität, Weimar, Germany
florian.patzer / thomas.kresken / wilmuth.mueller@iosb.fraunhofer.de
andreas.jakoby@uni-weimar.de

Abstract: *Storage services enable a high potential for time and location independent access to information particularly combined with smart mobile devices. In combination with corporate and local storage, those services can be a powerful extension to available storage in enterprise or governmental environments. In contrast, common secure storage strategies like encrypted partitions or disks are static and remotely inaccessible, but are comfortable to use in a local scenario. However, storing sensitive data on public servers is not an option due to the possibility that an unauthorized third party can access it. Generally security policies like corporate compliance prohibit those services explicitly. Thus, sensitive data has to be encrypted to allow its storage on public servers.*

The paper at hand describes a security overlay using a trusted environment to build a distributed virtual encrypted container that supports OTFE (on-the-fly-encryption). For this purpose, an easily extendable security overlay is introduced where each file or data set is encrypted independently. The overlay provides a hierarchical key structure, which hierarchically controls access to uploaded data and maps the data structure at the same time. Additionally, the directory structures and the meta-data are protected against unauthorized access. Therefore, the presented concept enables the creation of a deniable distributed file system that can enable an implementation to make strong security promises.

The trusted environment can be provided by a device called CyphWay®, which has been developed at the Fraunhofer IOSB and presented at ICCWS 2014. The device guaranties that cryptographic keys are only available within a Hardware Security Module. Thus, the whole key structure and the keys themselves are protected even against the user devices, which is important regarding potentially insecure mobile platforms.

Unlike several encrypted container solutions the presented system allows to distribute encrypted data over a huge number of divergent publically available storage services, like cloud storages. In addition, it is possible to combine those storages with private or corporate storage.

Keywords: *Security Architecture, Secure Cloud Storage, Mobile Security, Information Security Management, Secure Distributed Storage*

1. Introduction

Within the last years, storage became available anywhere at any time. The booming Storage-as-a-Service (*STaaS*) market, advanced possibilities to access corporate storage remotely and mobile devices as smartphones, tablets or laptops make this flexibility possible. But with the increasing amount of mobility and storage diversity the types and numbers of possible attacks on the stored data and the corresponding keys increases also, even when modern cyber suits are applied. Classical strategies like encrypted containers, partitions and disks need to be mapped on mobile and remote usage. Distributed storage is attractive to private as well as professional users. STaaS or *cloud storage* providers offer clients to remotely access their storage, which is often enabled through applications for different platforms like Android, Windows, Ubuntu and (platform independent) web browsers. In addition, most of the cloud providers offer different amounts of space in their prizing model and some advertise small storages free of charge. This makes them interesting for all kinds of users. However, because of security policies and corporate compliance, the usage of such services is not common in corporate and governmental environments. This is mainly because sensitive data can be easily read by the storage providers. Many applications and techniques addressing this problem do not consider the storage provider as an attacker and are, for that reason, no solutions for professional environments. As an example, US cloud providers are forced to release even sensitive data of their customers to the US government due to the USA Patriot Act.

However, users who are allowed to work with cloud, local and corporate storage have to use several clients to manage their data. There are very few solutions that allow to combine those storages,

especially in mobile operating systems. Systems that provide this desirable feature, do not meet security requirements for sensitive data and are therefore often useless in professional environments.

Lately, new possibilities of assembling cloud storage on the client side have been created ([1], [7], [3]). Those clients are not satisfying in terms of the provided security. Additionally, none of the available products and concepts provides a satisfying directory structure that contains distributed elements and can be used as one single directory structure, like local encrypted containers. In advance, some use cases might require client software, where users do not have to care about the distribution of their data over different storage locations.

On the server side assembling storage that is situated on different locations is currently done by distributed file systems like Andrew File-System [5] or Google File System [4]. Those have been developed to fulfill the needs of data centers and are mostly limited to the file system used by the respective center. A user accessing those systems is bound to their technology.

There are techniques available that partially provide more flexibility. Distributed file systems such as Tahoe [10] and its advancements like [9] allow the combination of diversely located storages with different underlying file systems. However, as every location needs to run the same virtual file system or server software, those techniques do not address the desired usage of STaaS, which requires the support of diverse interfaces. That is why they are only used in environments where a standardized file system or server software is deployed. In the domain of the global web business, this alignment would be against the desires of one provider to dissociate oneself from the competition, keeping their customers dependent. Additionally, a migration could imply tremendous costs.

There is also work available that concentrates on the security of distributed file systems [8]. Those methods depend on a trustworthy environment that is assumed as given when deploying the client software. The work that is been made on the field of security of distributed file systems does not fulfill the need of an appropriate level of security when mobile devices are used as clients. That is because the trustworthy environment that is used to perform cryptographic operations in the mentioned concepts is the mobile device itself. The security of those devices is widely contentious. Especially law enforcement and military usage of storage services needs a higher level of protection in a mobile environment. This can be achieved by using an external hardware device like the CyphWay® that provides such a trusted environment and is explained later in this document.

The paper at hand provides a technique to map classical encrypted containers to modern storage strategies, like the storage of critical data on public services. The document presents a security overlay that can be applied to create a distributed virtual encrypted container to achieve a high degree of security and good user experience when combining several unprotected storage locations. A storage location is defined as unprotected, if an unauthorized individual can access a dataset that is stored on that location or available anywhere within the network. We suppose that any remote storage and any channel is insecure. Additionally, we admit that user devices can be compromised as mobile devices tent to have many security issues. Therefore, the data to be stored needs to be protected carefully by encryption and the keys, applied to those encryption instances, need to be protected even against the user devices. The overlay is supported by an external trusted hardware device (comparable to a hybrid of TPM[1] and HSM[2]) to perform cryptographic operations. This device is the only location where keys ever appear in plaintext. The resulting environment is supplemented by a specially designed key management system. The stored data and its meta-data are being encrypted and controlled by access rights. In addition, the overlay achieves the protection of the data structure by hiding it completely from unauthorized individuals.

2. Overlay Structure

At first, the overlay structure of our concept is presented. This is the key to allow the desired high level of security and uses the necessary level of abstraction to build a distributed container. Therefore, it is shown how the data and also the meta-data get protected and hidden by this technique. This will be achieved by separating the meta-data from the actual data.

Data Structure: Let $G = (V, E)$ be a directed graph, where the vertices denote directories or files. Edges represent associations in the following way:

> If $(u, v) \in E$ then u represents a directory which includes a sub-directory or a file represented by v. In other words, u is parent of v.

Note that files are always represented by sinks and that in this document file-level granularity is used, but in general different granularity levels are conceivable. This might even be a recommendable parameter for implementations. In addition, a bijective function

[1] Trusted Platform Module
[2] Hardware Security Module

$i: V \cup E \to N$ is defined that maps the graph elements to a set of indices, in order to identify them.

Now, a simple notation to navigate through G is provided. This will later be mapped on a key-based navigation. For $v \in V$ let $\Gamma^-(v) = \{w|(v,w) \in E, w \in V\}$ and $\Gamma^+(v) = \{w|(w,v) \in E, w \in V\}$ denote the outgoing end entering edges of v. For every vertex $v \in V$ it is assumed that there exists a given order within $\Gamma^-(v)$ and within in $\Gamma^+(v)$. Let $\text{in}(v)$ denote the first edge in $\Gamma^+(v)$ and let $\text{out}(v)$ denote the first edge in $\Gamma^-(v)$. For edges $e_1 = (u, \quad v) \in E$ let $\text{in}(e_1)$ denote the direct successor of e_1 within the edges in $\Gamma^+(v)$ and let $\text{out}(e_1)$ denote the direct successor of e_1 within the edges in $\Gamma^-(u)$.

Meta-data Structure: The meta-data, mentioned before, includes information like file and/or directory names, additional access restrictions, location of storage, etc. In the following, meta-data will be extracted from the respective file or directory in order to build a meta-graph.

The partial functions $\sigma: N \to M_V$ and $\rho: N \to M_E$ represent mappers that link the real data to the related meta-data, where N denotes a universe of indices, M_V denotes the universe of the meta-data extracted from the vertices of G and M_E denotes the universe of meta-data extracted from the edges of G. Consequently, the resulting meta-graph is defined as $G_M = (M_V, M_E)$. Furthermore, in favor of an intuitive understanding $M_v \in M_V, M_e \in M_E$ are written as synonyms for $\sigma(v) \in M_V, \rho(e) \in M_E$.

The location of a real file is part of its meta-data. Therefore, the function $\tau: N \to S$ is needed which maps an index $n \in N$ onto a storage location $s \in S$ where S denotes the set of all storage locations that shall be included in the desired storage distribution. To map the storage location $\tau(n)$ onto the graph element and its meta-data we define $\varphi: S \to (V \text{x} M_V) \cup (E \text{x} M_E)$. As a result, we can use $\tau(n)$ to access the storage location where the graph element $\varphi(\tau(n))$ is stored for any $n \in N$.

Design: Because every stored structure and information within this overlay will be protected by using encryption, the following abstract encryption scheme (Enc, Dec) is introduced. It is intentional that the overlay does not rely on a particular encryption scheme and by abstracting from such schemes, their exchangeability is guaranteed. Let Enc denote the encryption and let Dec denote the decryption function. The encryption of the data d is denoted by $Enc(k_{Enc}, \quad d) \to c$ and the decryption is denoted by $Dec(k_{Dec}, \quad c) \to d$ where k_{Enc} and k_{Dec} are the respective encryption and decryption keys. In favor of simplicity we assume $\Pr[Dec(k_{Dec}, \quad Enc(k_{Enc}, \quad d)) \to d] = 1$ for all possible data instances d. The encryption strategy works as follows:

- M_v is encrypted using an (approximately) unique key pair (k_{Enc}^v, k_{Dec}^v), for every vertex $v \in V$.
- M_e is encrypted using an (approximately) unique key pair (k_{Enc}^e, k_{Dec}^e), for every $e \in E$. This is a simplified exposition of the key structure. It will be extended within the next paragraphs.
- The meta-data set of each vertex includes keys that are needed to encrypt and decrypt the meta-data of the incident edges. For example, let $e = (v, w) \in E$ then M_V contains (k_{Enc}^e, k_{Dec}^e).
- The meta-data set of each edge includes keys that are needed to decrypt the meta-data of its connected vertices.
- In addition, the meta-data set of any edge is encrypted by an access-right key which guarantees that only users with access rights to the end-vertices can gain any information about the corresponding vertex.

Consequently, the minimal content of each $M_v \in M_V$ and each $M_e \in M_E$ is set as follows: The meta-data set M_v of a vertex v contains

- The name of the represented directory or file.
- The values $\text{in}(v)$, $i(\text{in}(v))$, $\tau\left(i(\text{in}(v))\right)$ as well as $\text{out}(v)$, $i(\text{out}(v))$, $\tau\left(i(\text{out}(v))\right)$.
- A set of key pairs K^{in} used to encrypt the meta-data of the edges in $\Gamma^+(v)$ and a set of key pairs K^{out} used to encrypt the meta-data of the edges in $\Gamma^-(v)$. These entries are of the form (k_{Enc}^e, k_{Dec}^e) for the adjacent edges.

The meta-data set M_e of an edge $e = (u, \quad v)$ contains:

- The name of the directory or file represented by u and v.
- The values $i(\text{u})$, $\tau(i(\text{u}))$ as well as $i(v)$, $\tau(i(v))$.
- The values $\text{in}(e)$, $i(\text{in}(e))$, $\tau\left(i(\text{in}(e))\right)$ as well as $\text{out}(e)$, $i(\text{out}(e))$, $\tau\left(i(\text{out}(e))\right)$.
- A key pair $(k_{Enc}^{\text{head}}, \quad k_{Dec}^{head})$ used to encrypt the meta-data of u and the key pair $(k_{Enc}^{\text{tail}}, \quad k_{Dec}^{\text{tail}})$ used to encrypt the meta-data of v.

As mentioned, there is more to the encryption of M_e. An additional encryption of the meta-data edges is needed to handle access rights (see Section 3). Thus, every $M_e \in M_E$ will be encrypted using an additional key pair $\left(k_{Enc}^{U(v)}, \quad k_{Dec}^{U(v)}\right)$ where $e = (v,w) \in E, v,w \in V$, before encrypting it using (k_{Enc}^e, k_{Dec}^e). Therefore, the meta-data structure can be implemented as tuples as follows:

- $\langle i(\mathrm{v}), \quad Enc(k_{Enc}^v, \quad M_v)\rangle$ for every $v \in V$
- $\langle i(e), \quad Enc(k_{Enc}^e, \quad \mathrm{in}(e)), \quad Enc(k_{Enc}^e, \quad \mathrm{out}(e)), \left(k_{Enc}^{U(v')}, k_{Dec}^{U(v')}\right) E\left(k_{Enc}^{v'}, k_{Dec}^{v'}, M_e\right)\rangle$ for every $e = (v,w) \in E, w \in V$

At this point, the original data structure is fully mapped by the meta-data structure. As a result, the original data can now be diversified on arbitrary storages. This data has to be encrypted. Therefore, one may add additional keys and encryption schemes, but it is suggested to use the key pair (k_{Enc}^v, k_{Dec}^v) for a file or dataset represented by a vertex $v \in V$.

The presented overlay allows the implementation of an encrypted container that can be used on several devices simultaneously and not only stores the data in a cloud but facilitates the user to include multiple corporate and external STaaS entities. Furthermore, the overlay is designed to create comfortable client software which provides a container that can be utilized like a local directory.

Algorithmic Examples: The following algorithms are minimalistic in favor of simplicity and focus on the overlay manipulation. Some details like extractions through decryption are avoided as they are implicitly clear. The content of a directory associated with a given vertex v is computed as follows:

determineDirectoryContent($i(v)$, (k_{Enc}^v, k_{Dec}^v))
begin
  decrypt $Enc(k_{Enc}^v, \quad M_v)$ using k_{Dec}^v
  e = $(v,w) \leftarrow \mathrm{out}(v)$
  determine $k_{Dec}^e \leftarrow K^{out}$
  loop until the successor edge e is not defined
    determine $Enc(k_{Enc}^e, \quad M_v)$ by
    decrypting $E\left(k_{Enc}^e, \quad E\left(k_{Enc}^{U(w)}, \quad M_e\right)\right)$
    from
      $\langle i(e), \quad Enc(k_{Enc}^e, \quad \mathrm{in}(e)), \quad Enc(k_{Enc}^e, \quad \mathrm{out}(e)), \quad Enc\left(k_{Enc}^e, \quad E\left(k_{Enc}^{U(w)}, \quad M_e\right)\right)\rangle$
    if the user has access to $k_{Dec}^{U(w)}$ then
      decrypt $Enc\left(k_{Enc}^{U(w)}, \quad M_e\right)$
      add the necessary content
    information from M_e
      to the content of v
    end
    $e = (v,w') \leftarrow \mathrm{out}(e)$ by decrypting
  $Enc(k_{Enc}^e, \quad \mathrm{out}(e))$
  end
end

A new vertex v' can be added to $v \in V$ as follows:

appendFileOrDirectory($i(v)$, $i(v')$, (k_{Enc}^v, k_{Dec}^v))
begin
  determine M_v
  create $M_{v'}$ M_e where $e = (v,v') \in E$
  generate
  $\left(k_{Enc}^{U(v')}, k_{Dec}^{U(v')}\right)$ $\left(k_{Enc}^{v'}, k_{Dec}^{v'}\right)$ (k_{Enc}^e, k_{Dec}^e)
  add (k_{Enc}^v, k_{Dec}^v) as $(k_{Enc}^{head}, k_{Dec}^{head})$ and $\left(k_{Enc}^{v'}, k_{Dec}^{v'}\right)$ as $\left(k_{Enc}^{tail}, k_{Dec}^{tail}\right)$ to M_e
  add (k_{Enc}^e, k_{Dec}^e) to K^{out} of M_v and to K^{in} of $M_{v'}$
  determine $\mathrm{in}(e)$ and do $Enc(k_{Enc}^e, \quad \mathrm{in}(e))$
  determine $\mathrm{out}(e)$ and do $Enc(k_{Enc}^e, \quad \mathrm{out}(e))$
  do $Enc\left(k_{Enc}^e, \quad E\left(k_{Enc}^{U(v')}, \quad M_e\right)\right)$
end

Moving a vertex v' from parent v to parent w can be easily done by applying the following algorithm:

appendFileOrDirectory($i(v)$, $i(v')$, $i(w)$, (k_{Enc}^v, k_{Dec}^v), $\left(k_{Enc}^{U(v')}, k_{Dec}^{U(v')}\right)$, (k_{Enc}^w, k_{Dec}^w))
begin
  determine M_v, M_e, M_w where $e = (v,v') \in E$
  remove (k_{Enc}^v, k_{Dec}^v) (resp. $(k_{Enc}^{head}, k_{Dec}^{head})$) from M_e
  remove e from $\Gamma^-(v)$
  remove (k_{Enc}^e, k_{Dec}^e) from K^{out} of M_v
  add e to $\Gamma^-(w)$
  add (k_{Enc}^e, k_{Dec}^e) to K^{out} of M_w
  add (k_{Enc}^w, k_{Dec}^w) as $(k_{Enc}^{head}, k_{Dec}^{head})$ to M_e
end

To speed up the access on objects that are usually used at the same time, like the edges which represent the content of a directory, we can group these objects and store them within a meta-object at the same location.

If a user navigates through the visible directories he gets some knowledge on the hidden vertices of the directory tree, like the number of entries or the number of predecessors. To hide this kind of information we can add some dummy entries to our system. A dummy entry consists of a random edge "cipher", which appears within the lists $\Gamma^-(v)$ and $\Gamma^+(v)$ of a vertex v, but has no meaningful decrypted meta-data, i.e. the ciphertext will be a random string.

3. Handling Access Rights

In Section 2, the basic overlay structure and its encryption was presented. However, there is still no access handling regarding different users. To control the access rights of an individual user it has to be distinguished between two scenarios:

1. The initial access to a directory or file within the container, i.e. the first access of the user to an element within the file system.
2. The user has currently access to a directory. Therefore, her or his access rights regarding the directory content and the parent directories has to be controlled.

The control of the initial access within the first scenario follows the method to control the access on data records in a cloud as presented in [6]. An example can be found within Figure 1

For each user u a user key pair $(k^u_{Enc} \quad k^u_{Dec})$ is introduced. It is assumed that this key pair is stored in such a way that no unauthorized person has access to this pair (i.e. within the trusted environment that unlocks the key pair, if the user has authenticated himself via fingerprint). To manage the access rights it might be possible that some kind of a super-user within an organization must have access to these keys and is able to add new user keys (i.e. in the trusted environment).

If a user u has access to a vertex v within the file graph structure, the key pair $(k^v_{Enc}, \quad k^v_{Dec})$ and the corresponding storage location are stored as an entry point. This kind of information is called adaptor data A^u_v and is protected through encryption $Enc(k^u_{Enc}, \quad A^u_v)$. Thus, if a user u would like to access a directory or a file represented by a vertex v, he reads the corresponding encrypted adaptor data $Dec(k^u_{Dec}, Enc(k^u_{Enc}, \quad A^u_v)$. After extracting $(k^v_{Enc}, \quad k^v_{Dec})$ he or she can access the meta-data set M_v.

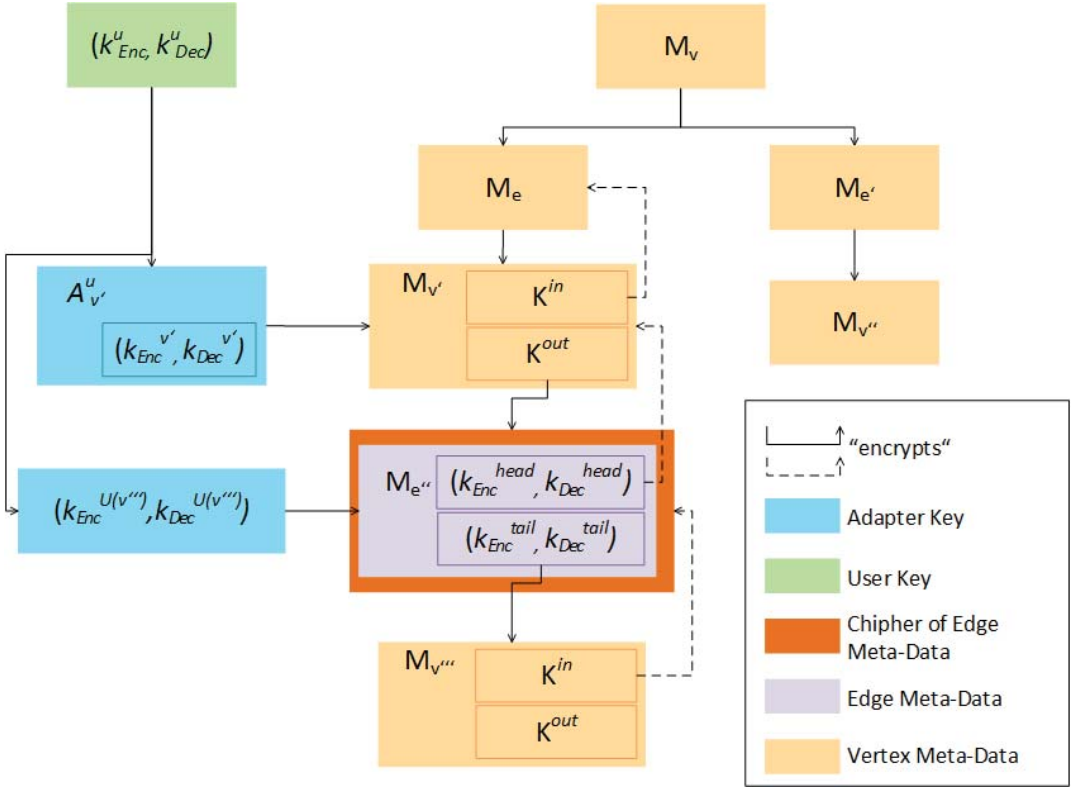

Figure 1: Illustration of an encrypted meta-data tree with an access example

To add another entry point for a user one can simply add the corresponding encrypted adaptor data for this user to the system. Analogously, if one would like to remove an entry point of a user one can remove the corresponding encrypted adaptor data from the system.

To control the access rights within the first scenario the key pairs $\left(k_E^{U(w)},\ k_D^{U(w)}\right)$ for $e = (v.w) \in E$ have been introduced.

It is possible to group the graph elements according to users who can access them. For any vertex v let $U(w)$ denote the set of users that have access to M_w. Then $\left(k_{Enc}^{U(w)},\ k_{Dec}^{U(w)}\right)$ represents the access key pair that is used to encrypt the meta-data of the edges with tail w. Analogue to the adaptor data these keys are stored encrypted by the user keys k_{Enc}^{u} for every user $u \in U(v)$.

4. Trusted Environment

As mentioned before, the trusted environment needed for cryptographic operations and access control within the presented overlay, can be provided by a hardware security device like the CyphWay®. The CyphWay® was developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation. The implemented demonstrator uses Android devices to visualize the data and is implemented on Raspberry Pi's using FPGAs for the necessary cryptographic operations and for the storage of administratively entered or cached keys. Overview of this demonstrator system is illustrated within Figure 2.

To access data the demonstrator tries to decrypt the corresponding encrypted meta-data. If the required key is not locally available for the CyphWay®, it tries to fetch the necessary key. Therefore it invokes a remote storage lookup to obtain a cipher of this key that can be decrypted by applying the current user key. If a user would like to access a directory where she or he is not authorized to access all subdirectories and included files, the CyphWay® filters the elements of the directory. It only forwards that information to the user which has been decrypted successfully. Consequently, elements the user is not authorized to see will be invisible for him or her.

The smartphone as well as any other final user device is only used to connect the CyphWay® to the storage and to manage the content of the directories and files. Plaintext keys are never available outside the the hardware security module of the CyphWay®. In addition, no cryptographic operations take place on the user device, which only signals encryption and decryption interests to the CyphWay®.

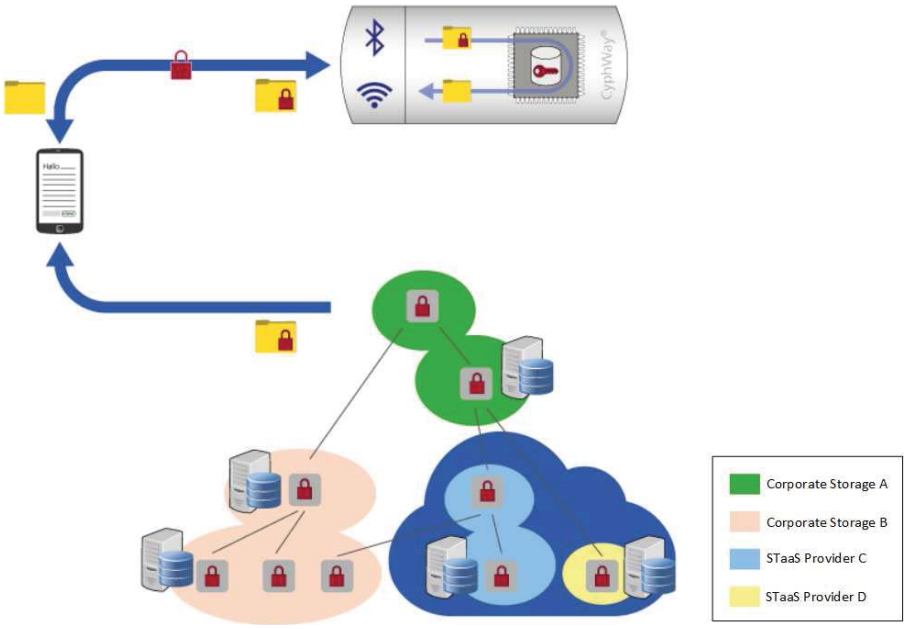

Figure 2: Illustration of the IOSB demonstrator and the distributed file system

Figure 2 illustrates the access to a specific file. The encrypted file is send from the storage to the smartphone and from the smartphone (e.g. via an encrypted Bluetooth channel) to the trusted hardware environment. There, the file is decrypted. In the next step the file is sent (e.g. via an encrypted Bluetooth channel) back to the smartphone, where it can be visualized and accessed.

Within the demonstrator the trusted environment is partitioned into a connector module and a core crypto module (the HSM). Thus, the used Bluetooth channel can be easily replaced against an arbitrary other secured communication channel, e.g. WLAN (which might lead to high energy consumption), or the crypto device can be connected directly to the smartphone or any other device via USB.

No keys will ever be available outside the trusted environment. As a result, even active attacks like Man-in-the-Middle attacks or information gathering Malware are not useful to extract any of the keys, if the underlying encryption scheme is sufficiently secure.

5. Conclusion

In this paper we presented a security overlay that allows the implementation of a distributed virtual encrypted container which supports OTFE. Because of the design of this concept, the overlay can be applied on a variety of underlying platforms, operating systems and file systems. The shown security overlay allows the combination of several storages, like STaaS, corporate datacenters and private clouds. Every data that gets stored within the virtual container gets encrypted, access controlled and is, therefore, protected from Dolev-Yao attackers [2] and even the storage owners. Additionally, the keys and meta-data are protected and access controlled. Utilizing the described demonstrator CyphWay® as a trusted environment makes it possible to protect the keys against every adversary, even against the owner's system. Therefore, we suggest the proposed security overlay for modern distributed storages that are accessed by mobile or other insecure clients. Since storing data on one encrypted partition is not a common use case anymore the presented technique can be used to meet the needs of modern storage strategies. In the future we will work on an implementation of a distributed virtual encrypted container using the overlay to demonstrate the potential of this concept.

References

[1] CloudFuze (2014) [online], https://www.cloudfuze.com/

[2] Dolev, D., Yao, Andrew C., "On the security of public key protocols", in: IEEE Transactions on Information Theory, Vol. 29, Issue 2, IEEE 1983, pp. 198-208

[3] Dongju, Y., Chuan, R. (2014) "VCSS: An Integration Framework for Open Cloud Storage Services", Proceeding of 2014 IEEE World Congress on Services (SERVICES), (pp. 155-160). Anchorage, AK.

[4] Ghemawat, S., Gobioff, H., Leung, S.-T. "The Google File System" Proceedings of the nineteenth ACM symposium on Operating systems principles SOSP '03 (29-43). New York, USA: ACM

[5] Howard, J. et al. (1988) "Scale and performance in a distributed file system". ACM Transactions on Computer Systems (TOCS), Volume 6 Issue 1, Feb. 1988, pp. 51-81.

[6] Jakoby, A., Müller, W., and Vagts, H. "Protecting Sensitive Law Enforcement Agencies Data - Data Security in the Cloud", Proc. of International Conf. on Cyber Warfare and Security (ICCWS 2014).

[7] Machado, G. S., Bocek, T., Ammann, M., Stiller, B. (2013) "A Cloud Storage Overlay to Aggregate Heterogeneous Cloud Services" Proceedings of the 38th Conference on Local Computer Networks (pp. 597 - 605). Sydney, NSW : IEEE.

[8] Pletka, R., Chachin, C. (2007) "Cryptographic Security for a High-Performance Distributed File System" Proceedings of 24th IEEE Conference on Mass Storage Systems and Technologies, 2007, MSST 2007. (S. 227 - 232). San Diego, CA : IEEE.

[9] Tseng, F.-H., Chen, C.-Y., Chou, L.-D., Chao, H.-C. (2012). "Implement A Reliable and Secure Cloud Distributed File System". Proceeding of the 2012 International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), (pp. 227 - 232). New Taipei : IEEE.

[10] Warner, B., Wilcox-O'Hearn, Z. (2008) "Tahoe – The Least-Authority Filesystem", Proc. of the 4th ACM international workshop on Storage security and survivability (pp. 21-26). Alexandria, VA, USA