Securing personal data in public cloud

Péter Vörös a, Dániel Csubák b, Péter Hudoba a and Attila Kiss a,c

aDepartment of Information Systems, Eötvös Loránd University, Budapest, Hungary; bSymmetria Hungary Co. Ltd., Budapest, Hungary; cDepartment of Mathematics and Informatics, J. Selye University, Komárno, Slovakia

ABSTRACT

More and more data are being stored in various cloud storages and they make it impossible to personally take care of data security, since most of these service providers do not offer any options for users to set up trusted encryption. We offer two types of solution to protect data, a client-side encryption and a security middleware. OpenWebCrypt is a browser extension designed to secure private data so it does not leave the client browser. Our security middleware, CrypStore PI offers a flexible solution to encrypt all the data that leaves the trusted LAN. By doing so, both guarantee that neither an attacker on the untrusted channel, nor the third-party external service provider has access to those. Furthermore, by separating the keys and the encryption itself from the user machines to dedicated hardware, CrypStore PI also improves the level of protection. Our goal is to offer easy to use, configurable and affordable solutions for personal users or small enterprises who would like to take responsibility for the protection of their data stored in public clouds. Because of the limited resources, we analysed the performance of today’s most used symmetric key encryption algorithms to be able to recommend the best choice.

1. Introduction

The amount of data produced by every single end-user is growing day by day. Social networks, IoT, sensors, etc. are all responsible for this growth, but many home users are not aware of the importance of the data they own. Companies spend a significant sum on securing their data, based on their business sector, size and budget because they must protect their industrial secrets, intellectual properties, etc. We believe the same should apply to the home users since a significant proportion of the data theft attempts are targeting them because not only the data of the companies represent business value, but also theirs. Forrester Consulting, showed in their study the most valuable customer identification data for marketing purposes by asking more than a hundred B2C directors (Forrester consulting: Most valuable customer data, 2017). Until 2018 billions of people’s data had been stolen, as it is shown in the CSO’s report (Cso: Biggest data breaches of the 21st century, 2017), which highlights the need for their data to be secured.

CONTACT  Péter Vörös vpetyaa@gmail.com

© 2019 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
Looking at data storing trends, we found that alongside most of the home users, even some companies use third-party external stores, most of which are cloud-based. EU statistics from 2014 show that in the region an average of 21% use cloud storages for storing files, while the maximum value is 42% (EUStats: Use of internet cloud storages, 2014). For end-users Dropbox, eDrive, and Google Drive are amongst the most popular choices, which is quite understandable because of the high availability and universal accessibility of these services and the fact that these service providers offer some storage capacity for free. Despite their data security measures, the user cannot really choose how their data is to be stored and protected since they are not responsible for these acts. Due to this, it may be possible for the service provider to have access to the user’s data. Not just the stored data but also the channel which is used to reach these third-parties can be compromised, if its security is not guaranteed by the service provider. In this case, the data can become accessible by man-in-the-middle attackers.

Hardware Security Modules (HSM) are enterprise solutions for the stated problem, but their price can be high for even small, and mid-sized companies, and clearly is for a home user. While the idea of separating the keys of the encryption algorithm to a dedicated machine is glorious, the price is unfortunate – this is why we choose to create a tool by using a dedicated Singe Board Computer (SBC).

The users need their data to be quickly available, yet secure. The base of data protection is the encryption, and the decryption algorithms, this is the reason behind our analysis and comparison of the performance of today’s most widely used algorithms with different file sizes on a specific architecture that is used by our SBC. Our goal is to find the best solution for the limited computing and memory capacity of an SBC, to use it in our tool.

The paper is organised as follows: In Section 2, we present some of the related papers in this topic. Later in Section 3, we describe the models for our solutions which we introduce in Section 4. In Section 4.1, we show our client-side data encryptor OpenWebCrypt. In Section 4.2, we introduce the general concept of our proxy like security middleware CrypStore PI. We present the main functionalities of the tool, such as cloud storage combination, to merge space, offered by different service providers to one, and the replication feature of CrypStore PI. In Section 4.2.5, we analyse the security of the CrypStore PI, presenting its durability against the different types of attacks and we also compare the various encryption algorithms that can be used. In Section 4.3, we compare the runtimes of the stream ciphers on our HSM. Finally, we conclude our work in Section 5.

2. Related work

Privacy issues in public clouds have had a huge attention throughout the past years. There are multiple surveys and papers about the topic. Most of them state that the main problem is that the providers are not transparent enough, lacking the capabilities for the tracking and auditing the access history of both the physical and virtual servers. Cloud Security Alliance collects the biggest threats of cloud computing from year to year. In the latest (2016) list (The Treacherous Twelve Cloud – Computing Top Threats in 2016, 2016), they collected the 12 highest priority issues. These are the following:

(1) Data Breaches
(2) Weak Identity Credential and Access Management
(3) Insecure APIs
(4) System and Application Vulnerabilities
(5) Account Hijacking
(6) Malicious Insiders
(7) Advanced Persistent Threats (APTs)
(8) Data Loss
(9) Insufficient Due Diligence
(10) Abuse and Nefarious Use of Cloud Services
(11) Denial of Service (DoS)
(12) Shared Technology Issues

In our previous works, we analysed both the Account Hijacking (Vörös & Kiss, 2016), and DoS/DDoS problems (Csubák, Szücs, Vörös, & Kiss, 2016; Vörös, Laki, & Kiss, 2017). These fields still require a lot of improvements, but now in this paper we set the focus on the highest threat, Data Breaches.

Any data that is stored unencrypted on the servers are potential targets of such incidents, therefore some extra security seems to be a necessary idea. There are numerous papers aiming to summarize the potential cloud threats. In Ko, Lee, and Pearson (2011), Ren, Wang, and Wang (2012), Kalloniatis et al. (2014) and Sun, Chang, Sun, and Wang (2011), they deeply analyse the different security issues, and possible solutions. In Ko, Jagadpramana et al. (2011), a framework is offered for cloud services that satisfies most of the privacy and stability needs. The idea of building our services above a trusted framework is really nice, however, unfortunately it is not widely used. The reason can be the need to rewrite existing applications which is a huge work, so we believe we need some user-side protection instead.

During our research, we found that there are some proprietary pieces of software, for example, Odrive or MultCloud, that implement some parts of OpenWebCrypt’s or CrypStore PI’s functionality. Some offer to merge storage spaces, others encrypt the uploaded files, but most of those that offer encryption also store the key files in their own space instead of letting the user keep them for their own. While this allows a wider accessibility, it also raises questions about security, since the user cannot be sure that their files are inaccessible by the service provider.

There are several papers that aim to analyse the security challenges for public clouds (Anthes, 2010; Chen & Zhao, 2012; Jansen & Grance, 2011; Ren et al., 2012), all of them stating the importance of encrypted data storing. Kamara and Lauter (2010) describe a possible way to build a cryptographic cloud storage. Wang, Chow, Wang, Ren, and Lou (2013) propose a secure cloud storage system supporting privacy-preserving public auditing. Xiong, Zhang, Yao, Wu, and Wen (2012) propose a k-out-of-n secret sharing solution in CloudSeal for Amazon Web Services, including EC2, S3, and CloudFront. These show promising results, but their approach to the point of the extended security layer differs from ours, because they do not intend to offload the encryption to a dedicated machine.

The most similar solution is Crypto Bone (The Crypto Bone – Privacy and secure communication under your control, 2017), which is software meant to be installed on an external device to make confidential communication secure and usable. Crypto Bone can be installed on Raspberry PI, Beagle Bone, or a Linux machine. It encrypts the messages between individuals selected by an initial secret agreed on by both parties in the
communication. Crypto Bone also stores the keys on the external device, furthermore, it generates new random keys for every message, and stores them in the memory. Although its use case is different from CrypStore PI’s, many times we had to consider the same problems as the developers of Crypto Bone.

3. Our models
Enterprises usually have strict rules and try to limit user errors as much as possible. Therefore, it is a common characteristic that they offer pre-installed systems fully over-watched by the administrators, revoking superuser rights from employees. They shall provide security to their users’ data by providing some sort of securing service, while it is still important to have an easy-to-use solution. We assume that the employees have the trust in their company/LAN.

As one can see in Figure 1, the internal network/trusted LAN – depicted as the rig rectangle – has several clients, all of which has its own encryption keys (these are colour-coded; Client1: lightgrey, Client2: grey, Proxy: line-filled). When they communicate with either internal or external servers they use their keys for encryption/decryption. Blank file means the unencrypted request, while colour-coded files stand for encrypted/decryp- table requests with the specific key.

3.1. Internal server case
Internal requests (Client2) are handled without reaching the proxy server because the requests do not go through the boundaries of the protected network so there is no

Figure 1. How data encryption works using a proxy server.
need for extra encryption. We assume that the internal server accepts only encrypted requests for security hardening. A typical request-response looks like this:

(1) Client2 prepares to send a request to an internal server
(2) Client2 encrypts the whole data with its darkgrey key
(3) The request is NOT routed through Proxy server but goes straight to the service provider
(4) The service provider processes the request
(5) The service provider sends back the encrypted answer to Client2
(6) Client2 decrypts the message with its key
(7) Client2 has the unencrypted message

### 3.2. External server case

Plenty of data have to be encrypted before sending it to an external service, in this subsection, we consider two different models for the external server case.

#### 3.2.1. Protection by local proxy

The proxy has to be in the secure LAN to use this model because in this solution the proxy server is able to see unencrypted requests, and this information can be used for various things if it falls into the wrong hands.

Outgoing requests using special services (e.g.: Google calendar or eDrive) are forwarded through an internal proxy server. This server has its own (line-filled) encryption key but also has a master key (first key of CrypStorePI in Figure 1) that is able to decrypt all the internal requests. How a typical conversation looks like:

(1) Client1 sends a request to gain an extra level of security service
(2) Client1 encrypts the whole data with its lightgrey key
(3) The request is routed through the Proxy server
(4) Proxy server uses the master key to decrypt the whole request
(5) Proxy server uses the master key for encrypting the request’s payload only (e.g.: the content of a calendar event)
(6) Proxy server encrypts the whole request with its line-filled key
(7) Proxy server sends the request to the service provider
(8) The service provider processes the request
(9) The service provider sends back the encrypted answer to the proxy server
(10) Proxy server decrypts the message with its line-filled key
(11) Proxy server decrypts the payload with the master key
(12) Proxy server encrypts the message with Client1’s (lightgrey) key
(13) Client1 decrypts the message with its own lightgrey key
(14) Client1 has the unencrypted message

It can clearly be seen that clients do not notice the extra layer of security, therefore they do not need any extra configuration, which makes it a great solution.
3.2.2. Local encryption

Individual users have full control over their machines, they can install software, and browser extensions freely. It is crucial to make our solution easily installable, since we cannot assume any IT pre-knowledge from these users.

As it can be seen in Figure 2, the encryption and decryption do not necessarily need to be propagated to a dedicated server, but the browser can do it in the background. An example service usage looks like this:

(1) Client sends a request to an extra level of security service
(2) The client’s browser encrypts the payload of the data with a user password or key
(3) The client’s browser also encrypts the whole data by its key
(4) Client sends the request
(5) The service provider processes the request
(6) The service provider sends back the encrypted answer to the client
(7) The client’s browser decrypts the whole message with its key
(8) The client’s browser decrypts the payload with a user password or key
(9) Client has the unencrypted message

4. Applications based on our models

In this section, we describe in detail how we designed the actual applications based on the models described above.

4.1. OpenWebCrypt

The goal of OpenWebCrypt (OWC) is to give a lightweight client-side method to make our data secure, unanalyzable and unusable by service providers.

OpenWebCrypt is a browser extension, that is based on the JavaScript framework jQuery, manipulates the visited webpage (e.g. Google Calendar), finds the input boxes and before every communication step it replaces the data with an encrypted/obfuscated version of the data, so it will not be visible for an adversary. For encryption, we have used stream ciphers.
As initialization vector and key we used a user passphrase and DOM element dependent information that has high entropy on the page (e.g. Google calendar event ID).

OpenWebCrypt works invisibly in the background. In Figure 3, one can see that adding a new entry can be done the same way as without using our script. Every new event we create will be stored with the proper encryption. In Figure 4(a), one can see how the encrypted calendar looks like if we choose no to decrypt it. That is what is stored on the servers. And in Figure 4(b), this is what the user can see if he uses the correct key for decryption.

The method is really easily portable because the encryption can be transferred with copying the passphrase to another machine. Cross-platform support is guaranteed because all major browsers allow JavaScript execution. Also Tampermonkey, the userscript manager that we use to inject our scripts, is compiled for Firefox, Chrome, Opera, Microsoft Edge.

4.1.1. Security analysis
This approach can be attacked only from inside the user's own machine because the used passphrase and the plain unencrypted data can be found only on it. Every information that leaves the machine is already encrypted. The DOM element depending initialization of the ciphers is preventing the statistical analysis of the encrypted data.

Cross Site Scripting (XSS) attacks may affect OpenWebCrypt if the service provider has XSS vulnerability. Attackers may retrieve the encryption key from our script with such a method, however OWC does not introduce any new vulnerabilities.

4.2. CrypStore PI
In Figure 1, the big rectangle is the trusted LAN, containing various clients like a PC, a laptop, or any kinds of smart devices, resources like private servers, or an NAS and the...
CrypStore PI. In our model, we assume that the user is responsible for the safety of their trusted private network.

As it is shown in Figure 1, the CrypStore PI acts as a gateway to ensure that all the data that leaves the trusted LAN to be stored at an external storage is not accessible by any 3rd parties, whether they try to access it at the channel (which might not be encrypted) or at the external storage. Our tool guarantees this by encrypting the user’s files that are to be stored externally inside the trusted private network and decrypting them only when they arrive back into it. To be able to support multiple service providers we separated the service dependent (interface) and the core parts of the software. The core acts as the skeleton of CrypStore PI, it handles user commands and provides all the features, while the interface library implements the API calls for each actual service. Due to this design, any new service can easily be supported by implementing its interface library.

Since the most crucial part of the used encryption algorithms is to guarantee the reliability of the keys, they are stored in the CrypStore PI. This way even the rightful user has no need to access their key files directly since one can use the encryption as a service provided by our tool. While the limited access to the keys improves their security, it raises questions about restoring keys after a hardware error, or about the protection of the CrypStore PI itself. We will present our solution for these problems later in this section.

For the implementation of our prototype, we chose Orange PI PC by Shenzhen Xunlong Software Co. Ltd. (Shenzhen Xunlong Software Co. Ltd.: Orange PI PC, 2016). This SBC has an ARM Cortex-C7 Quad-core processor with 1.6 GHz for each core, 1 GB of DDR3 SDRAM, a 100 Mb Ethernet port, 3 USB 2.0 ports and it stores all its data on an SD card. For the operating system, we chose the highly optimized Diet PI, which uses less memory, storage, CPU capacity and has more optimized software than Raspbian Lite (Diet PI vs Rasbian Lite, 2015).

4.2.1. Combine cloud storages
CrypStore PI is able to handle multiple users and multiple service providers for each user. Let \( m \) be the number of the users and \( \text{serviceNum}_i \) be the number of different service providers for user \( i \) \((i \in 1, \ldots, m)\). \( \text{ENC}() \) and \( \text{DEC}() \) are the encryption and decryption methods while \( \text{KEY}[i] \) (that can be the key and initialization vector packed together) is the encryption key used for user \( i \). \text{UPLOAD()} (Algorithm 1) and \text{DOWNLOAD()} (Algorithm 2) are the methods used by CrypStore PI to store and to retrieve data from the storage.

Algorithm 1: Explanation of UPLOAD() function

\[
\begin{align*}
\textbf{Data:} & \quad \text{FileToUpload, UserId} \\
\textbf{Result:} & \quad \text{The encrypted file is split, the encrypted pieces are uploaded to the appropriate server} \\
\quad & \text{serviceNum} := \text{the number of services for UserId} \\
\quad & \text{encryptedFile} := \text{ENC}(	ext{FileToUpload, KEY[UserId]}) \\
\quad & \text{pieces} := \text{SPLIT}(\text{encryptedFile, serviceNum}) \\
\quad & \text{for } i \text{ in } 1..\text{serviceNum} \text{ do} \\
\quad & \quad \text{Send}(\text{pieces}[i], \text{Servers[UserId]}[i])
\end{align*}
\]
Algorithm 2: Explanation of DOWNLOAD() function

**Data:** FileToDownload, UserId

**Result:** Encrypted file pieces are downloaded from the servers, the pieces are joined together, and decrypted

\[ \text{serviceNum} := \text{get the number of services for UserId} \]

\[ \text{for } i \text{ in } 1..\text{serviceNum} \text{ do} \]

\[ \text{pieces}[i] := \text{Receive(Servers[UserId][i])} \]

\[ \text{encryptedFile} := \text{JOIN(pieces)} \]

\[ \text{file} := \text{DEC(encryptedFile, KEY[UserId])} \]

Return (file)

Using several providers highly increases the security of our system because attackers have to get credentials or break into several servers to get the whole encrypted data. Another great impact of this method is that the storage offered by free services are limited if one store just \(1/\text{serviceNum}\) part of each file on each server it results in an increase of the available storage space.

The split function can be of any kind and the most trivial way is splitting up the encrypted data to \(\text{serviceNum}\) similar sized pieces.

4.2.2. Replication method

Our security model brings a serious single point of failure problem. Because of the users’ inability to access their keys, if something happens to the hardware (SD card error, hardware problems) users may end up in a situation where they are unable to decrypt their files.

To overcome this problem, we designed a replication procedure that helps to create a replica of the originally prepared system in an easy way. If the user connects another PI to the trusted LAN, the client is able to initiate the process.

The procedure synchronizes the keys over an encrypted channel. CrypStore PIs by default are set up with a certificate, issued to their hardware ID, so it is not possible for any other machine on the network to be identified as a CrypStore PI. This whole procedure still prevents users from compromising their data since the keys can only travel on an encrypted channel, and both ends of the channel use the same level of security.

The second PI can be used as a reversed backup or can be set up in a new LAN letting users synchronize their data in a different location.

4.2.3. Fragmenting

To increase the protection against known-plaintext attacks we can change the SPLIT and JOIN functions in Algorithms 1 and 2. With this extension, the adversary cannot restore a key part of one service only. To raise the security level, we use the following SPLIT and JOIN:
Algorithm 3: Explanation of MODSPLIT() function

Data: Data, n

\( \text{len} := \text{Size of Data in bytes} \)

for \( i \) in 1..\( \text{len} \) do

\( \text{pieces}[i \mod n] := \text{Data}[i] \)

Return \( (\text{pieces}) \)

4.2.4. Key size consequences

We have a special case of practical security, because we do not have to get rid of the biggest weakness of the symmetric encryptions, the need to have the same key at both sides.

**e Time Pad** The unconditionally secure encryption method, named e Time Pad can be used if we have a longer key size than the data that has to be encrypted. We have to make sure that we do not use any part of that again. We can use different starting points for the key based on a hash function, but that still can be attacked by plain text analysis if we store more data than the key file.

**Truncated SHA-2 hash function** If we are using for example ChaCha8 as encryption method, we need a 256 bit sized key and a 64 bit sized initialization vector (IV). The following technique is similar if you use a different encryption method. Based on a hash function, we can get a 256 and 64 bit sized part from the key file (the hash gives the position in the file) and use that as seed and IV. For example, if we have a \( 2^n \) sized key file, we calculate the SHA-256 (Gallagher & Director, 1995) hash of the full path of the file that we want to encrypt and use the first \( n \) bits to get the position of the key, and the second \( n \) bits for the IV.

Algorithm 4: Explanation of MODJOIN() function

Data: DataPieces

\( \text{pieceNum} := \text{Number of pieces in DataPieces} \)

\( \text{pieceLengths} := \text{Lengths of the pieces} \)

\( \text{finished} := \text{False} \)

\( i := 1 \)

\( j := 1 \)

while NOT \( \text{finished} \) do

if \( \text{pieceLengths}[j] < i \) then

\( \text{finished} := \text{True} \)

else

\( \text{data}[i] = \text{DataPieces}[j][i] \)

\( j := j + 1 \)

if \( \text{pieceNum<j} \) then

\( i := i + 1 \)

\( j := 1 \)

Return \( (\text{data}) \)
The SHA-2 is well uniformed, and the truncating does not decrease that property. If the hash function is ‘perfect’, the probability of having the same key and IV is very low. The low collision probability increases the security of the system because if an adversary successfully decrypts one file from the services, it probably does not help to decrypt any other file. If we have a 500 MB sized key (easily affordable on a dedicated device), that means $2^{32}$ positions and $2^{64}$ combinations of key and IV. The method detailed in Algorithm 6.

Algorithm 5: Explanation of MODGET() function

Data: Data,pos,len

dataLen := length of data
withoutRotateLen := min(dataLen-pos,len)
ret[0..withoutRotateLen] := Data[pos..pos + withoutRotateLen]
if withoutRotateLen<len then
    ret[withoutRotateLen + 1..len] := Data[0.pos + (len − withoutRotateLen)]
Return (ret)

Algorithm 6: Explanation of KEYANDIV() function

Data: path

len := Size of the keyFile (size of the keyFile must be power of 2)
n := log_2(len)
hash := SHA256(path)
key := MODGET(keyFile, hash[0..n], 256)
IV := MODGET(keyFile, hash[n + 1..2 * n], 64)
Return (key, IV)

4.2.5. Security analysis

In this section, our goal is to analyse the possible attack types against the CrypStore PI.

Attack from the internal network If someone accesses the local network (The big rectangle in Figure 1) it may be possible to discover the CrypStore PI. e may try to connect to the device and compromise real users accounts.

CrypStore PI is only accessible from the internal network and it requires username and password for the authentication, which is different from the credentials used for the cloud service providers. We assume that users do not let others know their login data, therefore the only way is to crack in. CrypStore PI is configured to block a machine for a while after five unsuccessful login attempts, which makes brute force attacks ineffective if users choose their passwords wisely.

It is, however, possible, that the attacker is able to register a new user, and then grant access to use CrypStore PI, but each user’s data is strictly isolated, so no extra information is acquireable by doing so. We try to make our device as safe as possible, so we do not let any users access the machine’s internal data in any circumstances. It is not possible for the users to access or download their own keys, therefore they can not compromise their own credentials by mistake or by purpose.
Attack from the LAN can target the replication functionality of our system as well, which means they need to prepare a machine with a valid certificate issued to the hardware. Without this certificate, valid CrypStore PI users on the network cannot initiate the replication procedure. We keep the rights for ourselves to issue certificates, therefore it is not likely possible for attackers to create a valid one themselves. If somehow they manage to do so, a user still needs to start the copy to the new system, which is not likely if the new machine is brought in by another person.

**Physical attack against the hardware** Attacks against hardware can be divided into two cases: targeting one of the computers or targeting the CrypStore PI.

If someone manages to log in to a computer that is using CrypStore PI, the data is directly accessible. There is not much that our solution can do to users who are not protecting their machines enough. We assume that if one is paying attention to keep their data secured in the cloud, than they also keep their machines safe. Therefore, this security risk can be neglected.

OrangePi uses SD card as its internal storage, and it is possible to remove it from the device. If someone manages to get the card, the whole filesystem which includes the keys could become easy prey. Physical protection of the CrypStore PI is crucial.

**Man in the middle** A Man in the middle attack happens when a communication between two systems is intercepted by an outside entity. (Line-filled files with locks in Figure 1) Cloud services nowadays use SSL/TLS certificate to upgrade HTTP protocol to HTTPS. HTTPS is widely used and considered to be a secure communication channel. If the service does not have any mixed content or any page element loading over HTTP protocol, then that makes man in the middle attacks impossible.

We believe HTTPS is secure, but if somehow someone is able to break the encryption, CrypStore PI still keeps our data safe, because no unencrypted files travel outside of the local network. The authentication to the service provider, however, uses only HTTPS, therefore, in this case, it may be possible to access user credentials to a service.

**Accessing the clouds** In the clouds there cannot be any unencrypted user data, to make sure that it is uncompromisable by the provider or any third person. Data breach is considered to be the number one threat in the latest (2016) list provided by Cloud Security Alliance (The Treacherous Twelve Cloud – Computing Top Threats in 2016, 2016).

Definition 4.1A Data Breach is an incident in which sensitive, protected or confidential information is released, viewed, stolen or used by an individual who is not authorized to do so.

Users have the option to combine the storage of several different services and different accounts. The more targets we use, the more difficult it becomes for the attacker to get all the login information. The files are split into \( n \) pieces where \( n \) is the number of services, the pieces are encrypted and stored on \( n \) servers. The data is only decryptable if all the pieces are put together, so \( n \) usernames and passwords are required to do so.

It is not likely for anyone to obtain these, but if one does, it is still not possible to decrypt the whole file without the main encryption key \( key_{user} \) which, as we mentioned above, cannot be retrieved from the CrypStore PI. Without the key, the decryption complexity depends on the encryption method that we discuss in Section 4.3.
4.3. Runtime of stream ciphers

We have the encryption/decryption phases on the same machine in both of the methods, so we can use fast block or stream ciphers to keep the user’s data securely. There are many symmetric encryption methods in cryptography and nowadays the most widely used is the Advanced Encryption Standard (AES).

According to an existing benchmark (Bernstein & Lange, 2014), we can observe that the AES is one of the fastest algorithms in many of the cases. There is a family of newcomer competitors that are selected for the eSTREAM Portfolio (Babbage et al., 2008) which are really good in ARM processor cases.

4.3.1. Public benchmark

ChaCha8 (Bernstein, 2008) published by Daniel J. Bernstein gives 128 bit security level, which is a variant of the Salsa20 (Bernstein, 2008), looks to be a better choice for ARM processors based on the published measurements. The 20 round version of this algorithm, named ChaCha20, is used by Google for HTTPS connections between Android versions of Chrome and Google (Bursztein, 2014, April). The algorithm is based on add-rotate-xor, therefore it can neither be corrupted by time-based attacks.

Note that the Table 1. ‘shows the speed of encrypting long messages; this is computed as 1/3072 of the difference in cycle counts between encrypting 4096-byte messages and encrypting 1024-byte messages’ (from bench.cr.yp.to).

4.3.2. Our benchmark

We completed our benchmark (Table 2) on the target device and the Sosemanuk is the fastest, which is also the second on the public benchmark and also has 128 bit security level.
5. Conclusion and further work

In our work, we presented solutions against data breach which is the number one online threat that targets personal data. We defined various network models and showed how an extra security layer fits in, to improve the security of personal data stored in public cloud. Our prototype is OpenWebCrypt, which is a client-side lightweight browser extension for individual users. CrypStore PI is our proxy-type encryption box, that makes sure that unencrypted files do not leave the trusted network. We use algorithms to store encrypted data split amongst multiple public cloud storages. We also showed the performance results of various encryption algorithms running on our dedicated hardware. Lastly for both models we analysed the possible new attacking methods that our solutions introduced.

In further work we plan to:

- extend the OpenWebCrypt with Steganography like methods to avoid detection of the encryption if an adversary breaks into the user’s online profile and
- add new algorithms to the CrypStore PI to avoid the statistical analysis of the fragments.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002). The authors also wish to thank Symmetria Hungary Co. Ltd. for their support.

Notes on contributors

Péter Vörös received the M.Sc. and Ph.D. degrees in computer science from the Doctoral School of Computer Science, Eötvös Loránd University, Budapest, Hungary in 2014 and 2019, respectively. He is currently an Assistant Lecturer of the Department of Information Systems. He is currently working on projects in network security, traffic analytics, and programmable data planes.

Dániel Csubák received the M.Sc. degree in Computer Science from Eötvös Loránd University, Budapest, Hungary, in 2014, and graduated from the Doctoral School of Computer Science, Eötvös Loránd University, in 2018. He is currently pursuing a Ph.D. degree with the Department of Information Systems.

Table 2. Benchmark of well-known stream chiphers on Orange Pi with Crypto++.

| MiB per Second | Stream chipher |
|----------------|----------------|
| 106            | sosemanuk      |
| 99             | salsa208       |
| 88             | panama         |
| 78             | salsa2012      |
| 58             | salsa20        |
| 16             | aes128ctr      |
| 14             | aes192ctr      |
| 13             | aes256ctr      |
| 9              | chacha8        |
| 8              | chacha12       |
| 5              | chacha20       |
Systems, Eötvös Loránd University. His current research interests include different aspects of IT security such as firewalls, intrusion detection systems, and cloud security. He currently works as an IT security system engineer.

Péter Hudoba received the M.Sc. degree in computer science from Eötvös Loránd University in 2015 and currently pursuing a Ph.D. degree. He is an Assistant Lecturer with the Department of Computer Algebra, Eötvös Loránd University. He is currently working on projects in cryptography and network security.

Attila Kiss was born in 1960. In 1985 he graduated (MSc) as a mathematician at Eötvös Loránd University, in Budapest, Hungary. He defended his Ph.D. in the field of database theory in 1991; his thesis title was Dependencies of Relational Databases. Since 2010 he is working as the head of the Information Systems Department at Eötvös Loránd University. His scientific research is focusing on database theory and practice, security, semantic web, big data, data mining, artificial intelligence, and bioinformatics. He was the supervisor of 7 students who received a Ph.D. He has more than 130 scientific publications.

**ORCID**

Péter Vörös [http://orcid.org/0000-0001-7539-9878](http://orcid.org/0000-0001-7539-9878)
Dániel Csubák [http://orcid.org/0000-0003-2331-5139](http://orcid.org/0000-0003-2331-5139)
Péter Hudoba [http://orcid.org/0000-0001-5810-4193](http://orcid.org/0000-0001-5810-4193)
Attila Kiss [http://orcid.org/0000-0001-8174-6194](http://orcid.org/0000-0001-8174-6194)

**References**

Anthes, G. (2010). Security in the cloud. *Communications of the ACM*, 53(11), 16–18.

Babbage, S., Canniere, C., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., ..., Robshaw, M. (2008). The eSTREAM portfolio. *eSTREAM, ECRYPT Stream Cipher Project*.

Bernstein, D. (2008, January). Chacha, a variant of salsa20.

Bernstein, D. J. (2008). The Salsa20 family of stream ciphers. In New Stream Cipher Designs (pp. 84–97). Berlin; Springer.

Bernstein, D. J., & Lange, T. (2014). eBACS: ECRYPT benchmarking of cryptographic systems (2011). Retrieved from [http://bench.cr.yp.to](http://bench.cr.yp.to). *Citations in this document*, 1(1):1.

Bursztein, E. (2014). *Speeding up and strengthening https connections for chrome on android* (Technical report). Retrieved from [https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html](https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html)

Chen, D., & Zhao, H. (2012). Data security and privacy protection issues in cloud computing. In 2012 international conference on computer science and electronics engineering (ICCSEE) (Vol. 1, pp. 647–651), Hangzhou.

The Crypto Bone – Privacy and secure communication under your control (2017). Retrieved from [https://crypto-bone.com/](https://crypto-bone.com/)

Cso: Biggest data breaches of the 21st century (2017). Retrieved from [https://images.idgesg.net/images/article/2017/10/biggest-data-breaches-by-year-and-accounts-compromised-1-100738435-large.jpg](https://images.idgesg.net/images/article/2017/10/biggest-data-breaches-by-year-and-accounts-compromised-1-100738435-large.jpg)

Csubák, D., Szűcs, K., Vörös, P., & Kiss, A. (2016). Big data testbed for network attack detection. *Acta Polytechnica Hungarica*, 13(2), 47–57.

Diet Pi vs Rasbian Lite (2015). Retrieved from [https://docs.google.com/spreadsheets/d/1mDHGZC-H6tU6_O8kuLTG8d4A8Nt7IV1Q7MXTR_6qw30/edit#gid=0](https://docs.google.com/spreadsheets/d/1mDHGZC-H6tU6_O8kuLTG8d4A8Nt7IV1Q7MXTR_6qw30/edit#gid=0)

EUStats: Use of internet cloud storages (2014). Retrieved from [http://ec.europa.eu/eurostat/statistics-explained/index.php/Internet_and_cloud_services_-_statistics_on_the_use_by_individuals](http://ec.europa.eu/eurostat/statistics-explained/index.php/Internet_and_cloud_services_-_statistics_on_the_use_by_individuals)

Forrester consulting: Most valuable customer data (2017). Retrieved from [https://www.marketingcharts.com/wp-content/uploads/2017/06/LiveIntentForrester-Most-Valuable-B2C-Customer-Identification-Data-June2017.png](https://www.marketingcharts.com/wp-content/uploads/2017/06/LiveIntentForrester-Most-Valuable-B2C-Customer-Identification-Data-June2017.png)
Kalloniatis, C., Mouratidis, H., Vassilis, M., Islam, S., Gritzalis, S., & Kavakli, E. (2014). Towards the design of secure and privacy-oriented information systems in the cloud: identifying the major concepts. *Computer Standards & Interfaces*, 36(4), 759–775.

Kamara, S., & Lauter, K. (2010). Cryptographic cloud storage. In *International conference on financial cryptography and data security* (pp. 136–149). Berlin; Springer.

Ko, R. K., Jagadpramana, P., Mowbray, M., Pearson, S., Kirchberg, M., Liang, Q., & Lee, B. S. (2011). Trustcloud: A framework for accountability and trust in cloud computing. In 2011 IEEE world congress on services (SERVICES) (pp. 584–588), Washington, DC.

Ko, R. K., Lee, B. S., & Pearson, S. (2011). Towards achieving accountability, auditability and trust in cloud computing. In International conference on advances in computing and communications (pp. 432–444), Berlin.

Ren, K., Wang, C., & Wang, Q. (2012). Security challenges for the public cloud. *IEEE Internet Computing*, 16(1), 69–73.

Shenzhen Xunlong Software Co. Ltd.: Orange Pi PC (2016). Retrieved from http://www.orangepi.org/orangepipc/

Sun, D., Chang, G., Sun, L., & Wang, X. (2011). Surveying and analyzing security, privacy and trust issues in cloud computing environments. *Procedia Engineering*, 15, 2852–2856.

The Treacherous Twelve Cloud – Computing Top Threats in 2016 (2016). Retrieved from https://cloudsecurityalliance.org/group/top-threats

Vörös, P., & Kiss, A. (2016). Tookie: A new way to secure sessions. In Recent Developments in Intelligent Information and Database Systems (pp. 195–207). Da Nang, Springer.

Vörös, P., Laki, S., & Kiss, A. (2017). Distributed firewall on dataplane against ddos attack. *4th Winter School of PhD Students in Informatics and Mathematics* (p. 37).

Wang, C., Chow, S. S., Wang, Q., Ren, K., & Lou, W. (2013). Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers*, 62(2), 362–375.

Xiong, H., Zhang, X., Yao, D., Wu, X., & Wen, Y. (2012). Towards end-to-end secure content storage and delivery with public cloud. In Proceedings of the second ACM conference on data and application security and privacy (pp. 257–266), San Antonio.