Key-value Storage with Cryptographic Client-separation

Maximilian Blochberger

blochberger@informatik.uni-hamburg.de

January 12, 2019

A key-value storage is presented, which in contrast to traditional access mechanism uses cryptographic methods in order to separate client data. The presented key-value storage protects the confidentiality and integrity of users against malicious service operators as well as implementation errors in the server software.

1 Introduction

Key-value stores are a fundamental pattern in modern software development. They allow storing arbitrary data (values) under a given unique identifier (key). Document storage systems are basically key-value stores as well, as they store documents (value) for a given file name and path (key).

Key-value storages are often offered as a service on the internet and can be used by multiple users simultaneously. In order to protect user data, access management has to be established. Traditional access management mechanisms link data stored to the service to a user account. Some services, such as Apple's iCloud or Amazon S3, offer end-to-end encryption, which protects data from being accessed by the service operator or in the case it gets leaked, e. g., as the result of an implementation error. Most services however, such as Dropbox or Amazon S3, offer encryption at rest (or server-side encryption), where the cryptographic key has to be transmitted as part of the request, or no encryption at all. Encryption at rest might protect against some accidental data leakage, but does not protect against a malicious service operator.

Since service operators and application developers are responsible for protecting user data, especially since the new General Data Protection Regulation (GDPR) is in effect. Therefore, they should be interested in solutions where they do not have access to data they do not need in order to provide the service's functionality (data minimization). In this paper a key-value storage is presented, which is fully functional but protects the confidentiality and integrity of user data in a way, that the service operator does not learn more than necessary for fulfilling the service's functionality.

2 Attacker Model

The attacker, against whom our system is still able to protect against, is an insider, e. g., the operator of the key-value storage. He can actively read and modify each key-value pair as well as network traffic. His computational complexity is limited and he cannot compromise the device used by the end user.

The proposed system protects the confidentiality and integrity of data submitted. The attacker is not able to link two files to the same user. Modifications of key-value pairs cannot be prevented, but will be detected by the user if they occur.

3 Cryptographic Client-separation

The key that is used to identify values stored to the key-value storage will be called name while the cryptographic key used for encrypting

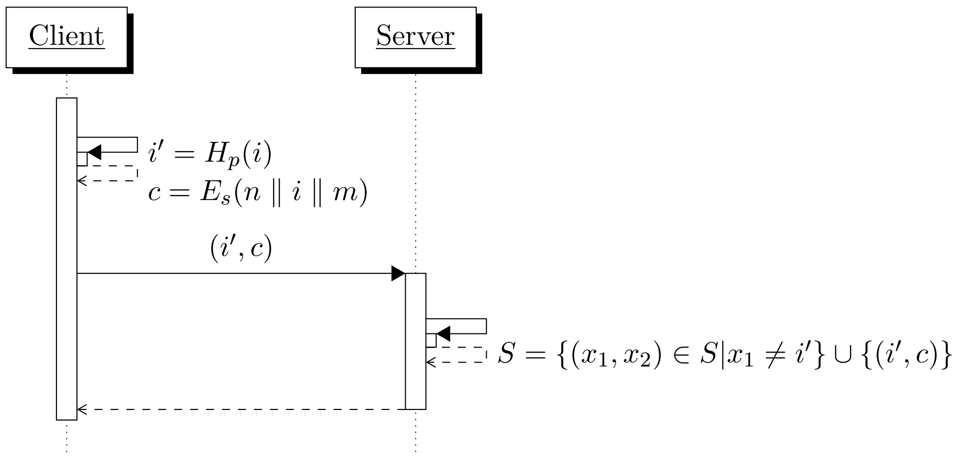
Client
Server
$i' = H_p(i)$
$c = E_s(n \parallel i \parallel m)$
(i', c)
$S = \{(x_1, x_2) \in S | x_1 \neq i'\} \cup \{(i', c)\}$

Figure 1: The process of storing a key-value pair.

values will be called secret key for disambiguation.

3.1 Encryption of Values

To protect the confidentiality of values stored to the key-value storage, each value m will be encrypted with a secret key s generated and stored on the user's device. The resulting ciphertext c can only be decrypted by the user in possession of the secret key. In order to avoid that the encryption of equal values will result in equal ciphertexts, a nonce n is used (IND-CPA).

To protect the integrity of values, an authenticated encryption system is used and the name i is added to the value prior to encryption.

$$c = E_s(n \parallel i \parallel m)$$

3.2 Personalization of Names

Names are developer- or user-chosen strings which are used to identify stored values, hence they can be protected using a one-way hashing function. Since there are no per-user name spaces on the server, they need to be globally unique (collision resistance). To avoid the same name of two different users or of two applications of the same user to point at the same remote value, a keyed hashing mechanism is used. The name i as well a personalization key p will be hashed with a cryptographically secure hashing function.

$$h = H_p(i)$$

3.3 Process

In order to store a value m for a given name i the name i is first hashed to i' using the personalization key p. The value m is then encrypted with the secret key s resulting in the encrypted value c. The protected key-value pair is transmitted to the server, where it is stored in the database S. This process is depicted in fig. 1.

In order to retrieve a value for a given name i, the name i is first hashed to i' using the personalization key p. The protected name is transmitted to the server. The server looks up the encrypted value c stored for the protected key i' and returns it to the client. The client can now decrypt the encrypted value c if it has not been modified, yielding the decrypted value m. This process is depicted in fig. 2

4 Demonstrator

In order to demonstrate the key-value storage presented a demo service and application were realized.

Cryptographic client-separation is not enough to protect values from being linked by an adversarial service operator. Values can be linked

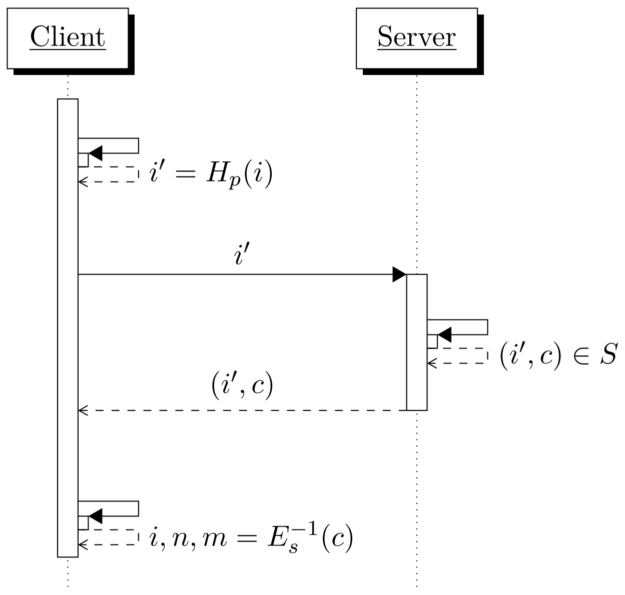

Figure 2: The process of retrieving a key-value pair.

by traffic data used in the communication between the server and the client as well. Therefore, an anonymization network such as Tor should be used for communication. Support for a lightweight anonyization proxy based on the work of Panchenko et al. [3] has been added to the client framework. A simple Apache or Squid proxy server can be used for that purpose.

4.1 Service

The key-value storage service was implemented as an open source project written in Python[1]. It is a simple web service that offers a REST-style API for creating, retrieving, updating, and deleting key-value pairs (CRUD). The service simply stores arbitrary binary values for given keys in an SQLite or Postgres database and returns them if requested by a given key.

4.2 Client Framework

The client code has been implemented as an open source software library written in Swift for iOS and macOS[2]. It is planned to release a library written in Java Android as well, which has not been published at the time of writing.

It offers a simple API for using the presented key-value storage with an instance of the provided service. In addition, it also takes care of securely storing cryptographic material in the device's keychain, which is protected by the tamper-resistant co-processor called Secure Enclave on iOS and macOS devices [1] if available.

4.3 Demo Application

An open source demo application has been written for iOS[3]. An application for Android has been written as well, in order to demonstrate that the key-value storage is platform-independant. The Android version has not been published at the time of writing.

The application offers a simple todo list where tasks can be added and marked as completed. Each change is propagated to the remote key-value storage.

[1] AppPETs/PrivacyService: Implementation of privacy-friendly services: `https://github.com/AppPETs/PrivacyService`

[2] AppPETs/PrivacyKit: Framework offering easy to use privacy enhancing technologies (PETs) for iOS and macOS applications: `https://github.com/AppPETs/PrivacyKit`

[3] AppPETs/Todo-iOS: Todo-list app for iOS demonstrating a secure key-value storage: `https://github.com/AppPETs/Todo-iOS`

Upon first launch of the application a master key is generated and persisted in the devices keychain. The secret key as well as the personalization key are derived from the master key given an app-specific context. This allows fine-grained control to the developer over whether the key-value store can be used in different applications. The master key can be exported as a QR code, so that the todo list can be synchronized between multiple devices of the same user. This works across different operating systems. The QR code is not shown immediately. The device owner has to authenticate beforehand by entering the device's passcode or scanning his fingerprint, depending on the devices capabilities and configuration. This functionality is also provided by the client framework.

5 Security Evaluation

It is assumed that for communication between the client and the service an anonymous communication network is used and our attacker model is not stronger than that of the anonymization network. Raymond [4] provides an overview of potential attacks on anymization systems.

5.1 Confidentiality

The values can only be decrypted by parties that are also in possession of the secret key s. The secret key s is stored on the user's device. It is assumed that the user does not export the secret key if he is under surveillance. A mechanism for sharing secrets between multiple devices using QR codes under adversarial conditions is proposed by Blochberger [2]. The attacker could guess the type of the value by inferring it from the value's size, e. g., if the value is about 700 MiB it could be a CD image.

Neither the service operator nor anyone else except who is in possession of the cryptographic keys can link two files. The identity of the user is protected by the anonymization network. The attacker can de-anonymize a single user by compromising the availability of a single key-value pair, presuming that the user complains about the reduced availability of the service. If enough users use the service, this attacks becomes infeasible. The protected name i' has a length of 256 bit. The probability for a collision, meaning $H(i_1) = H(i_2)$, is 50 % after 2^{128} tries (birthday paradox), making brute-force infeasible for a computationally limited attacker.

5.2 Integrity

The attacker can change key-value pairs. The change will be detected by the user, as the authenticated encryption scheme used will fail to decrypt if the ciphertext has been modified. Assuming that the attacker knows that two key-value pairs $(H_p(i_1), c_1)$ and $(H_p(i_2), c_2)$ belong to the same user, he could swap the values of these two pairs, so that $(H_p(i_1), c_2)$ and $(H_p(i_2), c_1)$. Since protected names are globally unique, it is ensued that $i_1 \neq i_2$. The client can now decrypt $i', n, m = E_s^{-1}(c_2)$ and will detect the change since $i_1 \neq i'$[4]. If the attacker swaps values of two different users, the values cannot be decrypted since different secret keys are used, hence the modification will be detected.

5.3 Availability

As the attacker controls the service, he can influence availability at his will.

6 Limitations

The key-value storage presented here has some limitations as well.

First, payment mechanisms have not been taken into account. Since the service operator cannot link files to users, he cannot simply calculate if paid-for quotas have been reached. In order to operate this service commercially and to somehow limit users from using too much space for free, future work should investigate how this is possible to achieve in a privacy-friendly manner.

Second, the service does not allow users to share values with one another. Currently there is no

[4]Note that this kind of integrity protection is not yet implemented at the time of writing: `https://github.com/AppPETs/PrivacyKit/issues/8`

way of restricting access to a value, e. g., to providing read-only access to another user.

7 Conclusion

A key-value storage was presented, that ensures the confidentiality and integrity of user data stored. The realization follows the data minimization principle, so that only data required for fulfilling the service's functionality can be accessed by the service operator.

Acknowledgements

This work was done in the AppPETs project[5] and supported by the BMBF.

References

[1] Apple Inc. *iOS Security – iOS 11.4*. White paper. Aug. 2018. URL: `https://www.apple.com/business/docs/iOS_Security_Guide.pdf`.

[2] Maximilian Blochberger. *Sharing Secrets between Mobile Devices*. White paper. Aug. 2018. URL: `https://github.com/AppPETs/SecretSharing-Whitepaper`.

[3] Andriy Panchenko et al. "SHALON: Lightweight Anonymization Based on Open Standards." In: *ICCCN*. IEEE Computer Society, 2009, pp. 1–7.

[4] Jean-François Raymond. "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems." In: *Workshop on Design Issues in Anonymity and Unobservability*. Vol. 2009. Lecture Notes in Computer Science. Springer, 2000, pp. 10–29.

[5] AppPETs – Datenschutzfreundliche Smartphone Anwendungen ohne Kompromisse: `http://app-pets.org`