SEkey: a distributed hardware-based key management system

SEkey: A Distributed Hardware-based Key Management System

Matteo FORNERO
CINI Cybersecurity National Lab.
Turin, Italy
matteo.fornero@consorzio-cini.it

Nicolò MAUNERO
Politecnico di Torino
CINI Cybersecurity National Lab.
Turin, Italy
nicolo.maunero@polito.it

Paolo PRINETTO
Politecnico di Torino
CINI Cybersecurity National Lab.
Turin, Italy
paolo.prinetto@polito.it

Antonio VARRIALE
B5 Labs Ltd.
Ta' Xbiex, Malta
av@blu5labs.eu

***Abstract*—Cryptography plays a key role in all the aspects of today cybersecurity and any cryptographic approach relies on cryptographic keys, i.e., series of bits that determine how a plain text is encrypted and decrypted, according to an agreed algorithm. The secrecy and security of an encryption key are thus crucial and fundamental: if the cryptographic key is compromised and known, everyone can decrypt a text encrypted according to the strongest encryption algorithm. As a consequence, several Key Management Systems (KMS) have been developed to easily support the management of cryptographic keys, whose number is constantly increasing, due to the amount of devices and communications that take place today, even in very restricted contexts. SEkey is a key management system developed targeting a distributed environment, where it is possible to identify a single central manager that acts as a Key Distribution Center (KDC) and many users that locally store and manage their own keys. Users, to a certain extent, can also work 'offline' without being always in direct communication with the central manager. SEkey is built leveraging the functionalities and physical properties of the SEcube™ Hardware Security Module (HSM). All the key values and critical information are stored inside the SEcube™ and never leave the device in clear, and all the cryptographic operations are performed by the SEcube™ itself. The guidelines provided by NIST where followed during the whole development process, guaranteeing all the most important security features and principles.**

I. Introduction

The increase in the number of connected devices, that has been taking place for several years now, is posing several security challenges by considerably enlarging the cyber attack surface. The quantity and quality of data exchanged every second among people and various devices is increasing at an exponential rate, making it mandatory to secure them.

Cybersecurity is a term that includes several concepts, but the fil rouge that connects them all is *cryptography*. As defined by NIST [4], cryptography is the discipline that embodies the principles, means and methods for the transformation of data in order to hide their semantic content, prevent their unauthorized use ore avoid their undetected modification. This data transformation process takes place through mathematical operations, more or less complex, that combine together the input data, usually referred to as cleartext, and the cryptographic key to obtain the modified data as output, what is usually referred to as ciphertext.

The cryptographic key is a parameter used in conjunction with a cryptographic algorithm that determines its spectrum of operation [7]. Drawing a parallel with everyday life, the role of a cryptographic key is similar to the key of a lock. Locking is like data encryption while unlocking is like data decryption and, just as in the case of a lock, also in cryptography protecting the key is of paramount importance: even in presence of the best encryption algorithm, if the cryptographic key is compromised and everyone knows it, then everyone can access the encrypted data.

Nowadays the amount of keys and the requirements for their security make it practically impossible to manage them by hand; for this reason the so-called *Key Management Systems* (KMS) were born, applications that aim to automate and simplify the management of cryptographic keys in highly complex contexts.

In this paper we present SEkey, a mixed hardware-software Key Management System, leveraging on the SEcube™[1] Hardware Security Module (HSM). SEkey is designed and developed having in mind a distributed ecosystem where each entity gets its own SEcube™ device. Each SEcube™ is in charge of securely storing all the encryption keys and of providing all the security primitives for securely managing keys and performing cryptographic operations. This allows to never expose the actual key value outside of the device when performing crypto operations. In addition, during the key distribution process, keys are over-ciphered with a unique key shared only by the administrator and the user that receive the update. Inside SEkey, two roles are available: the *security administrator* and the *user*. The former is in charge of distributing keys and synchronising all the SEcubes™, while the user passively uses its device for security purposes, everything related to the key management being automatically handled by the SEcube™ device.

The paper is structured as follows: the next section introduces a brief overview on the SEcube™ project and device. In the second section a brief analysis of the SOA on different types of KMS is proposed and the most important guidelines from the NIST for KMS development are reported. Then the implementation details and the most relevant features of SEkey

[1] https://www.secube.eu/

are presented, concluding with possible improvements and future works.

II. The SEcube™ Open Security Platform

The SEcube™ Open Security Platform [18] leverages on the functionalities of the SEcube™ SoC to provide a security-oriented open software and hardware platform. The SEcube™ SoC, developed by the Blu5 Group Company, includes three main cores:

- A STM32F4 microcontroller unit, equipped with an ARM Cortex-M4 processor.
- A reconfigurable hardware device (FPGA).
- An EAL 5+ certified Smart Card.

A 3D packaging of the three components and a set of custom technological solutions improve the resiliency to side-channel attacks [5] and to attempts to exfiltrate data from the device.

The SEcube™ platform is equipped with set of high-level APIs that abstract complex concepts of cybersecurity and cryptography [19], designed to ease the development of high security applications. Among the others, the open source libraries [8] include *SEfile* [9] and *SElink* [8], aimed at protecting data at rest and data in motion, respectively [20]. In particular, SElink provides a set of API that can be used to securely handle communications channels via end-to-end encryption, whereas SEfile provides a set of API for handling files in a secure way, allowing secure implementations of the most common system calls of the Posix Portable Operating System Interface and WIN32. These APIs are a simplified version of these system calls, not exposing all the functionalities provided by them, but managing internally all the security operations required to handle encrypted files.

III. Background

A. Key Management Systems Overview

Key Management Systems can be clustered according to different categories, including the way they are provided to the customers, the organization of the Key Distribution Center, and their key storage facilities.

According to the way a KMS is provided to the customers, four categories are mostly used: *software*, *virtual*, *appliance*, and *service* [6].

A *software KMS* is purely software-based and either implements its own protocol or is compliant with standard ones. The software runs on an Operating System (OS) that is hosting the KMS (typically, a server built by the customers to accommodate the KMS software).

A *virtual KMS* is a pre-installed virtual machine that runs the KMS software in a virtualised environment. The hardware where the VM runs is not shipped with the MKS and is under control and responsibility of the customers.

An *appliance KMS* is an integrated hardware-software solution. In this case both hardware and software are provided to the customer and they can be, for example, a server with certified hardware and software or a KMS running or leveraging on a hardware security module.

Type	Pros	Cons
Software	Wide compatibility	HW and OS provided by customer
	Runs on pre-existing hardware	Hardware may not be certified
	Runs on common OS	OS may not be certified
	Easy to fix and update	Usually weaker
Virtual	Wide compatibility	HW provided by customer
	Runs on pre-existing hardware	Hardware may not be certified
	Easy to run multiple installations	Usually weaker
	OS provided with the KMS	Virtualization overhead
	Easy to fix and update	
Appliance	HW and SW provided with the KMS	Lower flexibility
	Turnkey installation	Difficult to fix or update
	All-in-one solution	HW limitations
	Usually more secure	Usually more expensive
Service	No installation required	Keys stored in the cloud
	Easy to use	No physical control
	No local resources required	
	Flexible in terms of usage and payments	

TABLE I
Pros and cons of different kinds of KMS [6]

A *service KMS* is a cloud-based solution that can be used by the customers without the need of a specific hardware or infrastructure. This approach is also known as *KMS-as-a-service* and it is one of the most used solutions due to its flexibility and its migration capabilities.

Table I summarises pros and cons of each solution.

When categorised according to their *Key Distribution Center* (KDC), i.e., the entity responsible for distributing keys, key management systems are usually clustered as *distributed*, *centralized* and *decentralized* [14]. A *centralized KMS* is built around a single central entity that is in charge of managing the keys and distributing them to all the users. In a *distributed KMS* there is no single master entity and each user of the KMS manages her/his own keys and uses contributory key agreement protocols [2] to cooperate and contribute, with all other members of the group, to the creation of a shared key. In a *decentralized KMS* users are split into several smaller subgroups, each managed by an appointed manager who can, in turn, refer or not to a manager of the entire KMS.

With respect to the adopted key storage solution, KMS's are usually defined as *centralised* or *distributed* [10]. In the former case all the keys are stored by the master entity of the KMS that is in charge of providing secure storage for all of them, whereas in the latter one each user is in charge of storing her/his own keys in a secure way and should be provided with all the tools necessary to fulfil this requirement. An example of distributed KMS can be the Apple Secure Enclave Processor (SEP) [13] an isolated component from the main processor that provides secure storage for critical information, finger print, cryptographic keys, etc. but also cryptographic primitives for the main system.

B. NIST Recommendation

US NIST plays a key role in providing guidelines and recommendation for Key Management and KMS development [3], today widely and extensively adopted by the implementers

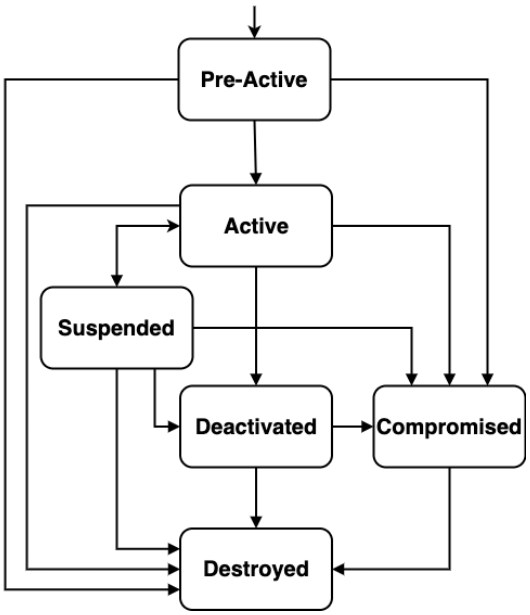

Fig. 1. Key State and Transactions

of KMS's worldwide. In the sequel we briefly recall some of the most significant issues pointed out in the NIST's documents.

Key life cycle: at any given point in time, a key is characterised by a specific *state* [3] that determines how the key can be used. Figure 1 shows the different states and the permitted transitions among states.

- *Pre-activation*: when a key is created it enters this state and it cannot be used until "activated".
- *Active*: in order to be used, a key must be in this state.
- *Suspended*: when in this state, the key cannot be used, but it can be activated again.
- *Deactivated*: the key can be used only to decrypt, but no longer to encrypt. When a key is replaced by a newer one, it is still needed for decrypting data encrypted by it.
- *Compromised*: this is a warning state. It means that the key is, or may be, compromised due to, for example, a data breach; the key can still be used for both encryption and decryption but with particular care. A compromised key cannot be reactivated.
- *Destroyed*: when in this state, the key is completely removed from the system.

Cryptoperiod is the time span during which a specific key can be used. This quantity is extremely important and it is strictly related to the security of a cryptographic key, the more a key is used, the more frequently it must be updated in order to lower the chance for it to be compromised.

Physical and logical access protection this is of paramount importance for the KMS. Access to keys must be protected physically and logically to avoid any disclosure of critical information, unwanted modifications, unauthorised usage or access. For the physical protection, NIST suggests the adoption of custom hardware solutions, such as hardware security modules. Logical protection measures include encryption, authentication, integrity checks, access control, and accountability.

Physical and logical separation of roles for the actors within the KMS. Access to physical assets, such as, key servers, backup servers, etc. must be limited and monitored. Similarly, from a logical perspective the adoption of different privilege levels can be used to limit the access to critical features of the KMS:

- *Separation of Duties*: no user in the system should have enough privileges to be able to misuse the system. Critical functionalities are split among different members to prevent a single user from having enough information or privileges to maliciously damage the whole system.
- *Least Privilege*: each member or actor of the system is given the least amount of access privileges that allows she/he to perform her/his jobs.

All the above guidelines and principles have been strictly followed and adopted during the design and implementation of the SEkey KMS.

IV. SEKEY

In this section we introduce the basic features of SEkey, a KMS that leverages on the features and functionalities provided by the SEcube™ hardware security module. In particular we shall focus on (i) SEkey general architecture, (ii) the concept of *User Groups*, (iii) the different *roles* within the KMS, (iv) how the SEcube™ is profitably employed, (v) the internal structure of SEkey, (vi) the cryptographic keys distribution mechanism, and (vii) the key management feature.

A. SEkey General Architecture

As shown in Figure 2, SEkey manages and distributes cryptographic keys shared among users who are clustered in groups [20], each one being characterised by a custom security policy. The KMS is controlled by an administrator who interacts with the users by means of APIs performing a wide range of actions, such as creating and distributing cryptographic keys, creating and managing users and groups, etc.

A peculiar aspect of SEkey is that each user is forced to make use of a dedicated SEcube™ device, thus implementing a distributed architecture wherein the cryptographic keys are automatically delivered to the users, who securely store them inside their SEcube™ devices. Therefore, the users make use of the KMS together with their SEcube™ devices in order to secure the data they need to exchange or store.

B. User Groups

At the core of SEkey there is the notion of *group* [20], which is the fundamental component used to control the users and the access to the cryptographic keys. Each group consists of an arbitrary number of users and cryptographic keys. Every user of SEkey belongs to a specific set of groups; similarly, each cryptographic key of the KMS is owned by a specific group. A user may belong to several groups, therefore the intersection

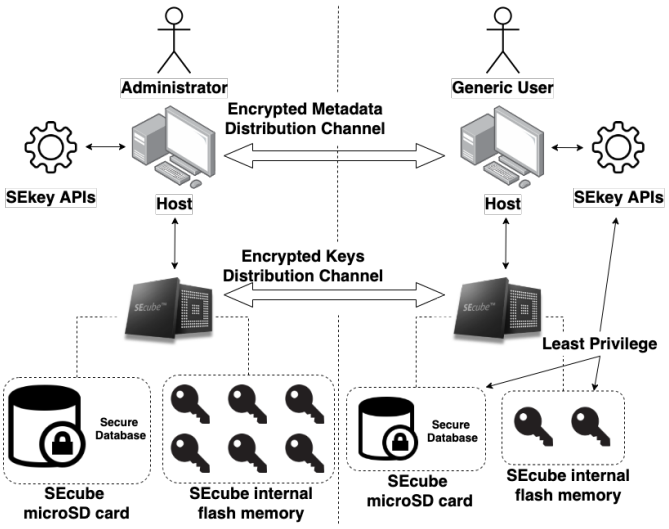

Fig. 2. SEkey General Architecture

of multiple groups may not be empty. On the other hand, the ownership of cryptographic keys is fixed; a key is always owned exclusively by a single group, without any possibility of changing the owner. Notice that the ownership of a key is always referred to a group, never to a single user.

The users gain access only to the cryptographic keys owned by the groups to which they belong; therefore, group members can encrypt shared information using the group cryptographic keys. Two users can share encrypted information only if both belong to at least one group, meaning that they both have access to (at least) one common encryption key. Moreover, each group is associated with a set of *security policies*, detailing specific rules to be followed when managing the security of that specific group. These include, among the others, details about the criptographic algorithms to be adopted, the resource (software, hardware, smart card) to be used for cyphering, the default cryptoperiod of the keys, the schedule of their updating, and so on.

This hierarchy is based on a simple concept: the smaller the group, the higher its security [20]. This idea arises from the assumption that a smaller group involves a reduced number of individuals, therefore the security risks are inherently mitigated because the surface available for a cyber attack is greatly reduced and the sensitive information is shared among a smaller number of people.

C. Roles of the Involved Actors

Actors operating in the SEkey KMS acts either as *administrator* or as *user*. Each role is fixed, meaning that the administrator is not a user and the users cannot act as the administrator.

The *administrator* plays a key role, being the only one having the privilege to modify the configuration of the KMS (i.e., create, distribute, destroy cryptographic keys) and to set up and update groups and users, defining their security perimeters and policies. The SEcube™ device of the administrator contains all the informations managed by the KMS, including all the cryptographic keys; this mainly allows the system to recover from faults that may occurs on the user side. Following the "need-to-know" principle, the administrator shares with the users only the minimum necessary set of information: for example, a user ignores the existence of other people outside of his groups.

Users play a passive role, they can use the KMS but are not allowed to perform any change, neither in the system configuration nor in the involved keys. A user can, in fact, access its own set of cryptographic keys, only; moreover, each key can be used to perform cryptographic operations towards specific recipients only. A user is unable to perform operations which have not been authorized by the administrator (e.g., communicating with users with whom he has got no group in common).

D. SEkey Internal Structure

SEkey, in addition to cryptographic keys, requires to properly manage additional information and metadata which are essential to the system management. To effectively and efficiently tackle this issue, each user of SEkey is given a private instance of the SEcube™ device, which is used to store these critical information items in different locations. In particular, keys are stored in the internal memory of the SEcube™ devices in order to guarantee the highest level of physical protection, whereas the metadata are stored into its MicroSD card. The main reason for this separation is that the size of the internal flash memory of the SEcube™ device is limited to 2 MB, thus it has been reserved to the cryptographic keys.

All the cryptographic primitives are executed by the SEcube™ itself, the user (and administrator as well) only gets the output of those operations, such as encrypted or decrypted data, computed signatures and so on. Moreover, the firmware of the device exposes neither any function to read the content of the internal memory nor the key values in clear, granting a good level of isolation from the main system. In a way similar to the concept of *tokenisations* [11] [12], used in digital payments, where credit card numbers are not sent directly, but instead a mathematically unrelated identifier is shared. Only when the payment has to be processed the unique ID is substituted with the corresponding credit card number; outside of the HSM each key is referred through its own *Unique ID*. It is, hence, impossible to retrieve actual key values because no trace of them can be found anywhere else except the internal memory of the SEcube™ devices.

Since the metadata about keys, users, and groups are stored into a MicroSD card, a different strategy is required to grant a suitable level of security and protection. This alternative strategy relies on SEfile (see Section II): a library of the SEcube™ Open Source SDK that allows to encrypt files and to work with them while keeping everything constantly encrypted on disk. SEfile works together with the open source SQLite[2]

[2]https://www.sqlite.org/index.html

database engine in order to implement a library called 'Secure Database'. In this library, specific for the SEcube™ device, the SQLite database engine has been tweaked to work on a constantly encrypted database while granting confidentiality, integrity and authentication of the DB files thanks to the cryptographic primitives provided by the SEcube™ device.

In addition, SEcube™ is protected by a pair of PIN codes that must be used to access the functionalities provided by the device. Each PIN code is unique to a given SEcube™ and it is associated with a specific privilege level, *admin* and *user*. The PIN codes of each SEcube™ are set during the physical initialization of the device, which takes place before the HSM being physically handed to the user or to the administrator.
The PIN codes of the SEcube™ devices are not related to the actual role of the actors of the KMS. Their only purpose is to stop unauthorised people from accessing the functionalities of the device or limiting the features exposed by the firmware of the SEcube™ to boost the overall security of the system. Following the *Least Privilege* paradigm (see Section III-B), only the minimum amount of information required by each involved actor to perform its operations, is disclosed [15]. For example, each user is provided only with the PIN that grants access to the *user* privilege level of his SEcube™ device while the PIN for the *admin* level is kept secret inside the SEcube™ of the administrator.

E. System Update and Cryptographic Key Distribution

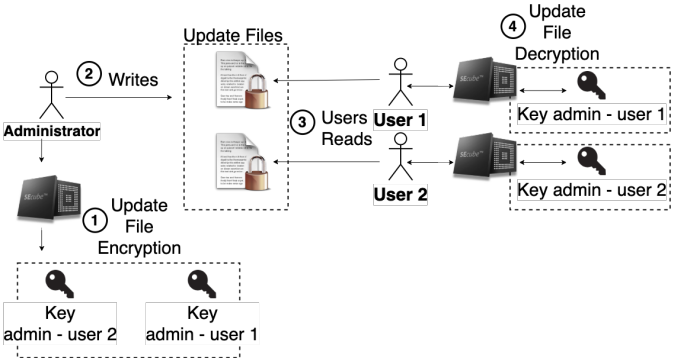

Fig. 3. Update Distribution

Having a distributed architecture where the SEcube™ devices of the users store locally every information that is required for the correct functioning of the KMS, a dedicated secure protocol to share and distribute the data (i.e. groups update, the cryptographic keys and so on) from the administrator to the users is required.

The distribution of the data is always initiated by the administrator, who automatically pushes the data to the users; then the users process these data and store them inside their SEcube™ devices.

This mechanism requires a very simple underlying infrastructure (Figure 3) that relies on update files generated specifically for each user of the system. The update files are encrypted with a key that is known only to the administrator and to the recipient; thus, a secure end-to-end channel is implemented between the host computers of the involved parties. Whenever a new update file is generated by the administrator of SEkey, it is written to a non-volatile memory support that must be accessible also to the users. This non-volatile memory could be anything ranging from a shared disk in a LAN to a cloud service, the only requirement being that all parties involved in the KMS must be able to access to it.
SEkey is configured to automatically generate the update files from the administrator side, and to automatically process them from the user side. The update files contain every data that a given user is entitled to store into her/his personal SEcube™. When SEkey needs to share a cryptographic key from the SEcube™ of the administrator to the SEcube™ of a user, that key must be exported from the HSM of the administrator and written to the update file of the user. The encrypted channel implemented by the update file is not sufficient to protect the key because its value would still be visible to the administrator (the plaintext content of the update file is initially built in the host computer of the administrator, then it is encrypted by the SEcube™ and finally written to the update file). In order to solve this problem, SEkey implements an additional encrypted end-to-end channel, created inside the update file. This channel is built directly between the SEcube™ devices of the involved parties (administrator and user), it allows to export a key from the SEcube™ of the administrator only if that key is already wrapped with another key (which is unique for each user). In this way, the key is already exported outside of the SEcube™ in an encrypted format, guaranteeing that even the administrator cannot see its real value. When the SEcube™ of a user receives a wrapped key, it removes the wrapping and stores the key inside its flash memory, never exposing the real value of the key outside of the HSM.
From a physical point of view, the generation of the cryptographic keys managed by the KMS is always performed inside the SEcube™ of the administrator using a True Random Number Generator embedded in the SEcube™ MCU [16], guaranteeing that each key is random and secure.

F. Key Management Features

The ultimate goal of a KMS is to manage the life cycle of the cryptographic keys. In this sense, each key is characterised by several properties, the most important being the *cryptoperiod* and the *state* (see Section III-B).

The cryptoperiod states how long a key can be used to encrypt data. It can be set, by default, to the value specified by the security policy of the groups that owns the key. However, it could be set to a smaller value when needed; values higher than the default one are not allowed.

The state, instead, determines the current condition of the key. For example, a key can be used to apply cryptographic protection (encrypt data) only if it is in the *active* state; on the other hand, it can be used to decrypt data also if it is not active. Some states, such as *destroyed* and *compromised*, always prevent SEkey from using a key due to security reasons.

Depending on its cryptoperiod and on its state, a key may be eligible for usage. SEkey automatically manages a portion of the life cycle of each cryptographic key, for example it deactivates the keys whose cryptoperiod is expired and it has built-in protection mechanisms to prevent the usage of keys depending on their current state.

When an application needs to perform an encryption operation, it can simply call an API of the KMS that returns the unique identifier of the most secure key to be used, then that identifier is passed to the encryption APIs of the SEcube™. The most secure key to be used in a given situation is determined by the list of the recipients of the data to be encrypted. Here comes into play the concept of *group* (see Section IV-B) so if a user needs to encrypt a message that must be sent to another user, SEkey will automatically search a usable key belonging to the smallest group in common between all the parties involved in the communication, because a smaller group is considered to be safer. The same holds if a user wants to encrypt data for private usage, for example before storing them on a cloud server. In that case the user will specify himself as the only recipient, so SEkey will search for a usable key belonging to a group where that user is the only member.

In addition to the keys managed by the KMS, additional cryptographic keys are required to properly manage the system. These keys are not under the direct control of the KMS or the administrator, but are generated automatically by the system. are not visible to the user, and are used to encrypt data locally to each SEcube™. For example, every SEcube™ generates a unique key that is used to encrypt the metadata database of SEkey.

V. Conclusions

In this paper, SEkey was presented, a key management system that leverages the peculiar features and functionalities of the SEcube™ hardware security module to provide all what is required to securely manage cryptographic keys.

During the design of SEkey, all the most important security dictates provided in the NIST guidelines were followed. Each key is associated with a cryptoperiod and a state. Seven different states are used to determine the type of operations that a key can perform. Moreover, following the 'Least Privilege' principle two actors with different privileges have been identified in the KMS, the *administrator* and the *user*: the former having the full privilege to perform any modification to the KMS data while the latter can just use the KMS passively without any authority to make changes.

The SEkey KMS is based on a distributed structure and its users are organised according to a particular hierarchy that provides multiple groups, each characterised by specific security policies. Users can communicate and share information with each other by means of symmetric cryptographic keys shared within the group. Each actor in the KMS has its own SEcube™ HSM and all the cryptographic keys and critical items are stored securely in the internal device flash memory. Moreover, all the cryptographic primitives are provided by the SEcube™ itself, hence keys never leave the device when performing crypto operations and are never exposed in clear. The keys that are distributed by the administrator are over-encrypted with a unique key shared only between the administrator and the user who must receive them. To limit the use of the device internal memory, all the metadata handled by the KMS are saved, on a MicroSD card connected to the SEcube™, in an always encrypted database, thus guaranteeing the integrity, confidentiality and authenticity of these data. Since the internal memory of SEcube™ is limited to 2MB available, the adopted approach allows storing inside a single flash memory sector (128KB) up to 4096 different keys (assuming a key size of 256 bits).

As far as future improvements there are the following aspects are going to be tackled in the near future:

- Management of session keys: keys that can be generated, used and dismissed within a group when there is the need of instantiating a communication channel. In this way it is possible to better separate keys that can be used to cryptographically secure data at rest (e.g., files) and data in motion (e.g., calls). Groups can internally manage the creation of these type of keys, using for example a contributory key agreement protocol, without querying the central manager.
- Improvement in the internal flash memory management of the device: since flash memories have a limited amount of write operations that can be performed, having to replace every now and then keys inside it can quickly wear out memory.
- Implementation of a PUF inside the device: this can be used either as a strong private cryptographic key, used for example for the metadata database encryption, or as a unique key shared by the administrator and each user used for the encryption of SEkey update messages.
- Addressing the problem of *non-repudiation* in group encryption. Methodologies exists involving either asymmetric encryption, such as *Ring Signature* [21] or *Threshold Signature* [1], or symmetric encryption such as the use of trusted third party top provide a One Time Password to be used in the signing process [17].

VI. Acknowledgments

The activities presented in the present paper are partially supported by the *European Union's Horizon 2020 research and innovation programme*, under grant agreement No. 830892, project SPARTA and by *B5 Labs Ltd.*.

References

[1] Michel Abdalla, Sara Miner, and Chanathip Namprempre. Forward-secure threshold signature schemes. In *Cryptographers' Track at the RSA Conference*, pages 441–456. Springer, 2001.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, J. L. Schultz, J. Stanton, and G. Tsudik. Secure group communication using robust contributory key agreement. *IEEE Transactions on Parallel and Distributed Systems*, 15(5):468–480, 2004.

[3] E. Barker. Recommendation for key management: Part 1 - general. *NIST, Tech. Rep*, 2020.

[4] W. C. Barker. Guideline for identifying an information system as a national security system. *NIST, Tech. Rep*, 2003.

[5] M. Bollo, A. Carelli, S. Di Carlo, and P. Prinetto. Side-channel analysis of secube™ platform. In *2017 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–5, 2017.

[6] CRYPTOMAThIC. Selecting The Right Key Management System. https://www.cryptomathic.com/hubfs/Documents/White_Papers/Cryptomathic_White_Paper_-_Selecting_The_Right_Key_Management_System.pdf, 2019. [Online; accessed 22-July-2020].

[7] K. Dempsey, M. Nieles, and V. Y. Pillitteri. An introduction to information security. *NIST, Tech. Rep*, 2017.

[8] M. Fornero, N. Maunero, P. Prinetto, G. Roascio, and A. Varriale. SEcube Open Security Platform - Introduction. https://www.secube.eu/site/assets/files/1218/wiki.pdf, 2019. [Online; accessed 22-July-2020].

[9] M. Fornero, N. Maunero, P. Prinetto, G. Roascio, and A. Varriale. SEfile Documentation. https://www.secube.eu/site/assets/files/1218/wiki.pdf, 2020. [Online; accessed 22-July-2020].

[10] V. Gopal, S. Fadnavis, and J. Coffman. Low-cost distributed key management. In *2018 IEEE World Congress on Services (SERVICES)*, pages 57–58, 2018.

[11] Gabriel Babatunde Iwasokun, Taiwo Gabriel Omomule, and Raphael Olufemi Akinyede. Encryption and tokenization-based system for credit card information security. *International Journal of Cyber Security and Digital Forensics*, 7(3):283–293, 2018.

[12] Ronald Julien Jr. *The cybersecurity aspects of Apple Pay*. PhD thesis, Utica College, 2016.

[13] T. Mandt, M. Solnik, and D. Wang. Demystifying the secure enclave processor. *Black Hat Las Vegas*, 2016.

[14] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys (CSUR)*, 35(3):309–329, 2003.

[15] F. B. Schneider. Least privilege and more [computer security]. *IEEE Security Privacy*, 1(5):55–59, 2003.

[16] STMicroelectronics. AN4230 Application Note - STM32 microcontroller random number generation validation using the NIST statistical test suite. https://www.st.com/resource/en/application_note/dm00073853-stm32-microcontroller-random-number-generation-validation-using-the-nist-statistical-test-suite-stmicroelectronics.pdf, 2020. [Online; accessed 22-July-2020].

[17] International Telecommunication Union. X.1156: Non-repudiation framework based on a one-time password. https://www.itu.int/rec/T-REC-X.1156-201306-I/en, 2014. [Online; accessed 22-July-2020].

[18] A. Varriale, E. I. Vatajelu, G. Di Natale, P. Prinetto, P. Trotta, and T. Margaria. Secube™: An open-source security platform in a single soc. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, 2016.

[19] Antonio Varriale, Giorgio Di Natale, Paolo Prinetto, Bernhard Steffen, and Tiziana Margaria. Secube (tm): an open security platform-general approach and strategies. In *Proceedings of the International Conference on Security and Management (SAM)*, page 131. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[20] Antonio Varriale, Paolo Prinetto, Alberto Carelli, and Pascal Trotta. Secube (tm): Data at rest and data in motion protection. In *Proceedings of the International Conference on Security and Management (SAM)*, page 138. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[21] Fangguo Zhang and Kwangjo Kim. Id-based blind signature and ring signature from pairings. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–547. Springer, 2002.