Published version: 10.1109/TrustCom60117.2023.00224
2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)

A New Design for Self-Encryption

1st Roland Kromes
Delft University of Technology
Netherlands
r.g.kromes@tudelft.nl

2nd João Rodrigues
INOV - Instituto de Engenharia de Sistemas e Computadores Inovação
Portugal
joao.rodrigues@inov.pt

3rd Duarte Nascimento
INOV - Instituto de Engenharia de Sistemas e Computadores Inovação
Portugal
duarte.nascimento@inov.pt

4th Gonçalo Cadete
INOV - Instituto de Engenharia de Sistemas e Computadores Inovação
Portugal
goncalo.cadete@inov.pt

5th François Verdier
Université Côte d'Azur, LEAT
France
francois.verdier@univ-cotedazur.fr

6th Kaitai Liang
Delft University of Technology
Netherlands
Kaitai.Liang@tudelft.nl

***Abstract*—Nowadays, Internet of Things applications face serious data and privacy protection vulnerabilities. To address some of the data protection and privacy issues, in this work we propose a new design for the self-encryption method based on a cryptographic-puzzle algorithm, that includes the generation of multiple secret keys, derived from the plaintext. As the ciphertext is constructed from several chunks of encrypted data, the absence of one of the decryption keys or one of the encrypted chunks renders recovery of the original plaintext nearly impossible. As security improvement upon to other related work proposing self-encryption, the plaintext is mixed with random values in order to use a technique known as Privacy Amplification. Privacy Amplification is achieved by applying cryptographic functions from which SHA-2 family is based on. Implementations of our design are also provided, and they are enabled for standalone and back-end execution systems. Furthermore, performance and security results and comparisons with previous related work are also provided. The security analysis confirms the use of the SHA-2 cryptographic hash family for Privacy Amplification.**

***Index Terms*—Privacy, Security, Encryption, IoT, Privacy Amplification**

I. Introduction

Nowadays, the Internet of Things (IoT) is considered a complex network where all kinds of devices are connected, communicating with each other. IoT is used in smart cities [1], supply-chain management [2] [3], smart grid [4], smart vehicle systems [5], and smart healthcare application fields [6]. With the number of connected devices increasing exponentially, and the vast amount of data produced by IoT devices, IoT data plays a major role as a drive for the big data paradigm future [7]. Many believe, that several IoT use cases can have a positive effect on human life, such as in healthcare related applications. To mention one of the many, patients can use a wireless body area network (WBAN) [8] to record and transmit their health data to a medical doctor, facilitating the prevention and cure of illnesses and diseases. On the other hand, IoT faces serious security and privacy issues [1] [9]. Person-related data is privacy-sensitive, thus protecting this information is essential and requires an encryption method that provides a high security level, correct usability and reasonable efficiency. Moreover, the method should also provide mechanisms to enforce a correct ownership of the data, implying how the shared data can be accessed [10] [11].

The application of a puzzle-based cryptographic algorithm is a promising approach for data encryption, as a missing encrypted chunk or decryption key makes it nearly impossible to recover the plaintext. Such algorithm results in a set of encrypted chunks and the corresponding decryption keys, and provides efficient operation at high security levels. As this method allows the storage of the encrypted chunks in different locations, it makes difficult the access to the fully encrypted data by unauthorized parties. MaidSafe.net proposed the self-encryption algorithm [12] as a data protection solution that applies puzzle-based encryption. As the original idea of this solution is promising, we elaborate its features, and propose a new design to improve its security, usability and performance. Furthermore, we apply Privacy Amplification for the key generation, which can be achieved by employing special types of hash functions [13]–[15]. It is important to note that Privacy Amplification carries an important result called the Leftover Hash Lemma. Currently, there are many versions of the Leftover Hash Lemma [15] and present results on Privacy Amplification [16]. Our proposed self-encryption method does not enforce access policies, but it can be used as an initial encryption mechanism by applications that enforce their own data sharing and access policies on the encrypted data. Our contributions are the following:

- Contrarily to the existing self-encryption implementations, ours supports two encryption modes: the OTP-like mode allows the encryption of data with OTP-like (*One-Time-Pad*) security level. The default mode provides a security of 2^r, where r is equal to 512 bits;
- We explore some of the existing security results regarding encryption algorithm architectures, namely the base architecture of SHA-2 and AES, in order to understand what assumptions we must fulfil in order to guarantee security in the theoretical sense. Based on the analysis,

we propose a new self-encryption method. The approach also implies the justification for using the SHA-2 cryptographic hash family for Privacy Amplification, based on related work [13];

- We provide a Golang implementation of our proposed self-encryption method, which can be run efficiently in a PC, back-end and edge environments. Our work is a completely open-source and reproducible project and all source codes are available on our GitHub repository[1].

This article is structured as follows: Sect. II describes the related works regarding self-encryption and its application. Sect. III showcases our proposed self-encryption method. Sect. IV provides security related knowledge regarding hash functions and AES-based architectures applied in our proposed encryption method. The experiments and results of our implementation are presented in Sect. V. In Sect. VI, an evaluation of the proposed self-encryption method is given. Finally, Sect. VII provides the conclusions of our work.

II. Related Work

The basic goal of the original self-encryption was to create a strong ciphertext without user intervention or passwords. Its original authors [12] highlight that the self-encryption algorithm is not a new cipher scheme but the combination of the AES-CBC (128-bit) cipher and a cryptographic hash algorithm with obfuscation operations. The author also claims that their implementation should be considered as an OTP encryption. This algorithm produces encrypted chunks based on chunked plaintext, in a deterministic way, thus providing deduplication of encrypted data. This may be, however, undesirable in use cases where the plaintext follows patterns that might be exposed by multiple colliding encrypted chunks. For more details on the original self-encryption, we refer to the work [12].

Grishkov *et al.* [17] applied the self-encryption in a blockchain [18] context for secure data sharing. The authors called their implementation ID-based self-encryption, and pointed out that storing the encrypted data chunks in a distributed way is beneficial as each chunk can be stored in a different location belonging to different network participants. In that same work, a Proof-of-Concept architecture was provided and was based on an IPFS network [19], where each encrypted chunk was stored in a different network node. IPFS uses the hash values of the data as references to store and locate data. Since the self-encryption provides the hash values of the encrypted data chunks, in the work proposed by Grishkov *et al.* [17] the hash values were used directly as references to the encrypted data chunks in the IPFS network.

To improve the original self-encryption, Grishkov *et al.* [17] proposed the addition of a self-signed X.509 certificate to the data to be encrypted. This self-signed certificate allows the authentication and identification of the encryptor or the data owner. Our proposed self-encryption method inherits this self-signed identity feature, as it can be useful when a proof is required regarding the identity of the issuer of the encryption or of the owner of the data. Furthermore, the self-signed certificate can also help to hide the real identity of the data owner, which is a benefit in the light of some possible privacy requirements.

ID-based self-encryption facilitates distributed data storage and the authentication of encrypted data. However, in both the original and the ID-based self-encryption, the keys are generated in a deterministic way, by using the hash of the data chunks of the plaintext. As the plaintext may not provide the necessary randomness, the first stage of the encryption of the data provided by the AES blocks is not provably secure. The authors of the original self-encryption [12] claimed that their implementation should be considered as an OTP. However, the keys used for the *One-Time-Padding* operation were not random, which is an essential requirement of the perfect secrecy provided by an OTP. In our implementation, the aforementioned issues are addressed to obtain a provably secure self-encryption method, and providing at least a security of 2^r, where r is 512 bits.

III. Proposed Self-Encryption Method

This section describes our improved implementation of the self-encryption algorithm, and highlights the differences in relation to the original, as it was presented in [12]. Figure 1 shows the core components of our proposed self-encryption method. The data flows of our implementation are as follows:

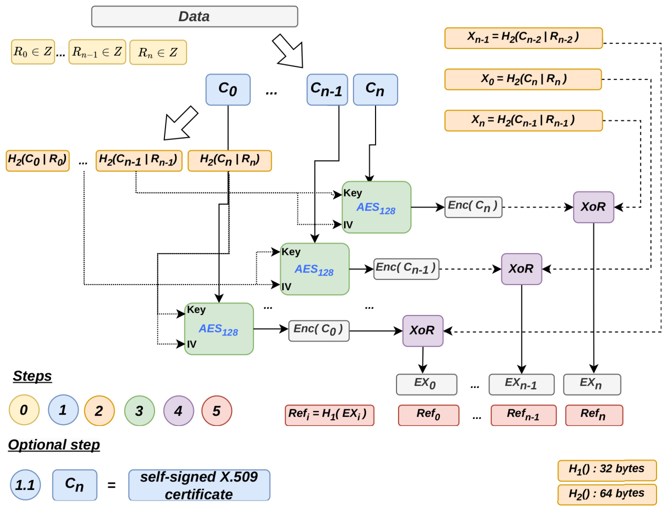
Data
$R_0 \in Z$ ... $R_{n-1} \in Z$ $R_n \in Z$
C_0 ... C_{n-1} C_n
$X_{n-1} = H_2(C_{n-2} | R_{n-2})$
$X_0 = H_2(C_n | R_n)$
$X_n = H_2(C_{n-1} | R_{n-1})$
$H_2(C_0 | R_0)$ ... $H_2(C_{n-1} | R_{n-1})$ $H_2(C_n | R_n)$
Key
AES_{128}
IV
$Enc(C_n)$
$Enc(C_{n-1})$
$Enc(C_0)$
XoR
EX_0 ... EX_{n-1} EX_n
$Ref_i = H_1(EX_i)$
Ref_0 ... Ref_{n-1} Ref_n
Steps
0 1 2 3 4 5
Optional step
1.1 C_n = self-signed X.509 certificate
$H_1()$: 32 bytes
$H_2()$: 64 bytes

Fig. 1. Our improved self-encryption method

(0) *Random number generation:* for each data chunk that is generated in step (1), a random number with a length of 128 bytes (i.e., 1024 bits) is generated (i.e., $R_i \in \mathbb{Z}$)

(1) *Division of the initial data into chunks:*
Default mode: the initial data (plaintext) is divided into n chunks of quasi-equal length $\{C_0, \ldots, C_{n-1}\}$. At least three data chunks must be generated.
OTP-like mode: the initial data is divided into n chunks

[1]Our proposed self-encryption is avaulable at GitHub https://github.com/KRolander/New-Design-for-Self-Encryption

and the length of an individual chunk must be less than or equal to 48 bytes. The limitation of the chunk length guarantees that the AES encrypted chunks will not exceed the 64 bytes length. That property is particularly useful in step ③, as it allows for the construction of an OTP-like encryption. More insights about the importance of this mode are provided after the description of the steps.

(1.1) *Addition of a self-signed X.509 certificate:* this functionality allows the issuer of the encryption to add a self-signed certificate as the last data chunk, hence the n-th chunk will contain the self-signed certificate. The certificate allows the authentication of the issuer of the encryption, or of the data owner. This step is optional, but proves useful when additional authentication is required.

② *Generation of secret keys and obfuscation values:* Each chunk (C_i) is concatenated with a random value of 128 bytes (R_i) generated in step ⓪. The concatenated value is then hashed with a cryptographic hash function ($H_2(C_i|R_i)$), where H_2 provides a hash digest of 64 bytes. These hashes are used as secret keys for the AES cipher. Moreover, the obfuscation values ($\{X_0, \dots X_{n-1}, X_n\}$) are also composed by these hash values. The i-th obfuscation value (X_i) is the hash value of the concatenation of the $i-1$-th chunk and random number, e.i., $X_i = H_2(C_{i-1 \mod n}|R_{i-1 \mod n})$.

③ *AES-128 encryption of the chunks:* The AES-CBC (128 bit – 16 bytes) is a block cipher that requires a key (Key) and initial value (IV) of 16-16 bytes. The cipher of the i-th chunk ($Enc(C_i)$) is realized by using the hash value of the $i-1$-th chunk concatenated with the $i-1$-th random value ($H_2(C_{i-1 \mod n}|R_{i-1 \mod n})$), as input of the AES cipher. The first 16 bytes of this hash correspond to the Key, and the last 16 bytes are equal to the IV of the AES cipher e.i., $Key = H_2(C_{i-1 \mod n}|R_{i-1 \mod n})_{[0-15]}$, $IV = H_2(C_{i-1 \mod n}|R_{i-1 \mod n})_{[16-31]}$.

④ *Obfuscation of the encrypted chunks with the obfuscation values:* The obfuscation values determined in ② are logic XOR-ed with the AES encrypted chunks created in ③ ($EX_i = X_i \oplus Enc(C_i)$). Note that the XOR function mentioned above is a circular implementation because the length of the AES encrypted chunks may differ from the obfuscation values (64 bytes).

⑤ *Generation of references:* The final encrypted chunks are hashed (Ref_i= $H_1(EX_i)$).

Necessary randomness. It must be noted, that the phases: ⓪, ①, (1.1) and ② of our self-encryption method are significantly different from the implementation proposed of the original self-encryption [20] as we introduce a necessary randomness while generating the keys for the AES cipher blocks. In [12], the author claims that the AES encrypted chunks XOR-ed with the obfuscation values (X_i) should be considered as an OTP. However, in that same work, the encrypted chunks and keys needed for the obfuscation values are generated deterministically, and one of the main requirements of the perfect secrecy provided by the OTP is that the keys must be uniformly distributed random numbers. Also, the self-encryption proposed in [17] fully inherits the key generation of the original self-encryption; thus, our method significantly differs from the implementation proposed by [17] as well.

OTP-likeness. In our implementation, the AES encrypted chunks and the obfuscation values contain randomness thanks to steps ⓪ and ②. Furthermore, in step ④ we can achieve a security level close to the perfect secrecy if the lengths of the chunks are 48 byte-long, resulting in 64 byte-long AES encrypted chunks used as input of the obfuscation. When applying 64 byte-long AES encrypted chunks, the obfuscation values (X_i) do not have to be circular, which implies that the AES encrypted chunks are XOR-ed with a same length value. Our implementation outputs AES encrypted chunks with a length of 64 bytes, when the the **OTP-like mode** of step ② is applied. Hence, with this **OTP-like mode**, we obtain an OTP-like encryption.

Default encryption mode. In step ④, we mentioned that if the encrypted chunk do not have a length of 64 bytes, the obfuscation value is circulated to achieve the length of the encrypted chunk. The circulated obfuscation value implies that this encryption mode offers a maximum security of 2^r with $r = 512$.

Decryption process. After obtaining all of the secret keys and encrypted chunks, the next step is to generate the obfuscation values (X_i). The encrypted chunks can then be XOR-ed with the obfuscation values to reach the AES encrypted chunks ($Enc(C_i)$). Next, the AES encrypted chunks must be decrypted to obtain the decrypted data chunks, which are finally concatenated to obtain the original data.

Technical specification. Our default implementation is written in Golang, which is widely compatible, as Golang functions can easily be called from other programming languages (e.g., C, Python, Ruby, Node, and Java). The implementation includes the standard Golang *crypto* library[2] to realize the AES-CBC block cipher, the SHA-256 and SHA-512 hash functions, random number generation, and the ECDSA digital signature with P-256 NIST curve.

IV. AES and SHA-2 Design Security

This section describes some known results and usages of hash functions and the symmetric key encryption with AES-based architecture.

A. Privacy Amplification

Consider the following scenario: Alice and Bob have full access to a random variable Z, and Eve knows some information regarding the variable Z. The challenge is for Alice and Bob to "extract" randomness from the information of Z

[2]Golang Standard Crypto Library is available at https://pkg.go.dev/crypto@go1.20.4

that is unknown to Eve in order to yield a symmetric key for secure communication [14], [15].

Bennet, Brassat and Roberts proved that, if Alice and Bob agree on a hash function g selected randomly from a family of Universal Hash Functions, Alice and Bob can apply g on Z and yield a completely random sequence of bits from Eve's perspective [14], [15].

A variant of Universal Hash families, the one we will be using in this work, is called δ-Almost Universal, and it was first proposed by Stinson [13], [21].

Definition 1 (Almost Universal Hash Functions): Let $\mathcal{G}$ be a class of functions $g : A \to B$, G be a random variable over $\mathcal{G}$, and Z_1 and Z_2 be uniform random variables over set A. Then G is δ-AU if for all $z_1 \in Z_1$ and $z_2 \in Z_2$, $Prob[G(z_1) = G(z_2)|z_1 \neq z_2] \leq \delta$.

Central to the Privacy Amplification security analysis of hash functions is the collision probability, Shannon entropy and mutual information, which we define below.

Definition 2 (Collision Probability): Let Z be a discrete random variable on the alphabet $\mathcal{Z}$. The collision probability is defined to be

$$P_C(Z) = \sum_{z \in \mathcal{Z}} P(z)^2 = E[P(Z)]. \tag{1}$$

Definition 3 (Shannon Entropy): Let Z be a discrete random variable on the alphabet $\mathcal{Z}$. The Shannon entropy of Z is given by

$$\mathcal{H}(Z) = \sum_{z \in Z} -P(z)\log(P(z)). \tag{2}$$

Definition 4 (Mutual Information): Let Y and Z be two discrete random variables on the alphabets $\mathcal{Y}$ and $\mathcal{Z}$ respectively. The mutual information between random variables Y and Z is given by

$$\mathcal{I}(Y;Z) = \mathcal{H}(Y) + \mathcal{H}(Z) - \mathcal{H}(Y,Z). \tag{3}$$

Following closely the proofs given in [14], but using the definition of δ-Almost Universal Hash family, we can state the following result:

Theorem 1 (Privacy Amplification): Let Z be a random n-bit string with uniform distribution over $\{0,1\}^n$, let $V = e(Z)$, for an arbitrary eavesdropping function $e : \{0,1\}^n \to \{0,1\}^t$ for some $t < n$. If Alice and Bob choose $K = G(Z)$ as their secret key, where G is chosen at random from a δ-Almost Universal class of hash functions from $\{0,1\}^n$ to $\{0,1\}^r$, then Eve's expected information about the secret key K, given G and V, satisfies

$$\mathcal{I}(K;G,V) \leq \mathcal{H}(K) + log_2\delta + \frac{2^{-log_2\delta+t-n}}{ln(2)}.$$

This result dictates how much information is available to Eve when she knows some information $e(Z)$ of Z and the selected hash function g.

If we set $\delta = 2^{-r} + \epsilon$ we get

$$\mathcal{I}(K;G,V) \leq \mathcal{H}(K) - r + log_2(1+2^r\epsilon) + \frac{2^{r-log_2(1+2^r\epsilon))+t-n}}{ln(2)}.$$

Setting $\epsilon = 0$, and assuming that the generated key is completely random, we have $\mathcal{H}(K) = r$, and we recover the same result as in [14]. This corresponds to the special case of Universal Hash Function family.

B. Leftover Hash Lemma

Privacy Amplification attains to extract an almost uniformly distributed secret key from a secret key partially known to Eve. In this sense, we can talk about the closeness of the random key generated recurring to the Universal Hash Function family to a uniformly distributed random variable. The closeness of two random variables can be formally defined as follows:

Definition 5 (Statistical Distance): Let Z_1, Z_2 be two random variables over the set $\mathcal{Z}$. The statistical distance between the distributions Z_1 and Z_2 is defined as

$$SD(Z_1, Z_2) = \frac{1}{2}\sum_{z \in \mathcal{Z}} |P(Z_1 = z) - P(Z_2 = z)|. \tag{4}$$

When Z_2 has uniform distribution U, the statistical distance can be upper bounded by [13]:

$$SD(Z_1, U) \leq \frac{1}{2}\sqrt{|\mathcal{Z}| \cdot P_C(Z_1) - 1}.$$

The Leftover Hash Lemma, which we state below, was proved for the class of Almost Universal Hash Functions in [13].

Lemma 1 (Leftover hash Lemma): Let Z be a random variable over the set $\{0,1\}^l$ and let $\mathcal{H}$ be a class of $\left(\frac{1}{2^r} + \epsilon\right)$-Almost Universal Hash Functions with respect to Z, where $h_k : \{0,1\}^l \to \{0,1\}^r$. Let U be a uniform distribution over the set $\{0,1\}^r$. Then

$$SD(k, h_k(Z), (k, U)) \leq \frac{1}{2} \cdot \sqrt{2^r \cdot (P_C(Z) + \epsilon)}. \tag{5}$$

Given that we have the Leftover Hash Lemma and the Privacy Amplification Theorem adapted to the Almost Universal Hash family, what remains to be shown is that the idealized design of the cryptographic hash family SHA-2 can, indeed be used as an Almost Universal Hash family, and under which conditions.

C. Merkle-Damgard Hash functions as Almost Universal Hash Family

The SHA-1 and SHA-2 family of cryptographic hash functions were designed based on the Merkle-Damgard construction, also known as the "*cascade construction*". This construction is described in [13] as follows. Let $\{f_k | f_k : \{0,1\}^b \to \{0,1\}^r, k \in \{0,1\}^r\}$ be a family of functions, called the "compression functions"; Let the input z be divided into L chunks, $z = (z_1, \ldots, z_L)$, where each chunk has b bits. Define the following $L+1$ variables:

- $\bar{z}_0 = k$;
- $\bar{z}_{i+1} = f_{\bar{z}_{i+1}}(z_{i+1})$ for $0 < i \leq L$;
- $F_k(z) = \bar{z}_L$.

The family of functions $F = \{F_k : k \in \{0,1\}^r\}$ is referred to as the "cascade construction".

In [13], by modelling the underlying family of "compression functions" as a family of random functions, the authors were able to prove in Lemma 2 that the family of functions F is Almost Universal, under certain conditions. Before presenting the lemma, we define the min-entropy to be the following.

Definition 6 (min-entropy): Let Z be a discrete random variable on the alphabet $\mathcal{Z}$. The min-entropy of Z is given by

$$\mathcal{H}_\infty(Z) = -log_2 \max(P(Z = z)). \tag{6}$$

Lemma 2: Let $F = \{F_k\}$ be the "cascade construction" defined over a family of random functions $\{f_k\}$. Let Z be an input distribution to F defined over L-block strings, and Z_L denote the probability distribution induced by Z on the last block Z_L. Let $\mathcal{H}_\infty(Z) > log_2(L)$. Then, the family F is $\left(\frac{1}{2^r} + \frac{L}{2^r 2^{\mathcal{H}_\infty(Z_L)}} + O(\epsilon(L, 2^r))\right) - AU$ with respect to Z. The term $\epsilon(L, K) = (d(L))^2 L K^{-2} + L^6 K^{-3}$, and $d(L)$ denotes the maximal numbers of divisors of any number smaller or equal to L. Moreover, if $L < K^{1/4}$, $\epsilon(L, K) = O(L^2/K^2)$.

Additionally, recurring to the Leftover Hash Lemma, the authors in [13] were able to prove the following result:

Theorem 2: Let $F = \{F_k\}$ be the "cascade construction" described above. Let Z be a random variable of rL bits, divided in L blocks of r bits and Z_L be the random variable induced by Z on the final block. Let U be the uniform distribution over $\{0, 1\}^r$. Then

$$SD(F(Z), U) \leq$$
$$\sqrt{2^r \cdot 2^{-\mathcal{H}_\infty(Z)} + L \cdot 2^{-\mathcal{H}_\infty(Z_L)} + O(2^r \cdot \epsilon(L, 2^r))}.$$

In particular, if $\mathcal{H}_\infty(Z) \geq 2r$, $\mathcal{H}_\infty(Z_L) \geq r$, and $L \leq 2^{r/4}$, then $SD(F(Z), U) \leq O\left(\frac{L}{2^{r/2}}\right)$.

D. Key-Alternating Ciphers

In [22], the authors explore theoretically the soundness of the design of existing block ciphers based on the Key-Alternating Ciphers design. They were the first ones to show the design soundness of Key-Alternating Ciphers from the perspective of indifferentiability. A block cipher is a function that has a key k-bit key and an n bit plaintext as inputs, and it outputs a ciphertext of length n.

Key-Alternating Ciphers are defined as a set of key addition and permutation rounds [22]:

$$KA_t(K, m) = k_t \oplus P_t(\dots k_2 \oplus P_2(k_1 \oplus P_1(k_0 \oplus m)) \dots),$$

where $k_0, \dots, k_t$ are determined recurring to a key derivation function applied to some master key K, and t is the number of addition permutation iterations. The authors in [22] were able to show, recurring to the indifferentiability framework, that a KA_t is indifferentiable from the ideal cipher, under certain conditions.

We state the simplified version of the indifferentiability theorem. For more detailed information, please refer to [22].

Theorem 3: The 5-round Key-Alternating Cipher KA_5 is indifferentiable from an ideal cipher, assuming $P_1, \dots, P_5$ are five independent random permutations, and the key derivation function f sets all round keys $k_i = f(K)$, where $0 \leq i \leq 5$ and f is modeled as a k-to-n-bits random oracle.

The AES encryption algorithm can be viewed as a 10-round Key-Alternating Cipher [22], [23].

Other security properties/proofs and frameworks were applied to study the Key-Alternating Ciphers. See, for instance, [22], [23] and the references therein.

E. Implications to the Self-Encryption algorithm

The original self encryption algorithm generates the obfuscation bits X_i recurring to the original data chunks $C_{i-1 \mod n}$, $C_{i-2 \mod n}$, $X_i = H(C_{i-1 \mod n})|H(C_{i-2 \mod n})$.

From the results presented in the Privacy Amplification subsection, the upper bound of Eve's expected knowledge increases with t. This means that, if the plaintext C is drawn from a predetermined context, for example, Eve's knowledge could potentially exceed $n - r$, making it non-negligible. Defining $t - (n - r) = q \geq 1$, Eve's knowledge about the key is no longer negligible (lesser than 1):

$$\mathcal{I}(K; G, V) \leq log_2(1 + 2^r \epsilon) + \frac{2^{q + log_2(1 + 2^r \epsilon)}}{ln(2)}.$$

Moreover, for the same reasons mentioned in the previous paragraph, the min-entropies of the C_i's ($\mathcal{H}_\infty(C_i)$ and $\mathcal{H}_\infty(C_i)_L$) could also be very low, making the inequality of theorem 2 non-negligible.

Considering that the generated hashes $H(C_i)$ are guessable, we are not in the condition to infer that the keys generated recurring to those hashes can be considered randomly generated. As such, in a practical sense, we cannot indicate that the key derivation function (of theorem 3) is a random $k - to - n$-bits random oracle.

If, for instance, we add $2r$ uniformly distributed bits, denoted R_i, we will get

$$\begin{aligned} \mathcal{H}_\infty(C_i|R_i) &= -log_2 \max_{C_i, R_i}(P(C_i)P(R_i)) \\ &= -log_2 \max_{R_i} P(R_i) - log_2 \max_{C_i}(P(C_i)) \end{aligned} \tag{7}$$

Since R_i is uniform, $P(R_i) = 2^{-2r}$. Also, $P(C_1) \leq 1$, hence

$$\begin{aligned} \mathcal{H}_\infty(C_i|R_i) &= 2r - log_2 \max_{C_i}(P(C_i)) \\ &\geq 2r. \end{aligned} \tag{8}$$

Moreover, the L'th (last) block of the sequence $S_L = (C_i|R_i)$, will be completely random as well, hence

$$\mathcal{H}_\infty(S_L) = r \tag{9}$$

Because of these considerations, we chose to include in our self-encryption algorithm proposal, $2r$ random bits into the key generation process. We note, however, that by doing so, we loose the deduplication property of the original algorithm. Moreover, by controlling the size of the data chunks, we can control the number L of blocks that will serve as input to the hash function. If we choose $L \leq 2^{r/4}$ we are in the condition to use the tighter bound presented in theorem 2.

V. Experiments and Results

This section describes the results of the experiments conducted to measure the performance of the our proposed self-encryption method. The goal of the following experiments is to measure the performance of our proposed self-encryption method. One of the objectives is to highlight the performance when the plaintext is split into different numbers of chunks. Hence, the experiment highlights the impact of the number of data chunks on the encryption execution time. On the other hand, considering that the size of the plaintext also influences the performance of the algorithm, it was also varied in the experiments, and the impacts were analyzed.

Another objective is to analyse the latency of our self-encryption method when operating in OTP-like mode (see Sect. III, step ①, OTP-like mode). Finally, since our proposed method has inherited the self-signed identifier feature from ID-based self-encryption [17], a comparison between both is also provided.

For the experiments, an 11th Gen Intel® Core™ i7-1185G7 @ 3.00GHz 8 CPU computer was used, and all benchmarked workloads ran synchronously, with no multi-threading or multi-tasking applied.

In the following, we investigate the execution time of our proposed self-encryption method when the number of chunks and the size of the plaintext vary. The measurements were performed for 4 different sizes of plaintext: 100KB, 1MB, 10MB, 100MB, and for 3 different plaintext split methods: 3, 6 and 9 chunks. The results are the average values of 1000 independent executions, and are as shown in Figure 2.

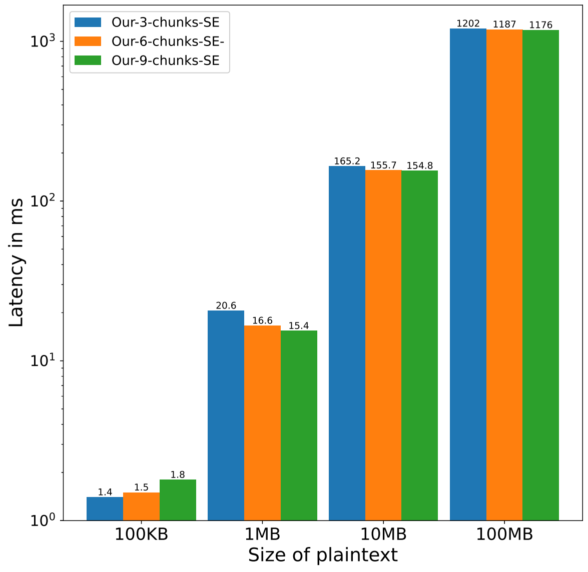

Fig. 2. Execution time of our self-encryption method, with variation on the number of data chunks (3, 6, 9) and the size of the plaintext (100KB, 1MB, 10MB, 100MB). "SE" refers to self-encryption

The results show that the difference between the number of chunks does not have a significant impact on the execution times. In secret sharing use cases, this feature is beneficial as a relatively high number of encrypted chunks and secret keys (i.e., 6, 9) can be issued with an insignificant time penalty compared to the default 3 chunks. However, in this experiment, the number of chunks is within a reasonable range, as a higher number may result in higher latency.

The second experiment highlights the latencies of our self-encryption method when the OTP-like mode is applied. As mentioned in Sect. III, step ①, OTP-like mode, in this mode the AES encrypted chunks have a length of 64 bytes and are XOR-ed with a secret key of the same length, which implies that the final encrypted chunks are one-time-padded data blocks. For this experiment, the plaintext size varies from 100KB to 1MB. The plaintext size is limited to 1MB, because having the same length key as the ciphertext is impractical. The results are the average values of 100 independent executions, and are shown in Figure 3.

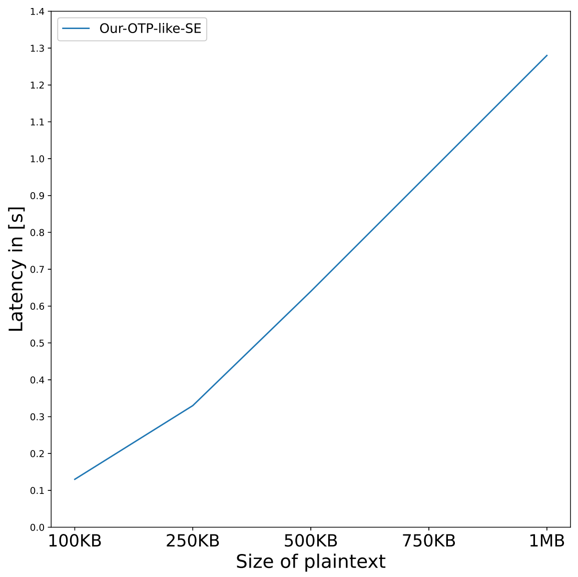

Fig. 3. Execution time of our self-encryption method when the OTP-like mode is applied: the plaintext is divided into 48 bytes chunks. Hence, each AES encrypted data chunk of 64 bytes is XOR-ed with a secret key of 64 bytes. Plaintext varies from 100KB to 1MB, with steps of 250KB. "SE" refers to self-encryption

The results show that the latency increases nearly linearly with the size of the plaintext. Comparing the results presented in Figure 2 and 3, it is clear that the OTP-like mode introduces additional latency. However, using the self-encryption in OTP-like mode can make sense when the plaintext size is small, like a typical password. Moreover, when the self-encryption is applied for the encryption of passwords, the application of further secret sharing schemes can also be avoided as each secret sharing participants can obtain one encrypted chunk of the encrypted password and one secret key produced by the self-encryption.

The objective of the third experiment is to compare our proposed self-encryption method with the ID-based self-encryption proposed in [17]. It must be noted that the latter is written in the Rust language, while our implementation uses Golang. The plaintext was split into 3 chunks, and the length of the plaintext varied from 100KB to 100MB. The results average the values of 100 independent executions, and are as shown in Figure 4. The results show that our implementation

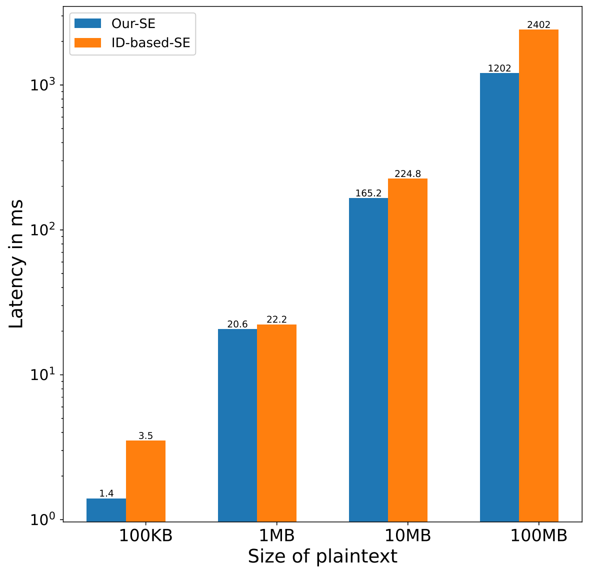

Fig. 4. Comparison between the execution times of our self-encryption method and the original method. Plaintext size varies from 100KB to 100MB. "SE" refers to Self-Encryption

performs better for that specific configuration (three encrypted chunks are generated and a self-signed identifier is added to the encrypted chunks).

VI. System Evaluation

As previously mentioned the original self-encryption and the encryption method proposed in [17] did not provide randomness during the key generation for the AES encryption phase. Therefore, our solution concatenates the data chunks with random bytes, which are then hashed and used as keys and initial values in the AES encryption blocks and values in the XOR phase of the encryption.

Since the key generation is issued by performing the hash value of the chunk concatenated with a random value, it is crucial to verify if the hash value "looks random". According to [13], and the analysis performed in this paper, when a 512 bit hash function with "cascade construction" receives inputs where the last 512 bits contains enough entropy, the distribution of the generated hash values are close to uniform. In our implementation of the key generation, plaintext chunks are concatenated with 1024 bits of random values, which implies that the hash values produced with SHA-512 are also close to uniform, since the last blocks used by the "compression function" contain a high amount of random values. To experimentally prove this property, we tested our key generation with *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* [24], issued by NIST.

For each experiment, the plaintext in the key generation contained only zeros. In each experiment, 1000 sequences of 1 million bits per sequence were used. The significance level of the test is set to 0.01 (α), which implies that a sequence is considered to be random with a confidence of 99%. In the first experiments, we applied the OTP-like mode of our encryption method, resulting in 4.259.840 different data chunks; and the key generation was repeated until 1Gbits of keys was achieved. In certain cases, 1 million bits per sequence for Non-periodic Template Matchings and Random Excursions tests were not enough, as some of the sub-tests showed non uniformity. Therefore we applied 500 sequences of 2 million bits per sequence. The longer sequences provided a more accurate testing, and the tests passed with a proportion of 488/500 binary sequences. In the second experiment the input data was chunked into 4 pieces, and the key generation was repeated until 1Gbits of keys was achieved. Table I shows the proportion of the sequence passing for both experiments.

TABLE I
Proportion of Sequences Passing when the significance level α is 0.01 (P-values ≥ 0.01) and the number of keys per encryption varies. Results are obtained after running *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* by NIST

	Number of keys per encryption	Proportion of Sequences Passing for P-values >0.01
1st Experiment	4 259 840	0.9899
2nd Experiment	4	0.9902

According to the results, we can assume that key generation of our proposed method is random, with a confidence level of 99%. This empirical result is close to the theoretical result described in Sect. IV.

Furthermore, in Table II we present a comparison between the original, ID-based and our proposed self-encryption method. The results in table II highlights performance of the different implementations in term of execution time, particularly when the plaintext size is significant. In addition, the table demonstrates the presence of Privacy Amplification and security proof. Lastly, it demonstrates whether the method includes the feature of protection against data deduplication.

VII. Conclusion

Many use cases require a high level of data security and data privacy, especially in IoT-based scenarios, where the protection of personal data is crucial. Deploying a strong and efficient encryption method based on "traditional" encryption components, such as AES and OTP, remains a challenge.

TABLE II
COMPARISON BETWEEN THE RELATED WORK AND OUR PROPOSED METHOD FOR SELF-ENCRYPTION

	Original [12]	ID-based [17]	Our
Contains self-signed ID	No	Yes	Yes
Security Proof	No	No	Yes
Protection against deduplication	Yes	Yes	No
Privacy Amplification	No	No	Yes
Execution time (for 100MB)	Unknown	2.4 s	1.2 s

Our self-encryption method is improved in terms of security upon those proposed in [12] and [17], as it is enriched with two possible encryption methods: the default mode allows the specification of the number of chunks the data must be divided into, providing a security level of 2^r with r equal to 512 bits. The OTP-like mode creates 48 byte-long chunks that are AES encrypted and XOR-ed with 64 byte-long keys, providing a security level close to the perfect secrecy. As another significant improvement, the weak secret key generation is amplified by using Privacy Amplification by applying additive randomness to the plaintext and SHA-2 cryptographic hash function. We can also conclude that the performance of our implementation is improved compared to the related work, since our implementation is able to encrypt 1MB of plaintext in less than 21 milliseconds, and 100MB of plaintext in less than 1,2 seconds. The OTP-like mode provides a remarkably fast encryption, with high security level, when the plaintext is small in size (e.g., 100KB was encrypted in 120 milliseconds). The usability of our implementation is also improved, since it can be used not only as a back-end service but also as an application in gateways and edge devices, which are considered as main components of today's IoT networks connecting constrained devices to the Internet.

ACKNOWLEDGMENT

This research is supported by European Union's Horizon 2020 research and innovation programme under grant agreement No. 101021727 (IRIS), and No. 101070052 (TANGO).

REFERENCES

[1] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano, "IoT-based Smart Cities: A survey," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, 2016, pp. 1–6.

[2] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation," in *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*, 2018, pp. 1–4.

[3] J. T. Mentzer *et al.*, "Defining supply chain management," *Journal of Business Logistics*, vol. 22, no. 2, pp. 1–25, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2158-1592.2001.tb00001.x

[4] M. A. Ferrag *et al.*, "A systematic review of data protection and privacy preservation schemes for smart grid communications," *Sustainable Cities and Society*, vol. 38, pp. 806–835, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670717308399

[5] L. Gerrits, R. Kromes, and F. Verdier, "A True Decentralized Implementation Based on IoT and Blockchain: a Vehicle Accident Use Case," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, 2020, pp. 1–6.

[6] H. H. Nguyen, F. Mirza, M. A. Naeem, and M. Nguyen, "A review on IoT healthcare monitoring applications and a vision for transforming sensor data into real-time clinical feedback," in *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2017, pp. 257–262.

[7] C. Dobre and F. Xhafa, "Intelligent services for Big Data science," *Future Generation Computer Systems*, vol. 37, pp. 267–281, 2014, special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X13001593

[8] S. Singh and D. Prasad, "Wireless body area network (WBAN): A review of schemes and protocols," *Materials Today: Proceedings*, vol. 49, pp. 3488–3496, 2022, national Conference on Functional Materials: Emerging Technologies and Applications in Materials Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321041912

[9] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "IoT Privacy and Security: Challenges and Solutions," *Applied Sciences*, vol. 10, no. 12, p. 4102, 2020. [Online]. Available: https://www.proquest.com/scholarly-journals/iot-privacy-security-challenges-solutions/docview/2415039549/se-2

[10] E. Bertino, "Data Security and Privacy: Concepts, Approaches, and Research Directions," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2016, pp. 400–407.

[11] P. Yang, N. Xiong, and J. Ren, "Data security and privacy protection for cloud storage: A survey," *IEEE Access*, vol. 8, pp. 131 723–131 740, 2020.

[12] D. Irvine, "Self encrypting data," 2010.

[13] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, "Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes," in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 494–510.

[14] C. Bennett, G. Brassard, C. Crepeau, and U. Maurer, "Generalized privacy amplification," *IEEE Transactions on Information Theory*, vol. 41, no. 6, pp. 1915–1923, 1995.

[15] M. Tomamichel, C. Schaffner, A. Smith, and R. Renner, "Leftover Hashing Against Quantum Side Information," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5524–5535, 2011.

[16] W. Yang, R. F. Schaefer, and H. V. Poor, "Privacy Amplification: Recent Developments and Applications," in *2018 International Symposium on Information Theory and Its Applications (ISITA)*, 2018, pp. 120–124.

[17] I. Grishkov, R. Kromes, T. Giannetsos, and K. Liang, "ID-Based Self-encryption via Hyperledger Fabric Based Smart Contract," in *Blockchain Technology and Emerging Technologies*, W. Meng and W. Li, Eds. Cham: Springer Nature Switzerland, 2023, pp. 3–18.

[18] K. Wüst and A. Gervais, "Do you Need a Blockchain?" in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 45–54.

[19] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," 2014.

[20] "Maidsafe github repository : self-encryption," Mar. 2022. [Online]. Available: https://github.com/maidsafe/self_encryption

[21] D. Stinson, "Universal hashing and authentication codes," *Designs, Codes and Cryptography*, vol. 4, no. 3, pp. 369–380, 1994.

[22] E. Andreeva, A. Bogdanov, Y. Dodis, B. Mennink, and J. P. Steinberger, "On the Indifferentiability of Key-Alternating Ciphers," Cryptology ePrint Archive, Paper 2013/061, 2013, https://eprint.iacr.org/2013/061. [Online]. Available: https://eprint.iacr.org/2013/061

[23] S. Chen and J. P. Steinberger, "Tight Security Bounds for Key-Alternating Ciphers," in *EUROCRYPT*. Springer, 2014, pp. 327–350. [Online]. Available: https://www.iacr.org/archive/eurocrypt2014/84410214/84410214.pdf

[24] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Booz-allen and hamilton inc mclean va, Tech. Rep., 2001.