Linearly homomorphic authenticated encryption with provable correctness and public verifiability

Patrick Struck, Lucas Schabhüser, Denise Demirel, Johannes Buchmann

Technische Universität Darmstadt, Germany
patrick.struck@stud.tu-darmstadt.de
{lschabhueser,ddemirel,buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. In this work the first linearly homomorphic authenticated encryption scheme with public verifiability and provable correctness, called LEPCoV, is presented. It improves the initial proposal by avoiding false negatives during the verification algorithm. This work provides a detailed description of LEPCoV, a comparison with the original scheme, a security and correctness proof, and a performance analysis showing that all algorithms run in reasonable time for parameters that are currently considered secure. The scheme presented here allows a user to outsource computations on encrypted data to the cloud, such that any third party can verify the correctness of the computations without having access to the original data. This makes this work an important contribution to cloud computing and applications where operations on sensitive data have to be performed, such as statistics on medical records and tallying of electronically cast votes.

Keywords: Authenticated encryption · Public verifiability · Cloud computing

1 Introduction

In this work the first "**L**inearly homomorphic authenticated **E**ncryption with **P**rovable **Co**rrectness and public **V**erifiability" (LEPCoV) scheme is presented. It improves Catalano et al.'s instantiated scheme [12] by avoiding false negatives during the verification algorithm.

Outsourcing data and computations to the cloud has become an increasingly important aspect of IT. These new techniques provide a higher level of efficiency and flexibility and are therefore very valuable for private and commercial users. However, they also pose new risks for data security. Thus, secure outsourcing is a highly relevant research field. Cloud technologies must ensure that no malicious party gets access to the outsourced data and that no unauthorized modifications can be performed, i.e. the solutions must provide confidentiality and integrity. Both security goals can be provided by encrypting and, respectively, signing the data before outsourcing it to the cloud.

To allow for computations on the outsourced data, encryption and signature schemes with homomorphic properties were developed. However, so far most

works focused on improving either of these schemes. Thus, Catalano et al. [12] developed a framework called "linearly homomorphic authenticated encryption with public verifiability" (LAEPuV) that allows to combine both primitives into one unified solution. They show that their framework can be instantiated with the Paillier cryptosystem and any linearly homomorphic signature scheme supporting the same message space. Furthermore, they provide a concrete instantiation using a variant of the linearly homomorphic signature scheme by Catalano et al. [11]. Since their primitive is linearly homomorphic, operations can be performed directly on the signed encrypted data and the correctness of the outcome can be verified. However, their concrete instantiation leads to false negatives, i.e. there are many functions for which the verification algorithm rejects correct computations on honestly generated ciphers. Thus, their solution does not provide correctness for all functions to be evaluated. Note that this affects the proposed instantiation rather than the generic construction. Furthermore, so far no work has tested the efficiency of their solution in practice.

Our Contribution In this paper we propose an instantiation for LAEPuV, called LEPCoV, based on [12] that does not lead to false negatives. Besides a detailed description of LEPCoV, we also present a comparison with the scheme proposed by Catalano et al. highlighting our improvements. Furthermore, we prove that our solution is secure and ensures correctness when evaluating functions over authenticated encrypted data. Another shortcoming of the work by Catalano et al. is that an efficiency evaluation is missing. Measuring the runtime of an instantiation is important before putting it into practice. Thus, we run a performance analysis for different security parameters and dataset sizes. The tests show that our algorithms run in reasonable time for parameters that are currently considered secure. In addition, further efficiency improvements are possible and highlighted at the end of this work.

Structure Our work is structured as follows. After providing the relevant definitions and the framework for LAEPuV in Section 2, we describe the instantiated scheme by Catalano et al. [12] in Section 3. Based on this, in Section 4 we point out the shortcomings of the scheme particularly with respect to correctness. Following this, in Section 5 we show how the correctness of the original solution can be improved, present our revised scheme LEPCoV, and prove its security and correctness. Finally, in Section 6 we demonstrate the practical use of our instantiation by providing the average runtimes of the algorithms based on our implementation and conclude in Section 7 with a summary of our contribution and possible future work.

1.1 Related work

There are several homomorphic encryption schemes, like Paillier [19], ElGamal [14], and Benaloh [6], which allow computations on messages by performing a corresponding computation on the ciphers. Anyhow, none of these schemes address authenticity nor integrity of the data encrypted.

A general definition of homomorphic signature schemes is given by Johnson et al. [17], as a redefined version of the concept by Desmedt [13]. Linearly homomorphic signature schemes have been defined by Boneh et al. [7]. Based on this, other works [2, 3, 4, 9, 10, 11, 15, 16], which provide either frameworks or realizations, have been proposed. However, these schemes keep neither the input data nor the output data confidential.

Authenticated encryption schemes aim at providing both privacy and authenticity. An and Bellare, for instance, introduced in [1] a new paradigm called *encryption with redundancy* achieving both security goals by adding some redundant information to the data to be encrypted. Later, Bellare and Namprempre [5] defined the term *authenticated encryption* together with corresponding security aspects. However, both works consider symmetric encryption and do not provide a solution for asymmetric encryption. Thus, closer to the setting described in this paper is the term *homomorphic authenticated encryption* defined by Joo and Yun [18]. While their scheme allows more functionalities, it is neither practical nor does it provide public verifiability. Thus, Catalano et al. [12] proposed a framework and an instantiation for a linearly homomorphic authenticated encryption scheme providing public verifiability. In this work we further improve their instantiation by providing provable correctness and a higher level of efficiency.

2 Notation and Preliminaries

In this section we provide the notation and preliminaries needed for our construction. We also give an intuition to the setup proposed by Catalano et al. [12] followed by the definition for Linearly Homomorphic Authenticated Encryption with Public Verifiability (LAEPuV). Afterwards, we present the hardness assumptions on which the security of the instantiation proposed by Catalano et al. and correspondingly our solution is based on.

2.1 Notation

Throughout this work we write $[k] = \{1, 2, ..., k\}$ for the natural number less or equal than k. For two integers $a, b \in \mathbb{Z}$, we write $\lfloor \frac{a}{b} \rfloor$ for the integer division of a and b, $a \mid b$ if a is a factor of b, and $a \nmid b$ if a is not a factor of b.

For a set S we write $s \xleftarrow{\$} S$ to indicate that s is chosen uniformly at random from S. We use $\mathcal{H}$ to describe a family of collision resistant hash functions which images can be interpreted as elements of $\mathbb{Z}_{N_E^2}$, where $N_E = p \cdot q$ for two primes p, q of equal size.

The i-th unit vector of $\mathbb{Z}^k$ is denoted by $\boldsymbol{e}_i$. We denote functions as vectors of coefficients, i.e. $\boldsymbol{f} = (f_1, ..., f_k)$. Note that for $\boldsymbol{f} = \boldsymbol{e}_i$ function evaluation $\boldsymbol{f}(m_1, \ldots, m_k)$ returns m_i.

2.2 Setup

Catalano et al. [12] introduced a cryptographic primitive called LAEPuV that allows a user Alice to outsource encrypted data and computations on this data to the cloud. For this to be secure the cloud must keep the data received confidential and provide measures that allow verifying the integrity of the computation results. Optimally, the results are publicly verifiable enabling third parties such as external auditors to perform these checks.

To ensure confidentiality Alice could encrypt her data using a homomorphic encryption scheme. Due to its homomorphic properties, functions can be evaluated over the messages by evaluating corresponding functions over the ciphers. This allows Alice to outsource the computations to a cloud such that it neither learns the input nor the result. However, Alice has to trust that the cloud evaluates the functions correctly.

To ensure integrity of the result Alice could sign her data using a homomorphic signature scheme before outsourcing it to the cloud. This allows Alice to delegate computations such that Alice, or any third party on behalf of Alice, can verify the correctness of the computations. However, without using an encryption scheme to encrypt the data, the cloud would learn the input and the output of the computations. Thus, both schemes must be combined. More precisely, Alice encrypts her data, signs the ciphers, and asks the cloud to evaluate the function over the ciphers. When Alice receives the resulting cipher along with its (homomorphically computed) signature from the cloud, she can verify the computation using the signature and obtain the message by decrypting the cipher.

A naive combination of these primitives requires that the cipher space of the encryption scheme and the message space of the homomorphic signature scheme are equal. The message space of the Paillier cryptosystem is $\mathbb{Z}_N$, where $N = pq$ for two primes p, q of equal size while the corresponding cipher space is $\mathbb{Z}_{N^2}$. This leads to a performance problem as the homomorphic signature scheme has to support a significantly larger message space than the Paillier cryptosystem. Thus, Catalano et al. [12] proposed a method which allows combining the Paillier cryptosystem with a homomorphic signature scheme in a more efficient manner. Instead of signing the ciphers, the scheme masks the ciphers and signs the decrypted masked ciphers which have the same size as the original messages. The framework for the combination of both schemes will be presented in the next subsection. Details regarding the instantiated scheme by Catalano et al. follow in Section 3.

2.3 LAEPuV

Catalano et al. [12] introduced a cryptographic primitive called linearly homomorphic authenticated encryption with public verifiability (LAEPuV). These schemes allow Alice to outsource encrypted data to the cloud such that the cloud can do computations for Alice which are publicly verifiable. Below, we formally define LAEPuV schemes.

Definition 1 (Linearly Homomorphic Authenticated Encryption with Public Verifiability (LAEPuV) [12]). *A linearly homomorphic authenticated encryption with public verifiability (LAEPuV) scheme is a tuple of five PPT algorithms* $\mathcal{L} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AVerify}, \mathsf{ADecrypt}, \mathsf{AEval})$*:*

$\mathsf{AKeyGen}(1^\kappa, k)$**:** *The input is a security parameter* κ *and the maximum number* k *of encrypted messages in each dataset. The output is a key pair* $(\mathsf{sk}, \mathsf{pk})$*, where* sk *is the secret key for decrypting and signing and* pk *is the public key used for verification and evaluation. The message space* $\mathcal{M}$*, the cipher space* $\mathcal{C}$*, and dataset identifier space* $\mathcal{D}$ *are implicitly defined by the public key* pk*.*

$\mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m)$**:** *The input is a secret key* sk*, a dataset identifier* τ*, an index* $i \in [k]$*, and a message* m*. The output is a cipher* c*.*

$\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f})$**:** *The input is a public key* pk*, a dataset identifier* τ*, a cipher* c*, and a linear function* $\boldsymbol{f}$*. The output is either* 1*, i.e. the cipher is valid, or* 0*, i.e. the cipher is invalid.*

$\mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f})$**:** *The input is a secret key* sk*, a dataset identifier* τ*, a cipher* c*, and a linear function* $\boldsymbol{f}$*. The output is a message* m *if* c *is valid and* $\perp$ *if* c *is invalid, respectively.*

$\mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]})$**:** *The input is a public key* pk*, a dataset identifier* τ*, a linear function* $\boldsymbol{f}$*, and* k *ciphers* $\{c_i\}_{i\in[k]}$*. The output is a cipher* c*.*

In the following we provide the definitions for both the security and the correctness of linearly homomorphic authenticated encryption with public verifiability (LAEPuV) schemes.

Definition 2. *We call a linearly homomorphic authenticated encryption with public verifiability scheme* $\mathcal{L} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AVerify}, \mathsf{ADecrypt}, \mathsf{AEval})$ *LH-IND-CCA secure, if the advantage of an adversary in the LH-IND-CCA game [12] is negligible in the security parameter* κ*.*

Definition 3. *We call a linearly homomorphic authenticated encryption with public verifiability scheme* $\mathcal{L} = (\mathsf{AKeyGen}, \mathsf{AEncrypt}, \mathsf{AVerify}, \mathsf{ADecrypt}, \mathsf{AEval})$ *correct, if for any key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\kappa, k)$ *the three conditions below are satisfied.*

Condition 1 *For any message* $m \in \mathcal{M}$*, any dataset identifier* $\tau \in \mathcal{D}$*, and any index* $i \in [k]$ *it holds that*

$$\mathsf{ADecrypt}(\mathsf{sk}, \tau, \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m), \boldsymbol{e}_i) = m.$$

Condition 2 *For any cipher* $c \in \mathcal{C}$*, any dataset identifier* $\tau \in \mathcal{D}$*, and any linear function* $\boldsymbol{f} = (f_1, ..., f_k) \in \mathbb{Z}_{N_E}^k$ *it holds that*

$$\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f}) = 1 \Leftrightarrow \exists m \in \mathcal{M} : \mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f}) = m.$$

Condition 3 *For any dataset identifier* $\tau \in \mathcal{D}$*, any messages* $m_1, ..., m_k \in \mathcal{M}$ *with corresponding ciphers* $c_1, ..., c_k \in \mathcal{C}$ *such that* $c_i \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m_i)$ *for* $i \in [k]$*, and any linear function* $\boldsymbol{f} = (f_1, ..., f_k) \in \mathbb{Z}_{N_E}^k$ *it holds that*

$$\mathsf{ADecrypt}(\mathsf{sk}, \tau, \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]}), \boldsymbol{f}) = \boldsymbol{f}(m_1, ..., m_k).$$

2.4 Hardness Assumptions

Below we define the hardness assumptions needed for [12] and our construction, i.e. the decisional composite residuosity assumption (DCRA) and the strong RSA assumption.

Definition 4 (Decisional Composite Residuosity Assumption [12]). *We say the decisional composite residuosity assumption ($DCRA$) holds if there exists no PPT $\mathcal{A}$ that can distinguish between a random element from $\mathbb{Z}^*_{N^2}$ and one from the set $\{z^N : z \in \mathbb{Z}^*_{N^2}\}$ (i.e. the set of N-th residues modulo N^2), when N is the product of two random primes proper size.*

Definition 5 (Strong RSA Assumption [11]). *Let N be a random RSA modulus of length κ, where $\kappa \in \mathbb{N}$ is the security parameter, and z be a random element in $\mathbb{Z}_N$. Then we say that the strong RSA assumption holds if for any PPT adversary $\mathcal{A}$ it holds that*

$$Pr[(y, e) \leftarrow \mathcal{A}(N, z) : y^e = z \mod N \wedge e \neq 1] \leq \mathsf{negl}(\kappa).$$

3 LAEPuV scheme CMP14 by Catalano et al.

Catalano et al. [12] proposed the first linearly homomorphic authenticated encryption with public verifiability scheme, henceforth referred to as CMP14. The scheme is based on the Paillier cryptosystem [19] and a variant of the linearly homomorphic signature scheme by Catalano et al. [11].

Instead of simply signing the cipher, the idea of the scheme is as follows. It encrypts the message contained in a dataset using the Paillier cryptosystem. The resulting cipher is masked by multiplying it with the hash of the dataset identifier concatenated with the index of the message within the dataset. This masked cipher is decrypted and the resulting message is signed using the linearly homomorphic signature scheme. Hereby, the message space of the linearly homomorphic signature scheme can be of size $\mathbb{Z}_{N_E}$, where $\mathbb{Z}_{N_E}$ is the message space of the Paillier cryptosystem, instead of the larger cipher space $\mathbb{Z}_{N_E^2}$. We describe the scheme below.

$\mathsf{AKeyGen}(1^\kappa, k)$: On input a security parameter κ and an integer k, the algorithm samples four (safe) primes p_E, q_E, p_S and q_S of size $\kappa/2$ such that for $N_E = p_E q_E$ and $N_S = p_S q_S$ it holds that $\varphi(N_S) = (p_S - 1)(q_S - 1)$ and N_E are coprime, i.e. $\gcd(N_E, \varphi(N_S)) = 1$. Subsequently, it samples an element $g \in \mathbb{Z}^*_{N_E^2}$ of order N_E and $k + 2$ elements $g_0, g_1, h_1, ..., h_k$ uniformly at random from $\mathbb{Z}^*_{N_S}$. Then, it chooses an (efficiently computable) injective function H_p which maps arbitrary strings to prime numbers of length $l < \kappa/2$ and a hash function $H \leftarrow \mathcal{H}$. The algorithm returns the key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = (p_E, q_E, p_S, q_S)$ and $\mathsf{pk} = (N_E, g, N_S, g_0, g_1, h_1, ..., h_k, H, H_p)$.

$\mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m)$: On input a secret key sk, a dataset identifier τ, an index $i \in [k]$, and a message m, the algorithm computes the Paillier encryption

$C \leftarrow g^m \beta^{N_E} \mod N_E^2$ of m, where $\beta \xleftarrow{\$} \mathbb{Z}_{N_E}^*$, and the masking $R \leftarrow H(\tau||i)$. It computes a tuple $(a, b) \in \mathbb{Z}_{N_E} \times \mathbb{Z}_{N_E}^*$ such that $g^a b^{N_E} = CR \mod N_E^2$ by invoking the following steps [19]:

- Obtain a by decrypting CR using the Paillier cryptosystem [19].
- Compute $c_* \leftarrow CRg^{-a} \mod N_E$.
- Set $b \leftarrow c_*^{N_E^{-1} \mod \lambda} \mod N_E$, where $\lambda = \mathsf{lcm}(p_E - 1, q_E - 1)$.

Then, it obtains the prime $e \leftarrow H_p(\tau)$, chooses a random element $s \in \mathbb{Z}_{eN_E}$, and computes x such that

$$x^{eN_E} = g_0^s h_i g_1^a \mod N_S$$

Finally, it returns the cipher $c = (C, a, b, e, s, \tau, x)$.

$\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f})$: On input a public key pk, a dataset identifier τ, a cipher $c = (C, a, b, e, s, \tau, x)$, and a linear function $\boldsymbol{f} = (f_1, ..., f_k)$, the algorithm computes $e \leftarrow H_p(\tau)$, $\boldsymbol{f'} = \frac{\boldsymbol{f} - (\boldsymbol{f} \mod eN_E)}{eN_E}$, and $\hat{x} = \frac{x}{\prod_{i=1}^k h_i^{f_i'}}$. It checks if

$$a, s \in \mathbb{Z}_{eN_E} \tag{1}$$

$$\hat{x}^{eN_E} = g_0^s \prod_{i=1}^k h_i^{f_i} g_1^a \mod N_S \tag{2}$$

$$g^a b^{N_E} = C \prod_{i=1}^k H(\tau||i)^{f_i} \mod N_E^2 \tag{3}$$

If all checks pass, the algorithm returns 1, i.e. c is a valid cipher. Otherwise, it returns 0, i.e. c is an invalid cipher.

$\mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f})$: On input a secret key sk, a dataset identifier τ, a cipher $c = (C, a, b, e, s, \tau, x)$, and a linear function $\boldsymbol{f} = (f_1, ..., f_k)$, the algorithm runs $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f})$ to check if c is a valid cipher, i.e. whether $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f}) = 1$. If true, the algorithm returns the message m obtained by decrypting C using the Paillier cryptosystem. Otherwise, it returns $\perp$.

$\mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i \in [k]})$: On input a public key pk, a dataset identifier τ, a linear function $\boldsymbol{f} = (f_1, ..., f_k)$, and k ciphers $c_i = (C_i, a_i, b_i, e_i, s_i, \tau_i, x_i)$, the algorithm first checks if there exists $i \in [k]$ such that $\tau \neq \tau_i$ or $H_p(\tau) \neq e_i$. If true, the algorithm aborts. Otherwise, the algorithm computes $e \leftarrow H_p(\tau)$ and

$$C \leftarrow \prod_{i=1}^k C_i^{f_i} \mod N_E^2 \qquad a \leftarrow \sum_{i=1}^k f_i a_i \mod N_E$$

$$b \leftarrow \prod_{i=1}^k b_i^{f_i} \mod N_E^2 \qquad s \leftarrow \sum_{i=1}^k f_i s_i \mod eN_E$$

$$s' \leftarrow \left(\sum_{i=1}^k f_i s_i - s \right) / (eN_E) \qquad x = \frac{\prod_{i=1}^k x_i^{f_i}}{g_0^{s'}} \mod N_S$$

Then, it returns the cipher $c = (C, a, b, e, s, \tau, x)$.

4 Shortcomings of CMP14

In this section we describe the shortcomings of CMP14. First, we show that there is a restriction regarding the functions which can be evaluated as most functions lead to false negatives during the verification algorithm, i.e. the verification algorithm rejects ciphers although they were generated honestly and correctly. It follows that CMP14 is not correct according to Definition 3 since these functions violate Condition 3. Following this, we describe a practical issue regarding the injective function H_p which makes the encryption infeasible in some datasets.

4.1 Restricted Function Evaluation

On a high level, CMP14 rejects honestly generated ciphers if the value a is reduced modulo N_E during AEval. In other words for a function $\boldsymbol{f} = (f_1, ..., f_k)$ and ciphers $c_i = (C_i, a_i, b_i, e, s_i, \tau, x_i)$, where $\boldsymbol{f}(a_1, ..., a_k) = \sum_{i=1}^{k} f_i a_i \geq N_E$, the verification of the cipher $c \leftarrow \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]})$ fails. However, there are a few exceptions for which the verification of the cipher does not fail, namely the functions $\boldsymbol{f} = (f_1, ..., f_k)$ for which $\boldsymbol{f}(a_1, ..., a_k)$ is a multiple of the order of g_1, i.e. $\boldsymbol{f}(a_1, ..., a_k) = q \cdot \mathsf{ord}(g_1)$, where $q \in \mathbb{N}$. Note that this is unlikely to happen, especially if using safe primes while generating keys.

We emphasize that Alice has no control over the values a_i, because they are obtained by decrypting the masked cipher CR, where C is the cipher of the message and $R \leftarrow H(\tau||i)$ is the masking. It follows that Alice can not simply adjust the functions $\boldsymbol{f}$ to ensure that $\boldsymbol{f}(a_1, ..., a_k) < N_E$.

To show this more formally, we first provide a lemma which specifies the type of functions which leads to the modulo operation during AEval and show an inequality that holds for this type of functions. Then, we show that for this type of functions the verification algorithm of CMP14 fails even though the ciphers were computed honestly and correctly, which violates Condition 3.

Lemma 1. *Let* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\kappa, k)$*, where* $\mathsf{sk} = (p_E, q_E, p_S, q_S)$ *and* $\mathsf{pk} = (N_E, g, N_S, g_0, g_1, h_1, ..., h_k, H, H_p)$ *be a key pair and* $a_1, ..., a_k \in \mathbb{Z}_{N_E}$ *be the decrypted masked (Paillier) ciphers of ciphers* $c_1, ..., c_k$*. Then, any linear function* $\boldsymbol{f} = (f_1, ..., f_k)$ *with* $\mathsf{ord}(g_1) \nmid q$ *, where* $q = \left\lfloor \frac{\boldsymbol{f}(a_1,...,a_k)}{N_E} \right\rfloor = \left\lfloor \frac{\sum_{i=1}^{k} f_i a_i}{N_E} \right\rfloor \in \mathbb{N}$*, leads to a modulo operation during* AEval*, i.e. it holds that*

$$g_1^{\sum_{i=1}^{k} f_i a_i} \neq g_1^{\sum_{i=1}^{k} f_i a_i \mod N_E} \mod N_S.$$

Proof. In order to prove the statement, it suffices to show that the exponents modulo the order of g_1 are not equal. Note that $\gcd(\mathsf{ord}(g_1), N_E) = 1$. This follows directly from the fact that, during AKeyGen, N_E and N_S are generated such that $\gcd(N_E, \varphi(N_S)) = 1$ and $\mathsf{ord}(g_1) \mid \varphi(N_S)$.

Write $\boldsymbol{f}(a_1, ..., a_k) = \sum_{i=1}^{k} f_i a_i = qN_E + r$, where $r \in \{0, ..., N_E - 1\}$ and $q \in \mathbb{N}$ such that $\mathsf{ord}(g_1) \nmid q$. It holds that

$$\sum_{i=1}^{k} f_i a_i = qN_E + r \mod \mathsf{ord}(g_1)$$

$$
\begin{aligned}
&\neq r && \mod \mathsf{ord}(g_1)\\
&= qN_E + r \mod N_E && \mod \mathsf{ord}(g_1)\\
&= \sum_{i=1}^{k} f_i a_i \mod N_E && \mod \mathsf{ord}(g_1)
\end{aligned}
$$

Hence, $g_1^{\sum_{i=1}^{k} f_i a_i} \neq g_1^{\sum_{i=1}^{k} f_i a_i \mod N_E} \mod N_S$. □

Proposition 1. *Let* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\kappa, k)$ *be an honestly generated key pair,* $c_i = (C_i, a_i, b_i, e, s_i, \tau, x_i) \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m_i)$ *be ciphers of messages* $m_i \in \mathbb{Z}_{N_E}$ *for* $i \in [k]$*, where* τ *is an arbitrary dataset identifier. For any linear function* $\boldsymbol{f} = (f_1, ..., f_k)$*, where* $\boldsymbol{f}(a_1, ..., a_k)$ *leads to a modulo operation during* AEval *as defined in Lemma 1, it holds that*

$$\mathsf{ADecrypt}(\mathsf{sk}, \tau, \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]}), \boldsymbol{f}) \neq \boldsymbol{f}(m_1, ..., m_k)$$

which violates Condition 3 of Definition 3.

Proof. Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\kappa, k)$ be an honestly generated key pair, τ be an arbitrary dataset identifier, $m_1, ..., m_k \in \mathbb{Z}_{N_E}$ be arbitrary messages, and $c_i = (C_i, a_i, b_i, e, s_i, \tau, x_i) \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m_i)$ be the resulting ciphers. Let $\boldsymbol{f} = (f_1, ..., f_k)$ be a linear function such that $\boldsymbol{f}(a_1, ..., a_k)$ satisfies Lemma 1 and $c \leftarrow \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]})$ be the cipher that is obtained by evaluating the function $\boldsymbol{f}$ over the ciphers c_i. It holds that

$$
\begin{aligned}
\hat{x}^{eN_E} &= \left(\frac{x}{\prod_{i=1}^{k} h_i^{f_i'}}\right)^{eN_E}\\
&= \left(\frac{\prod_{i=1}^{k} x_i^{f_i}}{g_0^{s'} \prod_{i=1}^{k} h_i^{f_i'}}\right)^{eN_E}\\
&= \frac{\prod_{i=1}^{k} (g_0^{s_i} h_i g_1^{a_i})^{f_i}}{\left(g_0^{\frac{\sum_{i=1}^{k} f_i s_i - s}{eN_E}} \prod_{i=1}^{k} h_i^{\frac{f_i - (f_i \mod eN_E)}{eN_E}}\right)^{eN_E}}\\
&= \frac{g_0^{\sum_{i=1}^{k} f_i s_i} \prod_{i=1}^{k} (h_i^{f_i}) g_1^{\sum_{i=1}^{k} f_i a_i}}{g_0^{\sum_{i=1}^{k} f_i s_i - s} \prod_{i=1}^{k} h_i^{f_i - (f_i \mod eN_E)}}\\
&= g_0^{s} \prod_{i=1}^{k} h_i^{f_i \mod eN_E} g_1^{\sum_{i=1}^{k} f_i a_i}\\
&\overset{Lemma\ 1}{\neq} g_0^{s} \prod_{i=1}^{k} h_i^{f_i} g_1^{\sum_{i=1}^{k} f_i a_i \mod N_E}\\
&= g_0^{s} \prod_{i=1}^{k} h_i^{f_i} g_1^{a}
\end{aligned}
$$

This yields $\mathsf{AVerify}(\mathsf{pk}, \tau, \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]}), \boldsymbol{f}) = 0$, hence, it holds that $\mathsf{ADecrypt}(\mathsf{sk}, \tau, \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]}), \boldsymbol{f}) = \perp \neq \boldsymbol{f}(m_1, ..., m_k)$ which violates Condition 3. □

Proposition 1 proves that CMP14 is not correct according to Definition 3 as there occur false negatives. However, it does not state whether this shortcoming affects the practical use of the scheme, i.e. whether Alice can prevent false negatives by choosing the messages and functions carefully.

Hence, we implemented and tested CMP14. The results show that CMP14 is impractical since, regardless of the security parameter κ and the dataset size k, even small functions, e.g. adding two messages, mainly lead to false negatives. Below we provide an example which illustrates this.

Table 1. Example values for false negatives.

Key pair					
Secret key sk		Public key pk			
$p_E = 151$	$q_E = 149$	$N_E = 22499$	$N_E^2 = 506205001$	$g = 457224679$	
Encryption values					
m	β	R	m	β	R
$m_1 = 17$	$\beta_1 = 14296$	$R_1 = 64489750$	$m_3 = 19$	$\beta_3 = 1576$	$R_3 = 157182719$
$m_2 = 4$	$\beta_2 = 17791$	$R_2 = 18170490$	$m_4 = 92$	$\beta_4 = 6190$	$R_4 = 365721887$

Table 1 shows a 16 bit key pair (the relevant values) and four messages m_i along with their random encryption values β_i and the maskings R_i, which allow to compute the values a_i. While the addition of m_4 and either m_2 or m_3 works, the addition of m_1 and any other message m_i as well as the addition of m_2 and m_3 lead to a false negative. We stress that, regardless of the actual function values, combining three or four of these messages, i.e. at most one function value is 0, always yields a false negative. We further emphasize that the values in Table 1 are generated arbitrarily and not specially constructed for this shortcoming.

4.2 Infeasible Encryption in Some Datasets

CMP14 also suffers from a minor practical issue regarding the function H_p which binds a unique prime to each dataset. There is no check whether the prime and the order of the group $\mathbb{Z}_{N_S}$ are coprime. If that is not the case, computing the signature value x during AEncrypt is equivalent to breaking the RSA assumption, which is assumed to be infeasible.

More formally, let e be a prime such that $\gcd(eN_E, \varphi(N_S)) \neq 1$. Under the strong RSA assumption (see Definition 5) one can not compute x such that $x^{eN_E} = g_0^s h_i g_1^a \mod N_S$ in polynomial time. Hence, for the dataset identified by τ, where $H_p(\tau) = e$, AEncrypt can not be executed efficiently.

5 Our improved scheme LEPCoV

In this section we describe our improved scheme LEPCoV based on CMP14. We start with a high-level description of the changes followed by a detailed description of the scheme. Finally, we show that our scheme is both secure and correct according to Definition 2 and Definition 3, respectively.

5.1 High-level Description of our Changes

First, we simplify the verification of ciphers. We require that all functions to be evaluated are described as vectors of coefficients where each coefficient is a value smaller than N_E. Note that this restriction still allows to express all linear functions. More precisely, let $m \in \mathbb{Z}_{N_E}$ be a message, $\beta \in \mathbb{Z}^*_{N_E}$ be a random encryption value, and $C = g^m \beta^{N_E} \in \mathbb{Z}_{N_E^2}$ be a Paillier cipher of this message. For any integer f, it holds that

$$\begin{aligned} \mathsf{Decrypt}(C^f) &= fm \mod N_E \\ &= (f \mod N_E)m \mod N_E \\ &= \mathsf{Decrypt}(C^{f \mod N_E}) \end{aligned}$$

where $\mathsf{Decrypt}(C)$ is the Paillier decryption of C. This allows us to simplify the verification algorithm as the values $\boldsymbol{f'}$ and $\hat{x}$ are no longer necessary.

To address the shortcoming that efficient encryption is not feasible in some datasets, as described in Section 4.2, we do not use the function H_p. Instead, Alice generates the prime for each dataset by herself and binds the prime to the dataset by signing the dataset identifier and the prime using a signature scheme $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Hence, for each dataset, Alice can generate a unique prime e such that $\gcd(eN_E, \varphi(N_S)) = 1$, which guarantees that Alice can encrypt messages in this dataset.

The other core problem of CMP14 is the evaluation of functions for which the value a is reduced during AEval, as descried in Section 4.1. Note that due to $\gcd(N_E, \varphi(N_S)) = 1$, one can not generate g_1 of order N_E to trivially fix this shortcoming. Thus, to avoid this problem, we change the computation of x during AEval. We stress that, in CMP14, the problem if a is reduced during AEval does not occur if s is reduced during AEval, because x is multiplied with the inverse of $g_0^{s'}$. Hence, similar to s' we compute a new value a'. Note however that simply multiplying x also with the inverse of $g_1^{a'}$ does not suffice as a and s are not elements within the same group. Instead, we multiply x with the inverse of $g_1^{a'e^{-1}}$, where e^{-1} is the inverse element of e modulo $\varphi(N_S)$. Since the efficient computation of e^{-1} requires the factorization of N_S, Alice has to compute and publish e^{-1}.

Based on the changes described above, a cipher $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x)$ of a message m in LEPCoV, contains the Paillier encryption C of m, the decrypted masked cipher a along with its random encryption value b, the prime e and its inverse element e^{-1}, the signature σ_e of $\tau||e$, the random signature value s, the

dataset identifier τ, and the signature x of a. Since a LAEPuV scheme does not require the complete dataset to verify a cipher, we have to address that Alice might not store the ciphers locally. Note that the values e, e^{-1}, σ_e are the same for each cipher within the same dataset. Thus, it is sufficient to assume that Alice keeps record of these values, i.e. she has access to a list L which contains tuples of dataset identifiers τ and the values e, e^{-1}, σ_e. If Alice runs $\mathsf{AEncrypt}$ with dataset identifier τ the first time, she computes e, e^{-1}, σ_e and stores $(\tau, e, e^{-1}, \sigma_e)$ in the list L. Otherwise, Alice takes the values from the list L. We emphasize that the list L allows Alice to generate a unique prime for each dataset.

5.2 Description of the scheme

Below we provide a detailed description of LEPCoV and highlight the differences compared to $\mathsf{CMP14}$. In the description, $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ describes a signature scheme used to bind primes to datasets.

$\mathsf{AKeyGen}(1^\kappa, k)$: On input a security parameter κ and an integer k, the algorithm samples the four (safe) primes p_E, q_E, p_S, q_S, the group elements $g_0, g_1, h_1, ..., h_k \in \mathbb{Z}^*_{N_S}$ and $g \in \mathbb{Z}^*_{N_E^2}$ of order N_E, and the hash function $H \in \mathcal{H}$ as described for the original approach. In addition, it runs $\mathsf{KeyGen}(1^\kappa)$ to obtain a key pair $(\mathsf{sk}_S, \mathsf{pk}_S)$ of $\mathcal{S}$ and returns the key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = (p_E, q_E, p_S, q_S, \mathsf{sk}_S)$ and $\mathsf{pk} = (N_E, g, N_S, g_0, g_1, h_1, ..., h_k, H, \mathsf{pk}_S)$ along with an empty list L.

$\mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m)$: On input a secret key sk, a dataset identifier τ, an index $i \in [k]$, and a message m, the algorithm computes R, the Paillier encryption C of the message m, and (a, b) as described for $\mathsf{CMP14}$. In addition, if τ is used the first time, it chooses a not yet used prime e of length $l < \kappa/2$ such that $\gcd(eN_E, \varphi(N_S)) = 1$, computes its inverse $e^{-1} \mod \varphi(N_S)$ and its signature $\sigma_e \leftarrow \mathsf{Sign}(\mathsf{sk}_S, \tau||e)$, and stores $(\tau, e, e^{-1}, \sigma_e)$ in the list L. Otherwise, it takes $(\tau, e, e^{-1}, \sigma_e)$ from the list L. Then, it chooses $s \xleftarrow{\$} \mathbb{Z}_{eN_E}$, computes the value x such that $x^{eN_E} = g_0^s h_i g_1^a \mod N_S$, and returns the cipher $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x)$.

$\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f})$: On input a public key pk, a dataset identifier τ, a cipher $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x)$, and a linear function $\boldsymbol{f} = (f_1, .., f_k)$, the algorithm checks if

$$\mathsf{Verify}(\mathsf{pk}_S, \tau||e, \sigma_e) = 1$$

$$a, s \in \mathbb{Z}_{eN_E}$$

$$x^{eN_E} = g_0^s \prod_{i=1}^{k} h_i^{f_i} g_1^a \mod N_S$$

$$g^a b^{N_E} = C \prod_{i=1}^{k} H(\tau||i)^{f_i} \mod N_E^2$$

If all four checks pass, the algorithm returns 1, i.e. c is a valid cipher. Otherwise, it returns 0, i.e. c is an invalid cipher.

$\mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f})$: On input a secret key sk, a dataset identifier τ, a cipher $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x)$, and a linear function $\boldsymbol{f} = (f_1, ..., f_k)$, the algorithm runs $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f})$ to check if c is a valid cipher. If true, the algorithm returns the message m obtained by decrypting C using the Paillier cryptosystem. Otherwise, it returns $\perp$.

$\mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i\in[k]})$: On input a public key pk, a dataset identifier τ, a linear function $\boldsymbol{f}$, and k ciphers $c_i = (C_i, a_i, b_i, e_i, e_i^{-1}, \sigma_{e_i}, s_i, \tau_i, x_i)$, the algorithm checks if there exists an index $l \in [k]$ such that $\tau \neq \tau_l$, like in $\mathsf{CMP14}$, or $\mathsf{Verify}(\mathsf{pk}_S, \tau||e_l, \sigma_{e_l}) = 0$. Furthermore, the algorithm checks if there are two indexes $i \neq j \in [k]$ such that $e_i \neq e_j$. If one of the checks is true, the algorithm aborts. Otherwise, the algorithm sets $e = e_1$, $e^{-1} = e_1^{-1}$, $\sigma_e = \sigma_{e_1}$, computes C, a, b, s, and s' like in the original approach, and

$$a' \leftarrow \left(\sum_{i=1}^{k} f_i a_i - a\right)/N_E \qquad x = \frac{\prod_{i=1}^{k} x_i^{f_i}}{g_0^{s'} g_1^{a'e^{-1}}} \mod N_S$$

Then, it returns the cipher $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x)$.

5.3 Security

The security of our improved scheme LEPCoV, according to Definition 2, is given in the theorem below.

Theorem 1. *The linearly homomorphic authenticated encryption with public verifiability scheme* LEPCoV*, described above, is secure according to Definition 2.*

Proof. For lack of space, we only sketch the proof. In $\mathsf{CMP14}$, the injective function H_p ensures that each dataset is associated with a unique prime e. In LEPCoV, these primes are generated by Alice, hence, she can generate a unique prime for each dataset. The signature scheme $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is used to bind primes to datasets, thus, the security of $\mathcal{S}$ guarantees that only the prime numbers chosen by Alice are accepted.

In case of the original scheme, an adversary $\mathcal{A}$ has to compute the eN_E-th root to forge a signature, which, under the strong RSA assumption (see Definition 5), is infeasible for the parameters chosen. In LEPCoV, Alice publishes the inverse of e, hence, the adversary has only to compute the N_E-th root in order to forge a signature. However, under the strong RSA assumption, this remains infeasible for the parameters chosen.

Based on these changes, the following statement by Catalano et al. [12] applies to the linearly homomorphic signature scheme used in LEPCoV: *The signature scheme is an unforgeable signature scheme under chosen message attacks according to the definition by Boneh and Freeman [8], if the strong RSA assumption (see Definition 5) holds [12, Theorem 31].*

Based on this, the following statement proves the security of LEPCoV: *In the random oracle model, if (1) the DCRA (see Definition 4) holds, (2) H is a random oracle and (3) the linearly homomorphic signature scheme over $\mathbb{Z}_{N_E}$ is unforgeable (under chosen message attacks), the scheme* LEPCoV*, described in Section 5, is LH-IND-CCA secure [12, Theorem 6].* □

5.4 Correctness

The correctness of LEPCoV, described above, follows from the following theorem which is proven below.

Theorem 2. *The linearly homomorphic authenticated encryption with public verifiability scheme* LEPCoV*, described above, is correct according to Definition 3.*

Proof. In the following, we show that each condition described in Definition 3 holds. Throughout this proof, let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{AKeyGen}(1^\kappa, k)$ be a key pair, where $\mathsf{sk} = (p_E, q_E, p_S, q_S, \mathsf{sk}_S)$ and $\mathsf{pk} = (N_E, g, N_S, g_0, g_1, h_1, ..., h_k, H, \mathsf{pk}_S)$.

Condition 1: Let $m \in \mathbb{Z}_{N_E}$ be an arbitrary message, τ be an arbitrary dataset identifier, $i \in [k]$, $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x) \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m)$ be the encryption of m, and $\boldsymbol{f} = \boldsymbol{e}_i$.

By construction we have $a, s \in \mathbb{Z}_{eN_E}$ and $\mathsf{Verify}(\mathsf{pk}_S, \tau||e, \sigma_e) = 1$. It holds that

$$x^{eN_E} = g_0^s h_i g_1^a = g_0^s h_i \prod_{\substack{j=1 \\ j \neq i}}^{k} h_j^0 g_1^a = g_0^s \prod_{j=1}^{k} h_j^{f_j} g_1^a$$

and

$$g^a b^{N_E} = CR = CH(\tau||i) = CH(\tau||i) \prod_{\substack{j=1 \\ j \neq i}}^{k} H(\tau||j)^0 = C \prod_{j=1}^{k} H(\tau||j)^{f_j}$$

which yields $\mathsf{AVerify}(\mathsf{pk}, \tau, \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m), \boldsymbol{f}) = 1$. Thus, ADecrypt returns the Paillier decryption of C, i.e. $\mathsf{ADecrypt}(\mathsf{sk}, \tau, \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m), \boldsymbol{e}_i) = m$.

Condition 2: We prove the equivalence by showing that both implications are satisfied.

$\Leftarrow$: Let $m \in \mathcal{M} = \mathbb{Z}_{N_E}$ be a message, $\boldsymbol{f} = (f_1, ..., f_k)$ be a linear function with $f_i < N_E$ for $i \in [k]$, and c be a cipher such that $\mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f}) = m$. The fact that $\mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f}) \neq \perp$ directly leads to $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f}) = 1$.

$\Rightarrow$: Let $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x) \in \mathcal{C}$ be a cipher, τ be a dataset identifier, and $\boldsymbol{f}$ be a linear function such that $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f}) = 1$. Since $\mathsf{ord}(g) = N_E$, this guarantees that the Paillier decryption of C yields $m \in \mathcal{M} = \mathbb{Z}_{N_E}$. Thus,

$$\exists m \in \mathcal{M} : \mathsf{ADecrypt}(\mathsf{sk}, \tau, c, \boldsymbol{f}) = m.$$

Condition 3: Let τ be an arbitrary dataset identifier, $m_1, ..., m_k \in \mathbb{Z}_{N_E}$ be messages, and $c_i = (C_i, a_i, b_i, e, e^{-1}, \sigma_e, s_i, \tau, x_i) \leftarrow \mathsf{AEncrypt}(\mathsf{sk}, \tau, i, m_i)$ be the cipher obtain by encrypting the message m_i for $i \in [k]$.

Let $\boldsymbol{f} = (f_1, ..., f_k)$ be a linear function such that $f_i < N_E$ for all $i \in [k]$ and $c = (C, a, b, e, e^{-1}, \sigma_e, s, \tau, x) \leftarrow \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i \in [k]})$ be the cipher obtained by evaluating the function $\boldsymbol{f}$ over the ciphers c_i.

By construction it holds that $\mathsf{Verify}(\mathsf{pk}_S, \tau||e, \sigma_e) = 1$. During AEval, s and a are reduced modulo eN_E and N_E, respectively. Thus, $s, a \in \mathbb{Z}_{eN_E}$. In order to show that $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f}) = 1$, it remains to show that

$$x^{eN_E} = g_0^s \prod_{i=1}^{k} h_i^{f_i} g_1^a \mod N_S \tag{4}$$

$$g^a b^{N_E} = C \prod_{i=1}^{k} H(\tau||i)^{f_i} \mod N_E^2 \tag{5}$$

For equation (4) we have

$$\begin{aligned}
x^{eN_E} &= \frac{(\prod_{i=1}^{k} x_i^{f_i})^{eN_E}}{(g_0^{s'} g_1^{a'e^{-1}})^{eN_E}} = \frac{\prod_{i=1}^{k} (g_0^{s_i} h_i g_1^{a_i})^{f_i}}{(g_0^{s'} g_1^{a'e^{-1}})^{eN_E}} \\
&= \frac{g_0^{\sum_{i=1}^{k} f_i s_i} \prod_{i=1}^{k} h_i^{f_i} g_1^{\sum_{i=1}^{k} f_i a_i}}{\left(g_0^{(\sum_{i=1}^{k} f_i s_i - s)/(eN_E)} g_1^{((\sum_{i=1}^{k} f_i a_i - a)/(N_E))e^{-1}}\right)^{eN_E}} \\
&= \frac{g_0^{\sum_{i=1}^{k} f_i s_i} \prod_{i=1}^{k} h_i^{f_i} g_1^{\sum_{i=1}^{k} f_i a_i}}{g_0^{\sum_{i=1}^{k} f_i s_i - s} \left(g_1^{(\sum_{i=1}^{k} f_i a_i - a)/(eN_E)}\right)^{eN_E}} \\
&= \frac{g_0^{\sum_{i=1}^{k} f_i s_i} \prod_{i=1}^{k} h_i^{f_i} g_1^{\sum_{i=1}^{k} f_i a_i}}{g_0^{\sum_{i=1}^{k} f_i s_i - s} g_1^{\sum_{i=1}^{k} f_i a_i - a}} = g_0^s \prod_{i=1}^{k} h_i^{f_i} g_1^a
\end{aligned}$$

For equation (5) we obtain

$$\begin{aligned}
C \prod_{i=1}^{k} H(\tau||i)^{f_i} &= \prod_{i=1}^{k} C_i^{f_i} \prod_{i=1}^{k} H(\tau||i)^{f_i} = \prod_{i=1}^{k} (C_i H(\tau||i))^{f_i} \\
&= \prod_{i=1}^{k} (g^{a_i} b_i^{N_E})^{f_i} = g^{\sum_{i=1}^{k} f_i a_i} \prod_{i=1}^{k} b_i^{f_i N_E} = g^a b^{N_E}
\end{aligned}$$

Thus, it holds that $\mathsf{AVerify}(\mathsf{pk}, \tau, c, \boldsymbol{f}) = 1$. Finally, we have

$$C = \prod_{i=1}^{k} C_i^{f_i} = \prod_{i=1}^{k} (g^{m_i} \beta_i^{N_E})^{f_i} = g^{\sum_{i=1}^{k} f_i m_i} \prod_{i=1}^{k} \beta_i^{f_i N_E}$$

hence Paillier decryption yields $\sum_{i=1}^{k} f_i m_i = \boldsymbol{f}(m_1, ..., m_k)$, which leads to

$$\mathsf{ADecrypt}(\mathsf{sk}, \tau, \mathsf{AEval}(\mathsf{pk}, \tau, \boldsymbol{f}, \{c_i\}_{i \in [k]}), \boldsymbol{f}) = \boldsymbol{f}(m_1, ..., m_k)$$

Thus, LEPCoV satisfies Condition 1 - 3 which proves the statement. □

We stress that g is an element of order N_E. Thus, the verification check in equation (5) does not fail if a is reduced during AEval. Also keep in mind that due to $\gcd(N_E, \varphi(N_S)) = 1$, one can not generate g_1 of order N_E to trivially fix the shortcoming of $\mathsf{CMP14}$ described in Section 4.1.

6 Implementation

We implemented LEPCoV in Java and measured the average runtimes of the algorithms on an Intel® Core M-5Y71 CPU @ 1.20GHz with 8GB RAM. We run our experiments for different security parameters $\kappa \in \{1024, 2048, 3072, 4096\}$ and dataset sizes $k \in \{50, 100, 500\}$. Note that AVerify is not considered in the experiments as its runtime is similar to ADecrypt.

Table 2. Average runtimes (in ms) of AKeyGen, AEncrypt, ADecrypt, and AEval for different security parameters κ and dataset sizes k.

	$\kappa = 1024$ bits			$\kappa = 2048$ bits		
	$k = 50$	$k = 100$	$k = 500$	$k = 50$	$k = 100$	$k = 500$
AKeyGen	285	299	346	2501	2686	2862
AEncrypt	65	62	69	502	537	560
ADecrypt	86	109	403	571	724	1925
AEval	57	112	1042	297	854	19995
	$\kappa = 3072$ bits			$\kappa = 4096$ bits		
	$k = 50$	$k = 100$	$k = 500$	$k = 50$	$k = 100$	$k = 500$
AKeyGen	10038	9994	10190	25279	25755	26040
AEncrypt	1804	1787	1791	4029	4040	4078
ADecrypt	1945	2290	9679	4199	4645	16810
AEval	737	1953	43211	1255	3250	67233

Table 2 summarizes the average runtimes of AKeyGen, AEncrypt, ADecrypt, and AEval. It shows that AKeyGen and AEval are, as expected, the most expensive algorithms followed by ADecrypt and AEncrypt. Note that AKeyGen is only performed once and AEval is outsourced to the cloud. Thus, Alice only has two run the two less expensive algorithms AEncrypt and ADecrypt. The runtime of AEncrypt depends only on the security parameter κ. Therefore, the constant and relatively cheap costs of the encryption allow executing it on a device with less computation power. The runtime of ADecrypt (and AVerify) depends, besides the security parameter κ, also on the dataset size k and can be executed on a more powerful device. For a security parameter of $\kappa = 2048$ bits, which is currently assumed secure, and dataset size $k \leq 100$, AEncrypt and ADecrypt take less than a second. Note that for growing dataset size k, ADecrypt becomes faster than AEval. It follows that the size of the datasets processed must be taken into account when considering this scheme for an application. However, further efficiency improvements, e.g. using the Chinese remainder theorem to speed up the Paillier cryptosystem, and implementation-based optimizations, like parallelized code, are still possible.

Summarized the tests show that for parameters that are currently considered secure all algorithms run in a reasonable amount of time.

7 Conclusion

In this paper we proposed the first provable correct linearly homomorphic authenticated encryption with public verifiability (LAEPuV) scheme LEPCoV that is based on the CMP14 scheme by Catalano et al. [12]. We showed to what extent our scheme improves the original approach, proved our scheme secure, and showed that all algorithms run in reasonable time for currently recommended security parameters.

For future work we plan to further improve the efficiency of our implementation by implementing additional optimizations. Furthermore, we aim at constructing homomorphic authenticated encryption schemes with public verifiability for a wider class of supported functions.

8 Acknowledgments

This work has been co-funded by the DFG as part of project "Long-Term Secure Archiving" within the CRC 1119 CROSSING. In addition, it has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No 644962.

References

[1] Jee Hea An and Mihir Bellare. Does encryption with redundancy provide authenticity? In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 512–528. Springer, 2001.

[2] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *International Workshop on Public Key Cryptography*, pages 17–34. Springer, 2011.

[3] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 367–385. Springer, 2012.

[4] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *International Workshop on Public Key Cryptography*, pages 386–404. Springer, 2013.

[5] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.

[6] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128, 1994.

[7] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *International Workshop on Public Key Cryptography*, pages 68–87. Springer, 2009.

[8] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 149–168. Springer, 2011.

[9] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2011.

[10] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *Theory of Cryptography*, pages 680–699. Springer, 2013.

[11] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In *Public Key Cryptography–PKC 2012*, pages 680–696. Springer, 2012.

[12] Dario Catalano, Antonio Marcedone, and Orazio Puglisi. Authenticating computation on groups: New homomorphic primitives and applications. In *Advances in Cryptology–ASIACRYPT 2014*, pages 193–212. Springer, 2014.

[13] Yvo Desmedt. Computer security by redefining what a computer is. In *Proceedings on the 1992-1993 workshop on New security paradigms*, pages 160–166. ACM, 1993.

[14] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 10–18. Springer, 1984.

[15] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *International Workshop on Public Key Cryptography*, pages 697–714. Springer, 2012.

[16] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *International Workshop on Public Key Cryptography*, pages 142–160. Springer, 2010.

[17] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *Cryptographers? Track at the RSA Conference*, pages 244–262. Springer, 2002.

[18] Chihong Joo and Aaram Yun. Homomorphic authenticated encryption secure against chosen-ciphertext attack. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 173–192. Springer, 2014.

[19] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptologyEUROCRYPT99*, pages 223–238. Springer, 1999.