Secure Cumulative Reward Maximization in Linear Stochastic Bandits

Radu Ciucanu[1], Anatole Delabrouille[2], Pascal Lafourcade[3], and Marta Soare[4]

[1] INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Orléans, France,
radu.ciucanu@insa-cvl.fr
[2] Univ. Bordeaux, LIMOS/LIFO, Clermont-Ferrand, France,
anatole.delabrouille@etu.u-bordeaux.fr
[3] Univ. Clermont Auvergne, LIMOS CNRS UMR 6158, Clermont-Ferrand, France,
pascal.lafourcade@uca.fr
[4] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, Orléans, France,
marta.soare@univ-orleans.fr

Abstract. The linear stochastic multi-armed bandit is a sequential learning setting, where, at each round, a learner chooses an arm and receives a stochastic reward based on an unknown linear function of the chosen arm. The goal is to collect as much reward as possible. Linear bandits have popular applications such as online recommendation based on user preferences, where obtaining a high reward means recommending an item with high expected rating. We address the security concerns that occur when outsourcing the data and the cumulative reward maximization algorithm to an honest-but-curious cloud. We propose LinUCB-DS, a distributed and secure protocol that achieves the same cumulative reward as the standard LinUCB algorithm, without disclosing to the cloud the linear function used to draw arm rewards. We formally prove the complexity and security properties of LinUCB-DS. We also show that LinUCB-DS can be easily adapted to secure the SpectralUCB algorithm, which improves LinUCB for a class of linear bandits. We show the feasibility of our protocols via a proof-of-concept experimental study using the MovieLens movie recommendation dataset.

1 Introduction

The *stochastic multi-armed bandit* game is a sequential learning framework, which consists of a repeated interaction between a learner and the environment. The learner is given a set of choices (arms) with unknown associated rewards and a limited number of allowed interactions with the environment (budget). With the goal of maximizing the sum of the observed rewards, the learner sequentially chooses an arm at each time step and the environment responds with a stochastic reward corresponding to the chosen arm. In the *linear stochastic bandit* setting, the input set of arms is a fixed subset of $\mathbb{R}^d$, revealed to the learner at the beginning of the game. When pulling an arm, the learner observes a noisy reward whose expected value is the inner product between the chosen

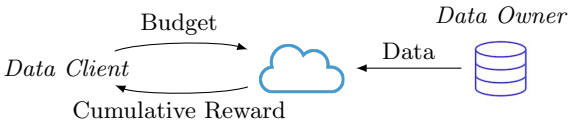


Fig. 1. Outsourcing Data and Computations.

arm and an unknown parameter characterizing the underlying linear function (common to all arms).

Stochastic linear bandits can be used to model online recommendation: the arms are the objects that might be recommended and a reward is the user's response to a recommendation e.g., the click through rate or the score associated to the recommendation. The recommender wants to maximize the sum of rewards, thus the recommender needs to predict which object is more likely to be of interest for a certain user. Also, while discovering such object, the recommender must not disappoint the user with too much bad recommendations. The unknown parameter of the reward function is the user preference, more precisely the weights that the user gives to each of the d features in assessing an item.

In this paper, we consider a scenario inspired from the *machine learning as a service* cloud computing model, where the machine learning data and algorithms are outsourced to the cloud, which yields inherent data security concerns [5]. We depict this scenario in Fig. 1 and we illustrate it next with an example. Let the *data owner* be a large company owning multiple surveys on many types of items and users. Then, let the *data client* be a small recommendation company that is willing to pay a budget to acquire a part of a survey i.e., to find out the cumulative reward obtainable for a subset of types of items and users. This is a classical scenario, where small companies benefit of large scale data without having to perform the survey themselves, and where large companies monetize their data. The accuracy of the result returned to the data client is correlated to the invested budget.

The machine learning as a service cloud computing model is useful when neither the data owner nor the data client want to perform the computations. They rather choose to entrust the computation to a third-party, for example to a public cloud such as Google Cloud Platform, Amazon Web Services, or Microsoft Azure. However, cloud providers do not usually address the fundamental problem of protecting data security. The outsourced data can be communicated over some network and processed on some machines where malicious cloud admins could learn and leak sensitive user data. The data owner wishes to remain the only one that has the complete knowledge of her data, and only the data client should be able to gain knowledge of the cumulative reward for which she paid.

We address the data security issues that occur when outsourcing the linear bandit data and cumulative reward maximization algorithm to a public cloud. We propose LinUCB-DS, a secure and distributed protocol based on the standard LinUCB algorithm [1], which yields the same cumulative reward as LinUCB while satisfying desirable security properties that we formally prove. The key

ingredients of LinUCB-DS are (i) *Paillier cryptographic scheme* that is additive homomorphic i.e., it allows to compute the encrypted value of the sum of two numbers, given only their ciphertexts, without revealing the numbers in plain, and (ii) *secure multi-party computation* i.e., the computation is split among two cloud participants, which can jointly compute the algorithm output, without revealing their partial input to each other.

Related Work. Algorithms based on computing *upper confidence bounds (UCB)* on arm values are commonly used for cumulative reward maximization strategies. The classical UCB algorithm [3] for multi-armed bandits has been applied to linear bandits in various works (for instance [1,2,12]) and is referred to as LinUCB, OFUL (Optimism in the Face of Uncertainty for Linear bandits), or LinRel (Linear Reinforcement Learning). Following [11, Chapter 19], we use LinUCB as a generic name for UCB applied to stochastic linear bandits and we specifically rely on the algorithm in [1], in the case where the set of arms is fixed.

There is a recent line of research on adding privacy-preserving guarantees to UCB-like algorithms, mostly using differential privacy techniques [7,13,16] including for linear bandits [15]. The use of differential privacy has a low impact on execution time compared to non-secured algorithms, but outputs different cumulative rewards. This is a consequence of the noise added to the input or the output of differentially-private algorithms. This difference propagates in the regret analysis, which suffers an additive or multiplicative factor compared to the regret of standard non-secured algorithms.

In contrast, our approach based on cryptographic schemes and secure multi-party computation implies heavier computations, but outputs exactly the same cumulative reward as standard non-secured algorithms. Hence, both approaches (differential privacy vs cryptography) have advantages and disadvantages. Secure bandit algorithms using cryptographic techniques have been already proposed for a different problem: best arm identification in multi-armed bandits [6].

To the best of our knowledge, our work is the first one that adds security guarantees to linear bandit algorithms using cryptographic techniques.

Summary of Contributions and Paper Organization. In Sect. 2 we introduce LinUCB algorithm and some cryptographic tools. Sect. 3 is the main contribution of the paper: we formalize the expected security properties, and we propose LinUCB-DS, a secure and distributed protocol based on LinUCB. We show its correctness and we analyze its theoretical complexity by characterizing the number of cryptographic operations. In Sect. 4, we show the security properties of LinUCB-DS. In Sect. 5, we present our experimental study based on the MovieLens dataset, which confirms the feasibility of LinUCB-DS. In Sect. 6, we show how our protocol can be easily adapted to secure SpectralUCB [17] that is another algorithm that relies on UCB in the linear setting.

2 Preliminaries

We introduce LinUCB algorithm, Paillier encryption, and IND-CPA security.

Input: Budget N and K arms $x_1, x_2, \ldots, x_K$ in $\mathbb{R}^d$
Constants: Regularizer $\gamma > 0$; confidence parameter $\delta > 0$; noise parameter $R > 0$; $S > 0$ such that $||\theta||_2 \leq S$; $L > 0$ such that $\forall\ i \in [\![K]\!], ||x_i||_2 \leq L$
Unknown environment: Expected arm values; the learner has access only to the output of reward function $pull(x_i)$
Output: Sum of observed rewards for all arms

/ Initialization: Pull an arm and initialize variables /
Let $r = pull(x_i)$ */ Random reward (scalar) for a randomly selected arm x_i /*
Let $s = r$ */ Sum of rewards of all arms (scalar) /*
Let $A = \gamma I_d + x_i x_i^\top$ */ $(d \times d)$ matrix /*
Let $b = r x_i$ */ $(d \times 1)$ vector /*

/ Exploration-Exploitation: At each round, pull an arm and update variables /
For $1 \leq t < N$
  Let $\widehat{\theta} = A^{-1} b$ */ Compute the regularized least-squares estimate of θ /*
  Let $\omega = R\sqrt{d \cdot \log(\frac{1+tL^2/\gamma}{\delta})} + \gamma^{\frac{1}{2}} \cdot S$ */ Exploration parameter /*
  / Compute the UCB arm score B_i (scalar) for each arm based on current $\widehat{\theta}$. First term for exploitation, second term for exploration /
  For $1 \leq i \leq K$
    Let $B_i = \langle x_i, \widehat{\theta} \rangle + \omega ||x_i||_{A^{-1}}$ */ With probability $\geq 1-\delta$, B_i is an UCB of $\langle x_i, \theta \rangle$ /*
  Let $x_m = \arg\max_{i \in [\![K]\!]} B_i$ */ Randomly choose among arms maximizing B_i /*
  Let $r = pull(x_m)$ */ Pull arm x_m and update the corresponding variables /*
  Let $s = s + r$
  Let $A = A + x_m x_m^\top$
  Let $b = b + r x_m$
Return s */ Return sum of observed rewards for all arms /*

Fig. 2. LinUCB Algorithm [1].

LinUCB. In cumulative reward maximization algorithms, the learner faces the so-called *exploration-exploitation dilemma*: at each round, she has to decide whether to *explore* arms with more uncertain associated values, or to *exploit* the information already acquired by selecting the arm with the seemingly largest value. UCB-like algorithms guide the exploration-exploitation trade-off by updating, after each new observed reward, a *score* for each arm, given by the upper-confidence bound of the estimated arm value. In LinUCB, the arm scores are based on a regularized least-squares estimate of the unknown parameter of the linear reward function. At the next round, the arm with the largest updated score is pulled. Following [1], we present the LinUCB algorithm in Fig. 2.

We rely on the following notations:

- $[\![z]\!]$ is the set $\{1, 2, \ldots, z\}$.
- K is the number of arms.
- N is the client's budget = the number of allowed arm pulls = the number of observed rewards.
- d is the space dimension = the size of each arm vector = the number of features of the unknown parameter θ.

• x_i (for $i \in [\![K]\!]$) is an arm = a $(d \times 1)$ vector; we assume that all arms are pairwise distinct.
• $||v||_2$ is the 2-norm of a $\mathbb{R}^d$ vector v.
• $||v||_A = \sqrt{v^\top A v}$ is the weighted 2-norm of a $\mathbb{R}^d$ vector v, where A is a $(d \times d)$ positive definite matrix.
• θ is a $(d \times 1)$ vector (unknown to the learner) that is the parameter of the linear reward function.
• $\langle x_i, \theta \rangle$ (for $i \in [\![K]\!]$) is a scalar (unknown to the learner) defining the expected reward value of arm x_i, computed as the dot product of vectors x_i and θ.
• $pull(x_i)$ is a function that returns a noisy reward $\langle x_i, \theta \rangle + \eta$, where the noise η is an R-sub-Gaussian random variable, where $R \geq 0$ is a fixed constant.
• $v(j)$ is the j^{th} element of a vector v and $M(j)$ is the j^{th} row of a matrix M.

Paillier Asymmetric Encryption. Paillier [14] is an asymmetric partial homomorphic encryption scheme defined by a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ and a security parameter λ. By 1^λ we denote the unary representation of λ, which is a standard notation in cryptography.

- $\mathcal{G}(1^\lambda)$ generates two prime numbers p and q according to λ, sets $n = p \cdot q$ and $\Lambda = \text{lcm}(p-1, q-1)$ (i.e., the least common multiple), generates the group $(\mathbb{Z}^*_{n^2}, \cdot)$, randomly picks $g \in \mathbb{Z}^*_{n^2}$ such that $M = (L(g^\Lambda \mod n^2))^{-1} \mod n$ exists, with $L(x) = (x-1)/n$. It sets $\mathsf{sk} = (\Lambda, M)$, $\mathsf{pk} = (n, g)$, it returns $(\mathsf{sk}, \mathsf{pk})$.
- $\mathcal{E}(m)$ randomly picks $r \in \mathbb{Z}^*_n$, computes $c = g^m \cdot r^n \mod n^2$ using pk, and outputs c.
- $\mathcal{D}(c)$ computes $m = L(c^\Lambda \mod n^2) \cdot M \mod n$ using sk, and outputs m.

Paillier's cryptosystem is *additive homomorphic*. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$. The product of the two associated ciphertexts with the public key $\mathsf{pk} = (n, g)$, denoted $c_1 = \mathcal{E}(m_1) = g^{m_1} \cdot r_1^n \mod n^2$ and $c_2 = \mathcal{E}(m_2) = g^{m_2} \cdot r_2^n \mod n^2$, is the encryption of the sum of m_1 and m_2. Indeed, we have: $\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) = c_1 \cdot c_2 \mod n^2 = (g^{m_1} \cdot r_1^n) \cdot (g^{m_2} \cdot r_2^n) \mod n^2 = (g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n) \mod n^2 = \mathcal{E}(m_1 + m_2)$.

It is also possible to compute the encryption of the product of a ciphertext and a plaintext: $\mathcal{E}(m_1)^{m_2} = c_1^{m_2} \mod n^2 = (g^{m_1} \cdot r^n)^{m_2} \mod n^2 = g^{m_1 \cdot m_2} \cdot r^{n \cdot m_2} \mod n^2 = g^{m_1 \cdot m_2} \cdot (r')^n \mod n^2 = \mathcal{E}(m_1 \cdot m_2)$.

Encryption and decryption with the public and private key of entity E are noted $\mathcal{E}_\mathsf{E}(.)$ and $\mathcal{D}_\mathsf{E}(.)$ respectively.

IND-CPA (INDistinguishability under Chosen-Plaintext Attack).

Let $\Pi = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a cryptographic scheme. The *probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ tries to break the security of Π. The IND-CPA game, denoted by $\text{EXP}(\mathcal{A})$, works as follows: the adversary $\mathcal{A}$ chooses two messages (m_0, m_1) and receives a challenge $c = \mathsf{Encrypt}(LR_b(m_0, m_1))$ from the *challenger* who selects a bit $b \in \{0, 1\}$ uniformly at random, and where $LR_b(m_0, m_1)$ is equal to m_0 if $b = 0$, and m_1 otherwise. The adversary, knowing m_0, m_1 and c, is allowed to perform any number of polynomial computations or encryptions of any messages, using the encryption oracle, in order to output a

guess b' of the encrypted message in c chosen by the challenger. Intuitively, Π is IND-CPA if there is no PPT adversary that can guess b with a probability significantly better than $\frac{1}{2}$. By $\alpha = \Pr[b' \leftarrow \text{EXP}(\mathcal{A}); b = b']$, we denote the probability that $\mathcal{A}$ correctly outputs her guessed bit b' when the bit chosen by the challenger in the experiment is b. A scheme is IND-CPA secure if $\alpha - \frac{1}{2}$ is negligible function in λ, where a function φ is negligible in λ, denoted $\text{negl}(\lambda)$, if for every positive polynomial $p(\cdot)$ and sufficiently large λ, $\varphi(\lambda) < 1/p(\lambda)$. Paillier is IND-CPA secure under the decisional composite residuosity assumption [14].

3 LinUCB-DS

We propose LinUCB-DS, a secure and distributed algorithm based on LinUCB cf. Fig. 2 in the setting from Fig. 1. We first list the desired security properties and the security hypothesis. We next outline the challenges of our problem setting and the ideas behind our solution. Then, we present the participants of LinUCB-DS and their pseudo-code. We end this section by arguing the correctness of LinUCB-DS and analyzing its cryptographic overhead.

Security Properties. We expect the following properties, which should hold until the end of the protocol:

1. No cloud node knows θ.
2. No cloud node knows the cumulative reward, nor any individual reward.
3. An external observer having captured all messages exchanged over the network does not know θ, the rewards, nor which arms have been pulled.

Security Hypothesis. We assume that the cloud is *honest-but-curious* i.e., it executes tasks dutifully, but tries to extract as much information as possible from the data that it sees. Our model follows a classical formulation [9] (Ch. 7.5, where *honest-but-curious* is denoted *semi-honest*), in particular (i) each cloud node is trusted: it correctly does the required computations, it does not sniff the network and it does not collude with other nodes, and (ii) an external observer has access to all messages exchanged over the network. The aforementioned security model is of practical interest in a real-world cloud environment. In particular, to satisfy all our theoretical security properties while achieving the no-collusion hypothesis, it suffices to host each cloud node of our protocol by a different cloud provider. This should be feasible as our protocol requires only two cloud nodes.

Challenges. Our problem could be theoretically solved by using a fully homomorphic encryption scheme [8], which allows to compute any function directly in the encrypted domain. However, it remains an open question how to make such a scheme work fast and be accurate in practice when working with real numbers. Indeed, by using state-of-the-art fully homomorphic systems (e.g., Microsoft SEAL[5] or HElib[6]), it is not currently possible to obtain exactly the same output as the standard, non-encrypted version when securing LinUCB.

[5] `https://github.com/Microsoft/SEAL`
[6] `http://homenc.github.io/HElib/`

Paillier additive-homomorphic encryption and secure multi-party computation (where some party does computations on reals in clear) allow us to develop a protocol satisfying the expected security properties while being feasible in practice. Indeed, if the data owner outsources $\mathcal{E}(\theta(1)), \ldots, \mathcal{E}(\theta(d))$, the cloud can generate an encrypted reward of arm x_i as $\mathcal{E}(\theta(1))^{x_i(1)} \cdots \mathcal{E}(\theta(d))^{x_i(d)}\mathcal{E}(\eta)$. Then, variables s, b, and B_i can be also updated in the encrypted domain. Since the B_i are encrypted, the cloud cannot compare them and we need to find a secure way to decrypt and compare the B_i. The idea (already known in the literature e.g., in the context of private outsourced sort [4]) is that the data owner does not use the data client's public key to outsource θ, but instead uses the key of a second cloud node whose only task is to compare the B_i. At the end, the cloud nodes perform a key switching without revealing s to the cloud.

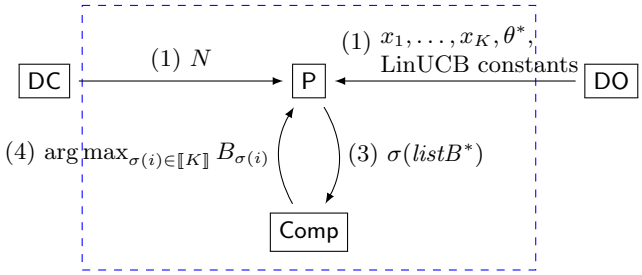
Participants of LinUCB-DS (2 of them in the cloud).
• DO (data owner) outsources data to the cloud.
• DC (data client) sends the budget to the cloud. At the end, she receives the result of the algorithm.
• P is the principal node of the cloud, which receives the arms, budget, and encrypted θ. This node pulls the arms and updates the variables.
• Comp is a cloud node whose Paillier public key is used to outsource θ. Comp is the only node that can decrypt and compare the B_i.

Next, we present the three phases of LinUCB-DS: *Initialization*, *Exploration-Exploitation*, and *Key Switching*. The numbers of the steps refer to those from Fig. 3 and 4. We rely on the following additional notation:
• z^* is $\mathcal{E}_{\mathsf{E}}(z)$, where E is clear from the context.
• $EDP_{\mathsf{pk}}(v^*, w) = \prod_{j=1}^{d}(v(j)^*)^{w(j)}$ is the *E*ncrypted *D*ot *P*roduct of v^* (vector of size d of data encrypted with pk) and w (vector of size d of data in clear).
• $pull^*(x_i) = EDP_{\mathsf{Comp}}(\theta^*, x_i)\mathcal{E}_{\mathsf{Comp}}(\eta)$ is the encrypted reward drawn for an arm x_i using the encrypted unknown parameter θ^* and a scalar noise η cf. Sect. 2. This computation is the encrypted version of the scalar product defined by the $pull(.)$ function and the homomorphic addition of η.

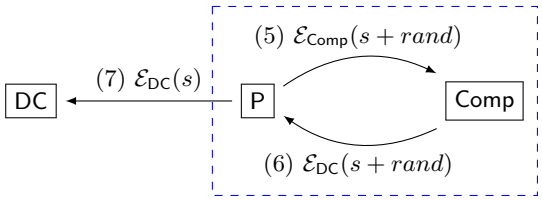
Initialization.
• Step (1): DC sends to P the budget N. Furthermore, DO sends to P the arms $x_1, \ldots, x_K$, the encrypted unknown parameter $\theta^* = (\theta(1)^*, \ldots, \theta(d)^*) = (\mathcal{E}_{\mathsf{Comp}}(\theta(1)), \ldots, \mathcal{E}_{\mathsf{Comp}}(\theta(d)))$, as well as all algorithm constants cf. Fig. 2.

• Step (2): P randomly chooses an arm x_i, generates an encrypted reward $r^* = pull^*(x_i)$, and initializes variables:
$s^* = r^*$.
$A = \gamma I_d + x_i x_i^\top$.
$b^* = (b^*(1), \ldots, b^*(d)) = ((r^*)^{x_i(1)}, \ldots, (r^*)^{x_i(d)})$.

Exploration-Exploitation. At each round, an interaction between P and Comp occurs to decide the next arm to pull. More precisely, for $1 \leq t < N$, we repeat:
• Step (3):



(a) Initialization and Exploration-Exploitation Phases.



(b) Key Switching Phase.

Fig. 3. Messages exchanged between LinUCB-DS participants. Steps 3 and 4 are done $N-1$ times. The dashed rectangle is the cloud. Details on each step are given in Sect. 3.

(i) P computes $\widehat{\theta^*}$ as the product between matrix A^{-1} and vector b^*:
$\widehat{\theta^*} = (\widehat{\theta}(1)^*, \ldots, \widehat{\theta}(d)^*) = (EDP_{\mathsf{Comp}}(b^*, A^{-1}(1)), \ldots, EDP_{\mathsf{Comp}}(b^*, A^{-1}(d)))$.
(ii) P computes $listB^*$ that is the list of $B_1^*, \ldots, B_K^*$ such that, for each arm x_i, $B_i^* = EDP_{\mathsf{Comp}}(\widehat{\theta^*}, x_i)\mathcal{E}_{\mathsf{Comp}}(\omega||x_i||_{A^{-1}})$, where ω is the exploration parameter cf. Fig. 2. Then, P generates a random permutation $\sigma : [\![K]\!] \to [\![K]\!]$ and sends $\sigma(listB^*)$ to Comp.

• Step (4):

(i) Comp decrypts each element of the permuted list of encrypted B_i values. Then, Comp sends $\arg\max_{\sigma(i)\in[\![K]\!]} B_{\sigma(i)}$ to P.
(ii) P retrieves x_m that is an arm maximizing B_i. Then, P computes $r^* = pull^*(x_m)$ and updates the variables:
$s^* = s^* r^*$.
$A = A + x_m x_m^\top$.
$b^* = (b^*(1), \ldots, b^*(d)) = (b^*(1)(r^*)^{x_m(1)}, \ldots, b^*(d)(r^*)^{x_m(d)})$.

Key Switching. The sum of rewards is re-encrypted using the DC's public key.
• Step (5): P chooses a random number *rand* and sends to Comp the following $\mathcal{E}_{\mathsf{Comp}}(rand)s^* = \mathcal{E}_{\mathsf{Comp}}(rand + s)$.
• Step (6): Comp decrypts $\mathcal{E}_{\mathsf{Comp}}(rand + s)$, encrypts the result using DC's public key, and sends it back to P. Note that Comp sees in clear $rand + s$ but cannot infer s because it does not know *rand*.
• Step (7): P sends $\mathcal{E}_{\mathsf{DC}}(s) = \mathcal{E}_{\mathsf{DC}}(rand + s)\mathcal{E}_{\mathsf{DC}}(-rand)$ to DC, which decrypts $\mathcal{E}_{\mathsf{DC}}(s)$ and learns s.

/ *Initialization: Pull an arm and initialize variables* /
Receive N from DC / *Step 1* /
Receive $x_1, \ldots, x_K$, θ^*, and algorithm constants from DO
Randomly choose $i \in [\![K]\!]$ / *Step 2* /
Let $r^* = pull^*(x_i)$
Let $s^* = r^*$
Let $A = \gamma I_d + x_i x_i^\top$
Let $b^* = ((r^*)^{x_i(1)}, \ldots, (r^*)^{x_i(d)})$
/ *Exploration-Exploitation: At each round, pull an arm and update variables* /
For $1 \leq t < N$
  Let $\widehat{\theta}^* = (EDP_{\mathsf{Comp}}(b^*, A^{-1}(1)), \ldots, EDP_{\mathsf{Comp}}(b^*, A^{-1}(d)))$ / *Step 3* /
  For $1 \leq i \leq K$
    Let $B_i^* = listB^*(i) = EDP_{\mathsf{Comp}}(\widehat{\theta}^*, x_i)\mathcal{E}_{\mathsf{Comp}}(\omega||x_i||_{A^{-1}})$
  Randomly choose permutation $\sigma : [\![K]\!] \to [\![K]\!]$
  Send $\sigma(listB^*)$ to Comp
  Receive $\sigma(m)$ from Comp / *Step 4* /
  Let $m = \sigma^{-1}(\sigma(m))$
  Let $r^* = pull^*(x_m)$
  Let $s^* = s^* r^*$
  Let $A = A + x_m x_m^\top$
  Let $b^* = (b^*(1)(r^*)^{x_m(1)}, \ldots, b^*(d)(r^*)^{x_m(d)})$
/ *Key Switching* /
Randomly choose $rand \in \mathbb{R}$ / *Step 5* /
Send $\mathcal{E}_{\mathsf{Comp}}(rand)s^*$ to Comp
Receive $\mathcal{E}_{\mathsf{DC}}(rand + s)$ from Comp / *Step 6* /
Send $\mathcal{E}_{\mathsf{DC}}(s) = \mathcal{E}_{\mathsf{DC}}(rand + s)\mathcal{E}_{\mathsf{DC}}(-rand)$ to DC / *Step 7* /

(a) Pseudo-code of P.

/ *Exploration-Exploitation* /
For $1 \leq t < N$ / *Step 4* /
  Receive $\sigma(listB^*)$ from P
  For $1 \leq i \leq K$ / *Decrypt all elements of the permuted list of B_i values* /
    Let $B_{\sigma(i)} = \mathcal{D}_{\mathsf{Comp}}(\sigma(listB^*)(i))$
  Send $\arg\max_{\sigma(i) \in [\![K]\!]} B_{\sigma(i)}$ to P
/ *Key Switching* /
Receive $\mathcal{E}_{\mathsf{Comp}}(rand)s^*$ from P / *Step 5* /
Let $rand + s = \mathcal{D}_{\mathsf{Comp}}(\mathcal{E}_{\mathsf{Comp}}(rand)s^*)$ / *Step 6* /
Send $\mathcal{E}_{\mathsf{DC}}(rand + s)$ to P

(b) Pseudo-code of Comp.

Fig. 4. Pseudo-code of cloud nodes.

Phase	*Encryptions*	*Decryptions*	*Additions*	*Multiplications*
Initialization	$d+1$		d	$2d$
Exploration - Exploitation	$N-1$	$(N-1)K$	$(N-1)(d^2+Kd+2d)$	$(N-1)(d^2+Kd+2d)$
Key Switching	1	2	2	

Fig. 5. Number of Paillier Cryptographic Operations.

This concludes the presentation of the steps of LinUCB-DS. Before ending this section, we analyze the correctness and complexity of LinUCB-DS.

Correctness. LinUCB-DS outputs exactly the same cumulative reward as LinUCB and it computes the same reward for the same arm at each round. The reason is that the Paillier scheme does not change the value of any element, hence throughout the exact computations on encrypted numbers we conserve the correctness. In fact, Paillier scheme operates in $\mathbb{N}$, but the values of the arms and θ are defined in $\mathbb{R}$. Furthermore, $\widehat{\theta}$ and the B_i are computed using matrix inverse, square root and division (all these operations are done in plain, but the results are added or multiplied to ciphered values). Consequently, we need to use Paillier with real numbers, or the other way around, use real numbers as integers. Transforming a value in order to use it with an encryption scheme is called *encoding*. The encoding[7] we perform on a decimal number is simply to multiply it by a power of 16 to make it an integer. When we decrypt it, we divide the result by the same power of 16. This implies storing that power alongside the ciphertext, in plain. In order not to leak any information on the ciphertexts, we can use the same power for every encryption. Moreover, we can reduce the choice of the random permutation σ that P generates at each step to the randomness in the arg max function of standard LinUCB when several B_i are equal. Thus, the task distribution does not change the choice of the next arm to pull. We also confirmed experimentally that there is no difference between the arm-selection strategy and the outputs of LinUCB vs LinUCB-DS.

Complexity. In Fig. 5, we show the number of Paillier encryptions, decryptions, and operations on encrypted numbers. We have $O(N+d)$ encryptions, $O(NK)$ decryptions, $O(N(d^2+Kd))$ additions and $O(N(d^2+Kd))$ multiplications.

4 Security Analysis

In this section, we take a close look at what each participant knows and does not know, and we formally show the security properties of LinUCB-DS.

• DC knows, at the end of LinUCB-DS, the cumulative reward for which she paid. DC does not take part in the cumulative reward maximization algorithm.

[7] `https://python-paillier.readthedocs.io/en/stable/_modules/phe/encoding.html`

• P knows which arm is pulled at each round, this is why it can update A in plain. Since P sees θ and the rewards encrypted, it cannot see in plain the value of any among s, b, $\widehat{\theta}$, B_i, hence it cannot learn θ nor the sum of rewards.

• Comp decrypts all B_i, but sees these values in a permuted order hence it cannot associate an arm x_i with its value B_i. Since Comp does not know θ, then every arm could have possibly produced every B_i with some θ, hence Comp cannot compute the exploration term of a B_i, hopping to retrieve the rewards generated by some arm.

• An *external network observer* has access to the exchanged data shown in Fig. 3. It sees in plain N and the arms, and at each round $\sigma(listB^*)$ as well as $\sigma(m)$ the index of the maximal element in the list. As σ is changed every round, it cannot deduce the arm that is really pulled. Moreover, it cannot retrieve s or θ because $\sigma(listB^*)$ and s are encrypted.

In the rest of this section, we formally state the security properties of P, of an external observer, and of Comp. We formally prove all these properties in Appendix A. Recall that we have presented the security hypothesis in Sect. 3. In particular, we assume that the cloud nodes Comp and P do not collude. For a participant E, we denote by $data_{\mathsf{E}}$ the data to which E has access. By $\mathcal{A}^{pb}(d)$ we denote the answer of a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ that knows data d and tries to solve problem pb. We recall that by $[\![K]\!]$ we denote the set $\{1, 2, \ldots, K\}$. By $\mathrm{negl}(\lambda)$ we denote any negligible function in λ.

Security of P. The data to which P has access is θ^*, then at each round t: the arm pulled, r^*, b^*, $\widehat{\theta}^*$, the matrix A, and s^*. At the end, P also knows $\mathcal{E}_{\mathsf{DC}}(s)$.

Theorem 1. *An honest-but-curious* P *cannot infer any coordinate* $\theta(i)$ *of the secret* θ *with probability better than random. More precisely, for all PPT adversary* $\mathcal{A}$, $|P\left[(i, \theta(i)') \leftarrow \mathcal{A}^{\theta}(data_{\mathsf{P}}); \theta(i)' = \theta(i)\right] - \frac{1}{|\theta(i)|}| \leq negl(\lambda)$, *with* $\theta(i)'$ *the guess of* $\mathcal{A}$ *of* $\theta(i)$, *and* $|\theta(i)|$ *the cardinality of the set of possible values of a coordinate.*

Theorem 2. *An honest-but-curious* P *cannot infer any reward generated during the protocol with better probability than random. More precisely, for any PPT adversary* $\mathcal{A}$, $|P\left[(t, r') \leftarrow \mathcal{A}^{r}(data_{\mathsf{P}}); r' = r\right] - \frac{1}{|r|}| = negl(\lambda)$, *with* (t, r') *the guess of* $\mathcal{A}$ *of the reward generated at round* t, *and* $|r|$ *the cardinality of the set of possible rewards for the arm chosen at round* t.

Theorem 3. *An honest-but-curious* P *cannot infer cumulative reward* s.

Security of an External Observer. An external observer has access to the following data: at the beginning θ^*, the arms and the budget N; at each round, $\sigma(listB^*)$ and the argmax of the list; at the end, $\mathcal{E}_{\mathsf{Comp}}(rand + s)$, $\mathcal{E}_{\mathsf{DC}}(rand + s)$, and then $\mathcal{E}_{\mathsf{DC}}(s)$.

Theorem 4. *An external observer having access to the set* $\mathcal{M}$ *of all the messages exchanged during the protocol cannot infer the value of any coordinate of* θ *with better probability than random. More precisely, for any PPT adversary* $\mathcal{A}$,

$|P\left[(i, \theta(i)') \leftarrow \mathcal{A}^{\theta}(M); \theta(i)' = \theta(i)\right] - \frac{1}{|\theta(i)|}| \leq negl(\lambda)$, *with* $\theta(i)'$ *the guess of* $\mathcal{A}$ *of* $\theta(i)$, *and* $|\theta(i)|$ *the cardinality of the set of possible values of a coordinate.*

Theorem 5. *An external observer having access to the set* $\mathcal{M}$ *of all messages exchanged during the protocol cannot infer the value of the sum of rewards with better probability than random. More precisely, for any PPT adversary* $\mathcal{A}$, $|P\left[s' \leftarrow \mathcal{A}^{s}(\mathcal{M}); s' = s\right] - \frac{1}{|s|}| \leq negl(\lambda)$, *with* s' *the guess of* $\mathcal{A}$ *of the sum of rewards, and* $|s|$ *the cardinality of the set of possible sums at the end of the protocol.*

Lemma 1. *Consider a list* $l = [l_1, \ldots, l_n]$, *a random permutation* σ *and the permuted list* $\sigma(l) = [l_{\sigma(1)}, \ldots, l_{\sigma(n)}]$. *Knowing* $\sigma(l)$, *a PPT adversary* $\mathcal{A}$ *cannot guess one element of* l *with probability better than random. More specifically,* $P\left[(i, g(i)) \leftarrow \mathcal{A}^{\sigma^{-1}}(\sigma(l)) \in \{i, \sigma^{-1}(i)\}_{i \in [\![K]\!]}\right] = \frac{1}{K} + negl(\lambda)$, *where* $g(i)$ *is* $\mathcal{A}$*'s guess for the preimage of the element in position* i.

Theorem 6. *An external observer having access to the set* $\mathcal{M}$ *of all messages exchanged during the protocol cannot infer the arm pulled at any round. More precisely for any PPT adversary* $\mathcal{A}$, $P\left[(t, x'_t) \leftarrow \mathcal{A}^{x}(\mathcal{M}); x'_t = x_t\right] = \frac{1}{K} + negl(\lambda)$, *with* x'_t *being* $\mathcal{A}$*'s guess of the arm pulled at round* t.

Security of Comp. Comp can decrypt the elements received from P, hence the data to which it has access is: at each round, a permuted list $\sigma(listB)$ of all B_i, and at the end the value $rand + s$.

Theorem 7. *An honest-but-curious* Comp *cannot associate an element of* $\sigma(listB)$ *to the arm to which it belongs. More precisely, for any PPT adversary* $\mathcal{A}$, $P\left[(i, B'_i) \leftarrow \mathcal{A}^{\sigma^{-1}}(data_{\mathsf{Comp}}); B'_i = B_i\right] = \frac{1}{K} + negl(\lambda)$.

Theorem 8. *An honest-but-curious* Comp *cannot infer cumulative reward* s.

5 Experiments

We present a proof-of-concept experimental study that confirms the theoretical analysis, and shows the scalability and feasibility of LinUCB-DS. For reproducibility reasons, we make our code available on a public Git repository[8].

Experimental Setup. We implemented LinUCB-DS in Python 3 and did our experiments on a laptop with CPU Intel Core i5-8350U @ 1.70GHz and 16GB RAM, running Ubuntu 18.04.5. For Paillier we used the *phe* library[9].

[8] `https://github.com/anatole33/LinUCB-secure`
[9] `https://python-paillier.readthedocs.io/en/develop/`

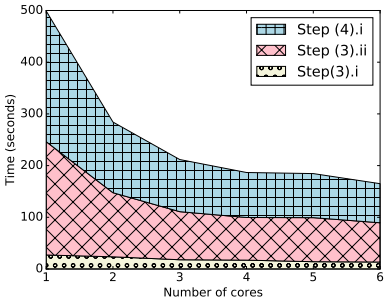
MovieLens Dataset. All our experiments are done on real data using the $100K$ MovieLens dataset [10]. This dataset is a collection of $100K$ movie ratings on a scale of 1 to 5, given by 943 users of the MovieLens website on 1682 movies. The collection of ratings is represented by a matrix F (943×1682), whose element (i, j) is the rating of user i on movie j if the rating exists, otherwise the element is 0. Since the user-movie matrix F is very sparse, we factored it using low-rank matrix factorization. To this purpose, we used the Google Colab matrix factorization code[10] and we obtained: a user embedding matrix U $(943 \times d)$, where row i is the embedding for user i, and a movie embedding matrix M $(1682 \times d)$, where row j is the embedding for movie j. The embeddings are learned such that the product $UM^\top$ is a good approximation of the ratings matrix F. Note that the (i, j) entry of $UM^\top$ is the dot product of the embeddings of user i and movie j, computed such that it should be close to the (i, j) entry of F. Then, for every user i in matrix U, we were able to use linear bandit algorithms to recommend movies j from matrix M. In the presentation of the experimental results, the reported d values correspond to choices of d in the aforementioned matrix factorization approach, whereas the reported K arms correspond to choosing the first K movies in the dataset. We set algorithm constants as in a standard related work setting [17]: $\gamma = 0.01, \delta = 0.001, R = 0.01$, and $S = \log t$.

Before discussing our experimental results, we would like to stress that for each run of LinUCB-DS we use exactly the same arm-selection strategy and obtain the same cumulative reward as LinUCB. The focus of our experiments is on the study of the feasibility and scalability of LinUCB-DS.

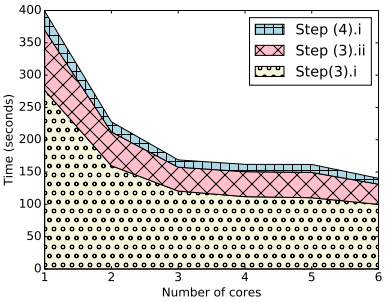
Experimental Results. As outlined in the theoretical complexity analysis at the end of Sect. 3, LinUCB-DS has an inherent overhead due to the use of cryptographic operations w.r.t. standard LinUCB. Our first implementation naturally showed this overhead. For example, for $d = 3$, $K = 15$, $N = 1000$, and Paillier keys of 1024 bits, LinUCB-DS takes 115 seconds, whereas LinUCB takes less than a second. Seen this overhead, we zoomed on the time taken by the different steps of LinUCB-DS to understand how we can optimize our implementation. We observed that three steps of LinUCB-DS take the lion's share of the computation time. We refer to these steps using the numbers listed in Sect. 3:

- Step (3).i (done by P): compute $\widehat{\theta}^*$ as the product of a matrix of dimension $(d \times d)$ and a vector of size $(d \times 1)$. This involves d^2 multiplications and d^2 additions on cyphertexts.
- Step (3).ii (done by P): compute B_i^* in the encrypted domain as the scalar product of two vectors of size d, which is done K times as there is a B_i^*-value for each arm.
- Step (4).i (done by Comp): decrypt the list of B_i^*, which takes K decryptions. The time of a decryption is higher than the time of an addition or a multiplication.

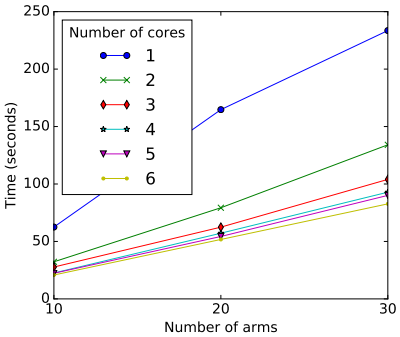
[10] `https://github.com/google/eng-edu/blob/master/ml/recommendation-systems/recommendation-systems.ipynb`



(a) $K = 50$, $d = 5$, and $N = 200$.



(b) $K = 6$, $d = 18$, and $N = 200$.



(c) K varies, $d = \frac{K}{10}$, and $N = 200$.

Fig. 6. Computation time of LinUCB-DS split on the three costliest steps, with different parameters, when increasing the number of cores (6(a) and 6(b)), and the scalability of LinUCB-DS when increasing K and d (6(c)).

Each of the aforementioned three steps is done $N - 1$ times. Fortunately, these steps are parallelizable. For instance, (3).ii and (4).i can be equivalently computed by splitting the list and parallelizing the computations. In (3).i, a coordinate of $\widehat{\theta}^*$ is obtained as the scalar product of a row of matrix A^{-1} and the vector b^*. We can divide the matrix and compute the coordinates of $\widehat{\theta}^*$ in parallel. We used the *multiprocessing*[11] library to implement a parallel version of LinUCB-DS that takes advantage of these ideas for parallelizing our code.

In Fig. 6(a) and 6(b), we present the *speedup of parallelization* on LinUCB-DS computation time, while zooming on the three aforementioned costliest steps (the other steps take negligible time), using two distinct input configurations. We used Paillier keys of 2048 bits. We believe that these figures are sufficient to show that our implementation correctly follows the theoretical expectations. Indeed, the computation of $\widehat{\theta}^*$ depends only on d, the decryption of the list of B_i^* only on K,

[11] `https://docs.python.org/3/library/multiprocessing.html`

and the construction of the list of B_i^* on both. Moreover, the computation time decreases when increasing the number of cores, which is a desirable feature as our implementation is able to take advantage of modern multi-core architectures in order to reduce the practical overhead due to cryptographic primitives.

In Fig. 6(c), we *stress test the scalability* of LinUCB-DS, in a scenario where K is 10 times larger than d. Indeed, in stochastic linear bandits the goal is to exploit the linear structure and reduce the number of needed estimations, from the estimation of K arm values to the estimation of the d features of the common unknown parameter θ. The observed computation time confirms our theoretical analysis. Moreover, we showed that the parallelization leads to a significant reduction in the computation time of LinUCB-DS.

6 Adaptability of LinUCB-DS

We show that LinUCB-DS can be easily adapted to secure SpectralUCB [17], an algorithm that models with linear bandits the problem of cumulative reward maximization on a graph. The arms are the graph nodes and the reward of an arm is a smooth function on the graph. A smooth graph function returns similar values for close nodes. When the graph models a social network, such a setting is useful for recommendation systems, since we expect that people close on the graph have similar tastes and probably like the same recommended items.

To give the right input to SpectralUCB, some preprocessing is necessary. A matrix of similarities (edge weights) of the graph is used to construct a graph Laplacian $\mathcal{L}$ that is a $(K \times K)$ matrix. Then, SpectralUCB computes the eigen-decomposition of $\mathcal{L}$ as $\mathcal{Q}\Lambda_{\mathcal{L}}\mathcal{Q}^\top$, with $\mathcal{Q}$ a $(K \times K)$ orthogonal matrix whose columns are the eigenvectors and $\Lambda_{\mathcal{L}}$ is a diagonal matrix whose elements are the corresponding eigenvalues. An arm is a row of $\mathcal{Q}$, and the expected reward value of arm q_i is given by $\langle q_i, \theta \rangle$, with θ the parameter of the smooth function, a $(K \times 1)$ vector. Note that this implies that in SpectralUCB the dimension of the vectors is equal to the number of arms ($K = d$).

As for LinUCB, when pulling an arm q_i, in SpectralUCB, one observes a noisy reward $\langle q_i, \theta \rangle + \eta$, where θ is the unknown parameter and the noise η is an R-sub-Gaussian random variable. To compute an estimation of θ, at each round t, SpectralUCB uses the arms previously pulled, joined in a matrix A_S of dimension $(K \times (t-1))$ and the rewards previously observed in a vector b_S of dimension $((t-1) \times 1)$. Then, SpectralUCB computes the estimate of the unknown parameter as $\widehat{\theta}_S = (A_S + \Lambda_{\mathcal{L}} + \gamma I_d)^{-1} b_S$, where $\Lambda_{\mathcal{L}}$ is an additional *spectral penalty* for the regularized least-squares estimate. As in LinUCB, to decide the next arm to be pulled, SpectralUCB relies on updated UCB on the arm-values and picks the arm with the largest UCB. Differently from LinUCB, the exploration term in SpectralUCB uses the *effective dimension* d' that depends on the eigenvalues and is small when eigenvalues grow rapidly above t, which is the case when $d = K >> t$. Specifically, the exploration parameter of SpectralUCB is given by $\omega_S = 2R\sqrt{d' \log(1 + t/\gamma) + 2\log(1/\delta)} + C$, where C is an upper-bound on $||\theta||_{\Lambda_{\mathcal{L}}}$. The UCB for an arm q_i is given in SpectralUCB by $B_{i,S} = \langle q_i, \widehat{\theta}_S \rangle + \omega_S ||q_i||_{A_S^{-1}}$.

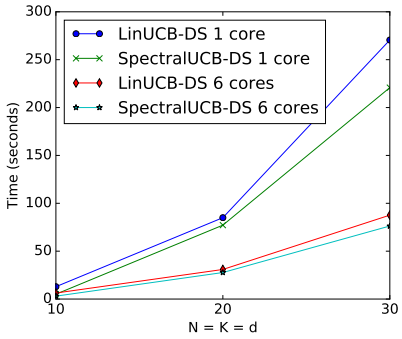


Fig. 7. Time of LinUCB-DS vs SpectralUCB-DS.

Given the similarities between LinUCB and SpectralUCB, we observed that it is not difficult to adapt the ideas behind LinUCB-DS to secure SpectralUCB. Encrypting θ results in generating encrypted rewards, and constructing vector b_s with encrypted values. Then, $\widehat{\theta}_S$ and all $B_{i,S}$ are also encrypted. The messages exchanged during the protocol are identical as for LinUCB-DS (cf. Fig. 3), in particular Comp chooses the next arm to pull. By SpectralUCB-DS we denote the secure and distributed version of SpectralUCB.

In SpectralUCB, the dimension of the vectors is equal to the number of arms: $K = d$ and for our proof-of-concept experiment (reported in Fig. 7), we fixed $K = N = d$. Following the setting in [17], we used a similarity graph over movies from the MovieLens dataset: the graph contains an edge between movies i and j if the movie j is among the 10 nearest neighbors of the movie i in the latent space M. As in [17], the weight on all edges is 1 and parameters' values are: $\gamma = 0.01, \delta = 0.001, R = 0.01$, $C = \log t$. As expected, SpectralUCB-DS is slightly faster than LinUCB-DS because it manipulates a $\widehat{\theta}$ computed with a matrix of size depending on t, and $t \leq K$. We have also zoomed on the time taken by each step of SpectralUCB-DS to observe that the three costliest steps are the same as for LinUCB-DS, and we have also observed that the parallelization technique described for LinUCB-DS has a similar positive impact on SpectralUCB-DS.

7 Conclusions

We tackled the problem of *secure cumulative reward maximization in linear stochastic bandits*. This problem has applications in recommendation systems and Web-targeted advertisements, where sensitive user data and preferences are used for personalized recommendations. We considered a *machine learning as a service* scenario, where data and computations are outsourced to some honest-but-curious cloud, which yields inherent security concerns. We proposed LinUCB-DS, a distributed and secure protocol that outputs exactly the same cumulative reward as standard LinUCB, while enjoying desirable security prop-

erties. Towards this goal, we relied on Paillier encryption scheme and secure multi-party computation. We characterized the overhead of cryptography from both theoretical and empirical points of view. Our experiments on the MovieLens movie recommendation dataset showed the scalability and feasibility of LinUCB-DS. Moreover, we showed that LinUCB-DS can be easily adapted to secure other UCB-like linear bandit algorithms. This happens because the security properties of our protocol hold true irrespective of the arm-selection strategy, which differs from an algorithm to another. To show this, we adapted LinUCB-DS to secure SpectralUCB.

Providing security guarantees for machine learning algorithms is a growing research topic. The use of distribution of tasks and cryptography is still an underexplored research direction for this task. We plan to rely on such techniques to develop further security protocols for other types of bandit algorithms and for different machine learning settings.

A Appendix: Security Proofs for Sect. 4

Proof of Theorem 1. Assume a PPT adversary $\mathcal{A}$ who, given $data_{\mathsf{P}}$ has a probability of $\frac{1}{|\theta(i)|} + x + \text{negl}(\lambda)$ of guessing one coordinate of θ. In the worst case, it makes a guess on each coordinate with the same probability $\frac{1}{d}$. We also assume that if $data_{\mathsf{P}}$ is different from the data P has really collected during the protocol (for instance if a value has been changed to another unrelated to the protocol), then $\mathcal{A}$ has not any advantage. We show that using $\mathcal{A}$, an adversary $\mathcal{B}$ obtains an advantage non-negligible in a Paillier IND-CPA game.

$\mathcal{B}$ chooses two values m_0 and m_1 and gives them to a challenger who returns $m_b^* = \mathcal{E}_{\mathsf{Comp}}(m_b)$, with $b = 0$ or 1 with probability $\frac{1}{2}$. Then $\mathcal{B}$ constructs an execution of the secure protocol, with θ and arms of his choice. In particular, it sets $\theta_1 = m_1$. At the end, it calls $\mathcal{A}$ on $data_{\mathsf{P}}$ except that it replaces θ'_1 by m_b^*. Let us call it $data'_{\mathsf{P}}$. If $\mathcal{A}^{\theta}(data'_{\mathsf{P}})$ returns $(1, m_1)$, then $\mathcal{B}$ answers 1 to the challenger, otherwise it answers at random 0 or 1 with probability $\frac{1}{2}$.

The probability of success of $\mathcal{B}$ in every situation is:

- In $\mathcal{A}$'s guess, if $i \neq 1$ (with probability $1 - \frac{1}{d}$), then $\mathcal{B}$ answers at random and his probability of success is $\frac{1}{2}$.
- If $i = 1$ (with probability $\frac{1}{d}$):
  - If $b = 0$ (with probability $\frac{1}{2}$) then $data'_{\mathsf{P}}$ is not valid and $\mathcal{A}$ has not any advantage.
    - It answers $(1, m_1)$ with probability $\frac{1}{|\theta_1|}$, where $\mathcal{B}$ answers 1 to the IND-CPA game and is wrong.
    - It gives an other value for θ'_1 with probability $1 - \frac{1}{|\theta_1|}$, then $\mathcal{B}$ answers at random and has a probability of success of $\frac{1}{2}$.
  - If $b = 1$ (with probability $\frac{1}{2}$) then $\mathcal{A}$ benefits of its advantage.
    - By hypothesis, $\mathcal{A}$ returns $(1, m_1)$ with probability $\frac{1}{|\theta_1|} + x + \text{negl}(\lambda)$. Then $\mathcal{B}$ trusts him and is right.

* By hypothesis, $\mathcal{A}$ returns another value for θ_1' with probability $1 - (\frac{1}{|\theta_1|} + x + \text{negl}(\lambda))$. $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$.

Summing it up, the probability of success of $\mathcal{B}$ in his IND-CPA game is: $P(\mathcal{B}) = (1 - \frac{1}{d})\frac{1}{2} + \frac{1}{d}\frac{1}{2}(1 - \frac{1}{|\theta_1|})\frac{1}{2} + \frac{1}{d}\frac{1}{2}(\frac{1}{|\theta_1|} + x + \text{negl}(\lambda)) + \frac{1}{d}\frac{1}{2}(1 - \frac{1}{|\theta_1|} - x - \text{negl}(\lambda))\frac{1}{2}\frac{1}{2} - \frac{1}{2d} + \frac{1}{4d} - \frac{1}{4d|\theta_1|} + \frac{1}{2d|\theta_1|} + \frac{1}{2d}x + \frac{1}{4d} - \frac{1}{4d|\theta_1|} - \frac{1}{4d}x + \text{negl}(\lambda) = \frac{1}{2} + \frac{1}{4d}x + \text{negl}(\lambda)$. It gives him a non-negligible advantage in a classical IND-CPA game on Paillier scheme, which contradicts the fact that Paillier is IND-CPA secure. Then our assumption was wrong and an adversary who has an advantage in retrieving a coordinate of θ with $data_{\mathsf{P}}$ cannot exist.

Proof of Theorem 2. The same proof as above can be applied, with $\mathcal{B}$ changing one of the rewards with m_1^* after the execution of the protocol. It yields to $\mathcal{B}$ an advantage of $\frac{1}{2} + \frac{1}{4N}x + \text{negl}(\lambda)$ to an IND-CPA game (with N the budget and the number of pulls) which is impossible if Paillier is IND-CPA secure.

Proof of Theorem 3. Let $\mathcal{A}$ be a PPT adversary trying to retrieve the cumulative sum of rewards and $\mathcal{B}$ an adversary trying to retrieve any of the N rewards generated. $\mathcal{A}^s(data)$ has a non-negligible advantage $\Leftrightarrow \mathcal{B}^r(data)$ has a non-negligible advantage.

$\Leftarrow \mathcal{A}$ can call $\mathcal{B}$ and obtains the correct value of one reward with probability non-negligible. It gives him a lower bound on the sum of all rewards, and consequently reduces the possibilities of s. It now has a better probability than random to guess s.

$\Rightarrow \mathcal{B}$ calls $\mathcal{A}$ and obtains the correct value of s with a non-negligible probability. It is an upper bound on the value of one reward, and reduces the possibilities of all r_t.

The bounds do not reduce significantly the space of possibilities if N is big but N can very well be small, even 1. In any case, the advantage one benefits from the other is non-negligible.

This ensures that P cannot retrieve s, because it would give him an advantage in retrieving one of the rewards, and it has been proven impossible.

Proof of Theorem 4. Same proof as for Theorem 1 applies.

Proof of Theorem 5. Assume a PPT adversary $\mathcal{A}$ who, given $\mathcal{M}$, has a probability of guessing the correct s with probability $\frac{1}{|s|} + x + \text{negl}(\lambda)$. Then we show how an adversary $\mathcal{B}$ can use $\mathcal{A}$ to gain a non-negligible advantage in a Paillier IND-CPA game. Again, we assume that if $\mathcal{M}$ is changed with a value unrelated to the protocol, then $\mathcal{A}$ does not conserve its advantage. $\mathcal{B}$ simulates the execution of the protocol with θ and arms of his choice. It knows the value of s at the end. He then chooses s as m_1 for the IND-CPA challenge, and a value out of the set of possible s for m_0. It gives m_0 and m_1 to the challenger who returns $\mathcal{E}_{\mathsf{DC}}(m_b)$ with $b = 0$ or 1 with probability $\frac{1}{2}$. $\mathcal{B}$ takes the set $\mathcal{M}$ of all messages exchanged by the nodes for the protocol, and replaces $\mathcal{E}_{\mathsf{DC}}(s)$ with $\mathcal{E}_{\mathsf{DC}}(m_b)$. It calls $\mathcal{A}^s(\mathcal{M}')$

and observes the output. If it is s, it answers 1 to the decisional challenge. Else, it answers at random 0 or 1 with probability $\frac{1}{2}$.

- $b = 0$ (with probability $\frac{1}{2}$)
  - $\mathcal{A}$ returns s with probability $\frac{1}{|s|}$ and $\mathcal{B}$ is wrong.
  - $\mathcal{A}$ returns something else with probability $1 - \frac{1}{|s|}$, $\mathcal{B}$ answers at random and is right with probability $\frac{1}{2}$.
- $b = 1$ (with probability $\frac{1}{2}$)
  - $\mathcal{A}$ returns s with probability $\frac{1}{|s|} + x + \text{negl}(\lambda)$, and $\mathcal{B}$ answers 1 and is right.
  - $\mathcal{A}$ returns something else with probability $1 - (\frac{1}{|s|} + x + \text{negl}(\lambda))$, $\mathcal{B}$ answers at random and is right with probability $\frac{1}{2}$.

In total, $\mathcal{B}$ answers correctly the challenge with probability $P(\mathcal{B}) = \frac{1}{2}(1 - \frac{1}{|s|})\frac{1}{2} + \frac{1}{2}(\frac{1}{|s|} + x + \text{negl}(\lambda)) + \frac{1}{2}(1 - \frac{1}{|s|} - x - \text{negl}(\lambda))\frac{1}{2} = \frac{1}{4} - \frac{1}{4|s|} + \frac{1}{2|s|} + \frac{1}{2}x + \frac{1}{4} - \frac{1}{4|s|} - \frac{1}{4}x + \text{negl}(\lambda) = \frac{1}{2} + \frac{1}{4}x + \text{negl}(\lambda)$. He has gain a non-negligible advantage in the Paillier IND-CPA game, which is impossible. Thus, such an adversary $\mathcal{A}$ cannot exist.

Proof of Lemma 1. This is immediate, as all images/preimages are equally likely if σ is uniformly selected.

Proof of Theorem 6. Assume an adversary $\mathcal{A}^x(\mathcal{M})$ who has a probability $\frac{1}{K} + x + \text{negl}(\lambda)$ of guessing the correct arm pulled at round t. It also knows the index of the maximal element in the permuted list of B_i that P and Comp exchange at round t. That index is the permuted index of the arm who is really pulled, x_t. It means that $\mathcal{A}$ can make a guess on an element of the permutation σ_t and is right with the same probability as it is right at guessing the arm pulled at round t, and it benefits of the same non-negligible advantage. But $\mathcal{A}$ does not know σ and should only have a probability of $\frac{1}{K}$ of guessing an element of σ according to Lemma 1. This is a contradiction, so $\mathcal{A}$ cannot exist.

Proof of Theorem 7. Assume a PPT adversary $\mathcal{A}$ who, given $data_{\mathsf{Comp}}$ is able to retrieve the value B_i of arm i with probability $\frac{1}{K} + x + \text{negl}(\lambda)$. After making his guess B'_i, he can look in $\sigma(listB)$ the position of B'_i and make a guess on the value of $\sigma(i)$. It benefits of the same non-negligible advantage in guessing an element of the permutation σ on which it has no information. This contradicts Lemma 1, thus such an adversary cannot exist.

Proof of Theorem 8. For a fixed s, if the random number *rand* is uniformly chosen, then $rand + s$ can take all possible values with the same probability. Hence when Comp sees $rand + s$, it gains no information on s.

References

1. Abbasi-Yadkori, Y., Pál, D., Szepesvári, C.: Improved Algorithms for Linear Stochastic Bandits. In: NIPS. pp. 2312–2320 (2011)

2. Auer, P.: Using Confidence Bounds for Exploitation-Exploration Trade-offs. JMLR **3**, 397–422 (2002)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-Time Analysis of the Multiarmed Bandit Problem. Machine Learning **47**(2-3), 235–256 (2002)
4. Baldimtsi, F., Ohrimenko, O.: Sorting and Searching Behind the Curtain. In: Financial Cryptography. pp. 127–146 (2015)
5. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In: CRYPTO. pp. 483–512 (2018)
6. Ciucanu, R., Lafourcade, P., Lombard-Platet, M., Soare, M.: Secure Best Arm Identification in Multi-Armed Bandits. In: ISPEC. pp. 152–171 (2019)
7. Gajane, P., Urvoy, T., Kaufmann, E.: Corrupt Bandits for Preserving Local Privacy. In: ALT. pp. 387–412 (2018)
8. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: STOC. pp. 169–178 (2009)
9. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004)
10. Harper, F.M., Konstan, J.A.: The MovieLens Datasets: History and Context. ACM TiiS **5**(4), 19:1–19:19 (2016)
11. Lattimore, T., Szepesvári, C.: Bandit Algorithms. Cambridge University Press (2020), `https://tor-lattimore.com/downloads/book/book.pdf`
12. Li, L., Chu, W., Langford, J., Schapire, R.E.: A Contextual-bandit Approach to Personalized News Article Recommendation. In: WWW. pp. 661–670 (2010)
13. Mishra, N., Thakurta, A.: (Nearly) Optimal Differentially Private Stochastic Multi-Arm Bandits. In: UAI. pp. 592–601 (2015)
14. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: EUROCRYPT. pp. 223–238 (1999)
15. Shariff, R., Sheffet, O.: Differentially Private Contextual Linear Bandits. In: NeurIPS. pp. 4301–4311 (2018)
16. Tossou, A.C.Y., Dimitrakakis, C.: Algorithms for Differentially Private Multi-Armed Bandits. In: AAAI. pp. 2087–2093 (2016)
17. Valko, M., Munos, R., Kveton, B., Kocák, T.: Spectral Bandits for Smooth Graph Functions. In: ICML. pp. 46–54 (2014)