Secure Outsourcing of Multi-Armed Bandits

Radu Ciucanu
INSA Centre Val de Loire, LIFO
radu.ciucanu@insa-cvl.fr

Pascal Lafourcade
Univ. Clermont Auvergne, LIMOS
pascal.lafourcade@uca.fr

Marius Lombard-Platet
Univ. PSL, DIENS, Be-Studys
marius.lombard-platet@ens.fr

Marta Soare
Univ. Orléans, LIFO
marta.soare@univ-orleans.fr

***Abstract*—We consider the problem of cumulative reward maximization in multi-armed bandits. We address the security concerns that occur when data and computations are outsourced to an honest-but-curious cloud i.e., that executes tasks dutifully, but tries to gain as much information as possible. We consider situations where data used in bandit algorithms is sensitive and has to be protected e.g., commercial or personal data. We rely on cryptographic schemes and propose UCB-DS, a distributed and secure protocol based on the UCB algorithm. We prove that UCB-DS computes the same cumulative reward as UCB while satisfying desirable security properties. In particular, cloud nodes cannot learn the cumulative reward or the sum of rewards for more than one arm. Moreover, by analyzing messages exchanged among cloud nodes, an external observer cannot learn the cumulative reward or the sum of rewards produced by some arm. We show that the overhead due to cryptographic primitives is linear in the size of the input. Our implementation confirms the linear-time behavior and the practical feasibility of our protocol, on both synthetic and real-world data.**

I. Introduction

The *stochastic multi-armed bandit* game is a sequential learning framework where a learning agent aims at maximizing its *cumulative reward* while successively interacting with an uncertain environment. At each time step, the agent chooses an action *(a bandit arm)* from a fixed set of actions with unknown associated values. The environment responds with a stochastic feedback *(reward)* drawn from the distribution associated with the chosen action. The agent uses the received feedback to update its estimate of the values for the chosen action and to decide which action to choose next. The agent has to continuously face the so-called exploration-exploitation dilemma and decide whether to *explore* by choosing actions with more uncertain associated values, or to *exploit* the information already acquired by choosing the action with the seemingly largest associated value. Cumulative reward maximization has been already extensively studied for several multi-armed bandit settings (see [1] for a survey) and for various applications, from clinical trials, to online advertising and recommendation systems. In this paper, we address the security concerns that occur when outsourcing the cumulative reward maximization data and computations to the cloud.

Our scenario is inspired by the *machine learning as a service* cloud computing model, for which security is known as a major concern [2]. As a motivating example, assume:

• A *data owner*: a company that wants to monetize some collected data, while keeping ownership over it. The collected data may be a large quantity of surveys on customer preferences for several products. By product, we mean any type of object or service. The K bandit arms are the surveyed products and only the data owner knows their associated rewards, based on the collected surveys.

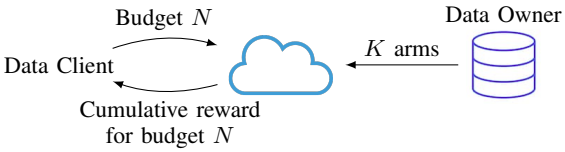


Fig. 1. Outsourcing data and computations.

• A *data client*: a company that wants to spend some budget to use some of the data owner's data. The data client may be a small company that cannot afford doing its own surveys, but wants to estimate the income that it could generate for the products surveyed by the data owner. The cumulative reward captures such information because it sums the rewards produced by each product. The budget N is the number of data owner's surveys used to compute the cumulative reward and the bandit algorithm has to decide *how* to choose these N surveys in order to maximize the cumulative reward. A larger budget gives a higher accuracy for the largest cumulative reward. The data client only sees the cumulative reward, without knowing the values associated to each arm.

We assume that the interaction between the data owner and the data client is done using the *cloud* (as shown in Fig. 1), where both data and computations are *outsourced*. The data owner does the data outsourcing, and the data client interacts directly with the cloud, by sending the budget and receiving the obtained cumulative reward. The outsourced data may be sensitive (e.g., personal, commercial, or medical data). We want the outsourced learning algorithm to be run while protecting data against unauthorized access. The problem that we address is how to allow the data client to obtain precisely the same cumulative reward as with a standard bandit algorithm, *Upper Confidence Bound (UCB)* [1], [3], within a reasonable computation time and while preserving the data security. Indeed, the outsourced data can be communicated over an untrustworthy network and processed on some untrustworthy machines, where malicious cloud users may learn sensitive data that belongs only to the data owner.

The privacy-preserving cumulative reward maximization is a hard problem. To solve it, the authors of [4], [5], [6] use *differential privacy* introduced in [7]. However, in these approaches, the returned reward is not the same reward obtained for the data client's budget using standard algorithms. This happens because differential privacy guarantees depend on noise be-

2324-9013/20/$31.00 ©2020 IEEE
DOI 10.1109/TrustCom50675.2020.00038

ing injected in the input/output. We take a complementary approach by relying on *cryptography* instead of differential privacy. To the best of our knowledge, our approach is original and its goal is to give security guarantees, while obtaining the same output as standard (non-secure) algorithms. The security for obtaining the same output has a price because the computation time may increase because of cryptographic primitives that are time-consuming in practice. More precisely, we require that the data owner (which can be seen as an oracle knowing the reward functions associated with each arm) encrypts her data before outsourcing it to the cloud. Then, the cumulative reward maximization algorithm is run directly in the encrypted domain, and the (encrypted) output should be exactly the same as for standard UCB, with the cost of an increased computation time.

From a theoretical point of view, the problem could be straightforwardly 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. Indeed, the state-of-the-art fully homomorphic systems (SEAL[1] and HElib[2]) yield only approximate results when they work with real numbers, by using the CKKS scheme [9]. Hence, it is not currently possible to program an algorithm such as UCB in a fully homomorphic system and obtain exactly the same result as in the standard, non-encrypted UCB.

Consequently, our challenge is to rely on simpler cryptographic schemes and design a distributed protocol with several cloud node participants such that each of them can only learn the specific data needed for performing its task and nothing else e.g., if a participant does in clear computations on real numbers, these computations concern data of only one arm, and no other participant has access to this piece of data. Our distributed algorithm returns exactly the same cumulative reward as UCB, while satisfying desirable security properties such as: only the data client can see the cumulative reward, which cannot be learned by any cloud node participant nor by an external observer. We precisely characterize our security model and security guarantees later on in the paper. To achieve our goals, we rely on *indistinguishable under chosen-plaintext attack* (IND-CPA) cryptographic schemes: symmetric encryption AES-CBC [10], [11] and asymmetric partially homomorphic Paillier's scheme [12]. We formally prove the security of our protocol and we precisely characterize the number of needed cryptographic operations.

a) Related work: Each line in Table I corresponds to a standard problem in stochastic multi-armed bandits. The most popular problem is *cumulative reward maximization* and UCB is a standard algorithm for solving it [1], [3]. There is a recent line of research on enhancing algorithms such as UCB with differential privacy [4], [5], [6]. There are some fundamental differences between this line of work and our work based on cryptography. On the one side, the running time overhead of differentially-private algorithms is negligible, whereas our approach has an overhead in computation time coming from the use of cryptographic primitives. On the other side, the cumulative reward returned by differentially-private algorithms is different from the output of standard UCB. Indeed, to obtain differentially-private guarantees for a bandit algorithm, noise is added to the algorithm input or output. Thus, the cumulative reward obtained using a differentially-private algorithm is different from that obtained by the algorithm without privacy guarantees. This is reflected in the *regret analysis* of the algorithms (where the *regret* is given by the difference in the cumulative reward obtained by a learning agent and the best cumulative reward possible obtained by always playing the best arm): the regret of differentially-private bandit algorithms have as overhead an additive [6] or multiplicative factor [4], [5] with respect to the regret of their non-private version. In contrast, our cryptography-based algorithm is guaranteed to return exactly the same cumulative reward as standard UCB.

TABLE I
SUMMARY OF RELATED WORK AND POSITIONING OF OUR CONTRIBUTION.

	Differential privacy	Cryptography
Cumulative reward maximization aka cumulative regret minimization	[4], [5], [6]	**This paper**
Best arm identification aka simple regret minimization	Not yet studied to the best of our knowledge	[13]

The second line in Table I corresponds to a different bandit problem that is best arm identification [14], equivalent to minimizing the simple regret, that is the difference between the values associated with the arm that is actually the best and the best arm identified by the algorithm. From the cryptography point of view, there exists a distributed algorithm [13] that enhances the Successive rejects algorithm [14] for best arm identification with security guarantees that are similar to the ones from this paper. Naturally, the algorithms that are secured (Successive rejects [14] in [13] and UCB [3] in this paper) solve different problems, thus the corresponding secure protocols are different and cannot be reduced to one another.

All related works discussed thus far are for standard stochastic bandit models. Securing cumulative reward maximization algorithms using cryptography has been recently studied for a different bandit model i.e., linear bandits [15], where the arms are vectors and the rewards are unknown linear functions of the arms. The corresponding secure protocols are again different and cannot be reduced to one another.

b) Summary of contributions and paper organization: In Sect. II, we introduce some basic notions: standard UCB algorithm and some cryptographic tools. Then, Sect. III is the core of our contribution:

- We propose UCB-DS, a secure and distributed protocol for cumulative reward maximization that guarantees the same cumulative reward as standard UCB.
- We show that UCB-DS satisfies desirable security properties that we precisely characterize.
- We analyze the theoretical complexity of UCB-DS, by

[1]https://github.com/Microsoft/SEAL
[2]http://homenc.github.io/HElib/

quantifying the number of needed cryptographic primitives: $O(NK)$ AES-CBC encryptions/decryptions, K Paillier encryptions, and one Paillier decryption.
- We propose the UCB-DS2 refinement, with stronger security guarantees at the price of K more AES-CBC keys and $O(NK)$ more AES-CBC encryptions/decryptions, and the same number of Paillier encryptions/decryptions.

In Sect. IV, we include a proof-of-concept empirical evaluation that confirms the theoretical complexity, and shows the scalability and practical feasibility of our protocols, on synthetic and real-world data. Finally, we conclude our paper and outline directions for future work in Sect. V.

II. Preliminaries

We first recall the *UCB* algorithm [3]. Then, we briefly present two cryptographic schemes that we use to build our protocols: *Paillier asymmetric encryption* scheme and *AES-CBC symmetric encryption*, which are both *IND-CPA secure*.

a) Upper Confidence Bound (UCB): is a class of algorithms commonly used when facing the exploration-exploitation dilemma. Each bandit arm is associated with a distribution whose mean is unknown to the learning agent. When pulling an arm, the agent observes an independent reward drawn from the distribution associated to the chosen arm. Specifically, we consider rewards drawn from Bernoulli distributions with expected values $\mu_1, \ldots, \mu_K$ unknown to the agent. For a chosen arm i, a call to the function $pull(i)$ randomly returns 0 or 1 according to the associated Bernoulli distribution, i.e., the probability of returning 1 is μ_i and the probability of 0 is 1–μ_i. The agent sequentially selects the N arms to be pulled with the goal of maximizing the sum of rewards.

To guide the choice of the learner, arm scores have been proposed [16] to construct *upper confidence bounds* (UCB) based on the empirical mean of arm-specific rewards and the number of arm pulls. In the class of UCB algorithms, an important breakthrough was the introduction of algorithms with a finite-time analysis [3]. Specifically, in the UCB algorithm [3] presented in Fig. 2, for each arm i, the score B_i is an upper-confidence bound on μ_i, obtained as the sum between (i) the *exploitation* term given by the empirical mean of rewards observed from arm i, and (ii) the *exploration* term, which takes into account the uncertainty. Notice that after each observed reward, scores for all arms are updated, since the *exploration* term $\sqrt{\frac{2\ln(t)}{n_i}}$ depends on the total number of rewards observed up to current round t. Thus, an arm i being pulled few times (i.e., with small n_i) will have a relatively large *exploration* term. The score B_i is thus an *optimistic* estimate for the value associated to arm i, since it can be interpreted as the largest statistically plausible mean value associated to arm i, given the observed rewards. As shown in Fig. 2, UCB chooses to pull next the arm with the largest updated B_i score, thus following the principle of *optimism in the face of uncertainty*. This principle suggests to follow what seems to be the best arm, based on the *optimistically* constructed scores. The same principle is employed in various sequential decision making problems (see [17] for a survey).

Input: Budget N, number of arms K

Unknown environment: K distributions associated to the K arms, with expected values $\mu_1, \ldots, \mu_K$ unknown to the learning agent. The agent has access to a reward function $pull(.)$ that can be called N times. A call $pull(i)$ returns a random value from the distribution associated to arm i.

Output: Sum of observed rewards for all arms

/ Initialization: pull each arm once & initialize variables */*
for $1 \leq i \leq K$
  let $r = pull(i)$ */* Random reward for arm i */*
  let $s_i = r$ */* Sum of observed rewards for arm i */*
  let $n_i = 1$ */* Number of pulls of arm i */*
  let $B_i = \frac{s_i}{n_i} + \sqrt{\frac{2\ln(K)}{n_i}}$ */* B_i is an UCB on μ_i */*
/ Exploration-exploitation: pull one arm at each round t */*
for $K+1 \leq t \leq N$ */* Only a budget of $N-K$ is left */*
  let $i_m = arg\,max_{1 \leq i \leq K}(B_i)$ */* Ties broken randomly */*
  let $r = pull(i_m)$
  let $s_{i_m} = s_{i_m} + r$
  let $n_{i_m} = n_{i_m} + 1$
  for $1 \leq i \leq K$
    let $B_i = \frac{s_i}{n_i} + \sqrt{\frac{2\ln(t)}{n_i}}$
return $s_1 + \ldots + s_K$

Fig. 2. UCB Algorithm [3].

b) Paillier asymmetric encryption: Paillier's cryptosystem is *additive homomorphic* [12]. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$. The product of the two associated ciphertexts with the public key pk, denoted $c_1 = \mathcal{E}_{\mathsf{pk}}(m_1)$ and $c_2 = \mathcal{E}_{\mathsf{pk}}(m_2)$, is the encryption of the sum of m_1 and m_2. Indeed, we have: $\mathcal{E}_{\mathsf{pk}}(m_1) \cdot \mathcal{E}_{\mathsf{pk}}(m_2) = \mathcal{E}_{\mathsf{pk}}(m_1 + m_2)$. We also denote by $\mathcal{D}_{\mathsf{sk}}(c)$ the decryption of the cipher c by the secret key sk.

c) AES-CBC symmetric encryption: AES [10] is a NIST standard for encrypting messages of 128 bits. We use it with CBC mode (Cipher Block Chaining) and denote $c = \mathsf{Enc}(m)$ the encryption of m and $m = \mathsf{Dec}(c)$ the decryption of c with the same symmetric key shared between the participants.

Both Paillier and AES-CBC are **IND-CPA**: (i) Paillier is IND-CPA under the decisional composite residuosity assumption [12], and (ii) AES-CBC is IND-CPA under the assumption that AES is a pseudo-random permutation [11]. All theoretical security properties of our protocols also hold if we choose any other IND-CPA symmetric scheme instead of AES-CBC, and any other additive homomorphic IND-CPA asymmetric scheme instead of Paillier. Our choice to rely on the aforementioned schemes is due to practical reasons. AES-CBC is very efficient in practice and implemented in standard libraries for modern programming languages. Paillier is also supported by a number of libraries that can be used in practice.

III. UCB-DS: A Distributed and Secure Protocol Based on UCB Algorithm

We define the security model in Sect. III-A. We propose our secure protocol UCB-DS (Sect. III-B), and we analyze its correctness, security, and complexity (Sect. III-C). We introduce a refinement of UCB-DS in Sect. III-D.

A. Security Model

As outlined in Introduction and in Fig. 1, we assume that the data (i.e., the reward functions associated to K bandit arms) and the computations (i.e., the cumulative reward maximization algorithm) are outsourced to an *honest-but-curious* cloud. This means that the cloud executes tasks dutifully, but tries to extract as much information as possible from the data that it sees. Our model follows the classical formulation in [18] (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 data client indicates to the cloud her budget N and receives the cumulative reward R that the cloud computes using the K arms outsourced by the data owner and the data client's budget N. The data client does not have to do any computation, except for decrypting R when the data client receives this information encrypted from the cloud. We expect the following *security properties*:

1) No cloud node can learn the cumulative reward.
2) The data client cannot learn information about the rewards produced by each arm or which arm has been pulled at some round.
3) By analyzing the messages exchanged between different cloud nodes, an external observer cannot learn the cumulative reward, the sum of rewards produced by some arm, or which arm has been pulled at some round.

We give a brief intuition for each property. Property 1 implies that only the data client can see in clear the cumulative reward for which she spends a budget. Property 2 ensures that the data client can see only the information for which she pays, and nothing else. Otherwise, depending on the difficulty of the bandit problem, the data client could estimate the arm values based on the contribution of each arm to the cumulative reward, which would leak information that should be known only by the data owner. Property 3 states that if some curious cloud admin analyzes all messages exchanged over the network, then she should have no clue on any input, output, or intermediate data that is used by the cumulative reward maximization algorithm.

We design a distributed protocol that satisfies the aforementioned properties by exchanging only encrypted messages, and by distributing the computations among several cloud node participants, each of them having access only to the specific data that it needs for performing its task and nothing else. The challenge is to efficiently distribute tasks among as few cloud participants as possible, while minimizing the time needed for cryptographic primitives.

B. Overview of UCB-DS

In Fig. 3, we present an overview of UCB-DS. There are $K+1$ cloud participants: K arm nodes R_i and a node AS (Arm Selector) that is the controller of the protocol. We assume that the data owner and all cloud participants share the same symmetric AES-CBC key, used for encryption function Enc. The data client (DC) generates a Paillier's key pair (pk, sk) and for sake of clarity we denote $\mathcal{E}_{\mathsf{DC}}(m)$ for $\mathcal{E}_{\mathsf{pk}}(m)$. By $[\![x]\!]$, we denote the set $\{1,\ldots,x\}$, and by $y||z$ we denote the concatenation of y and z. UCB-DS works as follows:

- Fig. 3(a) (steps 0 and 1). For $i \in [\![K]\!]$, the data owner outsources to arm node R_i the reward function (encrypted with Enc) associated to arm i. The data client sends to the cloud her budget N.
- Fig. 3(b) (steps 2, 3, and 4). This is the core of the protocol, being done during $1+N-K$ iterations: once for the initialization phase of UCB and $N-K$ times for the exploration-exploitation phase of UCB cf. Fig. 2. For each iteration, all arm nodes interact to decide which arm should be pulled next and communicate this information to AS. The arm nodes communicate in a random order, which changes at each iteration. All messages exchanged between nodes are encrypted with Enc. Although each arm node stores information about its rewards, it never reveals this information to other nodes.
- Fig. 3(c) (steps 5 and 6). After spending the data client's budget, each arm node sends to AS the sum of rewards that it produced, encrypted with $\mathcal{E}_{\mathsf{DC}}$. Due to the additive homomorphic property of Paillier cryptosystem, AS is able to sum up the K partial rewards to compute the cumulative reward $\mathcal{E}_{\mathsf{DC}}(R)$ directly in the encrypted domain. Only the data client can decrypt this information.

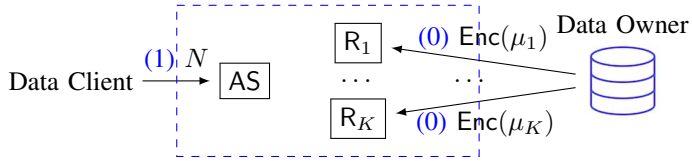
We next detail each step and present pseudocode only when the step is not trivial.

a) Step 0: We recall (cf. Fig. 2) that the data owner knows $\mu_1,\ldots,\mu_K$ defining K Bernoulli distributions associated to the K arms. The data owner sends to each arm node R_i the encrypted value $\mathsf{Enc}(\mu_i)$, for $i \in [\![K]\!]$. Since the data owner and the cloud share the symmetric key, then each arm node R_i can decrypt and obtain μ_i. Moreover, each node R_i initializes to 0 the following two variables that it later on updates during the protocol: s_i (i.e., sum of rewards for arm i) and n_i (i.e., number of times the arm i has been pulled). Additionally, each arm node R_i initializes a variable $t = K-1$, which is later on updated and needed for the computation of B_i.

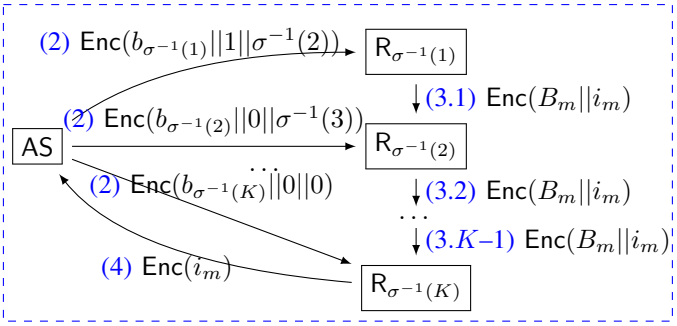
b) Step 1: The data client sends her budget N to AS.

Pseudocodes of Steps 2, 3, and 4 are presented in Fig. 4.

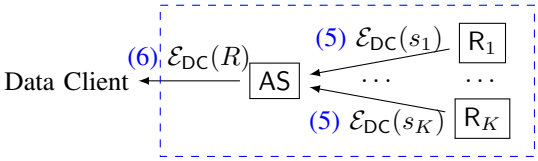
c) Step 2: It corresponds to everything except the last two lines in Fig. 4(a) and has $1+N-K$ iterations. At each iteration, AS sends to the R_i nodes a bit b_i indicating whether the arm i should be pulled or not. At the first iteration (that corresponds to the initialization phase of UCB cf. Fig. 2), AS sends $b_i = 1$ to each arm, and at the next $N-K$ iterations (that correspond to the exploration-exploitation phase of UCB cf. Fig. 2), AS sends $b_i = 1$ only to a chosen arm i_m and



(a) Steps 0 and 1 are done only once at the beginning.



(b) Steps 2, 3, and 4 are done $1 + N - K$ times.



(c) Steps 5 and 6 are done only once at the end.

Fig. 3. Overview of UCB-DS. The dashed rectangle is the cloud.

```
let i_m = 0
for j ∈ [[N − K + 1]]
    let σ=random permutation of [[K]]
    for i ∈ [[K]]
        /* b_i is a bit indicating if arm i should be pulled */
        if i_m = 0 or i_m = i then let b_i = 1 else let b_i = 0
        /* first_i is a bit indicating if i is the first arm node
in the ring cf. σ */
        if σ(i) = 1 then let first_i = 1 else let first_i = 0
        /* next_i indicates the next arm node in the ring, or
0 if i is the last cf. σ */
        if σ(i) ≠ K then let next_i = σ^{-1}(σ(i) + 1) else
let next_i = 0
        send Enc(b_i||first_i||next_i) to arm node R_i
    receive ciphertext from arm node R_{σ^{-1}(K)}
    /* ciphertext is Enc(i_m) */
    let i_m = Dec(ciphertext)
```

(a) Pseudocode of AS.

```
receive ciphertext1 from AS
/* ciphertext1 is Enc(b_i||first_i||next_i) */
let b_i||first_i||next_i = Dec(ciphertext1)
let t = t + 1
if b_i = 1          /* Pull arm i and update its variables */
    let r = pull(i)
    let s_i = s_i + r
    let n_i = n_i + 1
let B_i = s_i/n_i + sqrt(2 ln(t)/n_i)
if first_i = 0
    receive ciphertext2 from preceding arm node in ring
    /* ciphertext2 is Enc(B_m||i_m) */
    B_m||i_m = Dec(ciphertext2)
if first_i = 1 or  B_m < B_i
    let i_m = i
    let B_m = B_i
if next_i ≠ 0
    send Enc(B_m||i_m) to R_{next_i}
else
    send Enc(i_m) to AS
```

(b) Pseudocode of R_i, for $i \in [\![K]\!]$.

Fig. 4. Pseudocode of AS and R_i during steps 2, 3, and 4 cf. Fig. 3(b).

sends $b_i = 0$ to all other arms. Moreover, at each iteration, AS generates a permutation $\sigma : [\![K]\!] \to [\![K]\!]$ (i.e., a function for which every element occurs exactly once as an image value), based on which AS computes two more components that it sends to R_i: $first_i$ that indicates whether the arm node is the first of the ring hence it should initialize B_m and i_m, and $next_i$ that indicates to which node the updated B_m and i_m should be sent during Step 3. The arm node that receives 0 on the $next$ component is the last one of the ring and sends i_m to AS, which thus knows which arm should be pulled next. All information that AS sends to R_i are thus useful for the ring computation of i_m in Step 3. The permutation changes at each AS iteration because it is important to have a random order during the ring communication. Without a random order, it may happen that the last arm is much better than all others and it is almost always pulled, hence it has a very good estimate of the cumulative reward.

d) Step 3: This step corresponds to everything except the last two lines in Fig. 4(b). Note that the variable t stores how many arm pulls have been done in total since the beginning of the protocol. As discussed for Step 0, each arm initialized $t = K - 1$, hence $t = K$ after the first iteration of AS, which allows to compute the first B_i values at the end of the initialization phase. Then, during the next $N - K$ iterations of AS, the variable t is incremented, which allows to compute B_i values during the exploration-exploitation phase. To decide which arm has the highest B_i and should be pulled at the next iteration, the arm nodes R_i do a distributed ring computation, where the first arm node according to permutation σ (i.e., the only arm node that received $first_i$=1) initializes max value B_m and argmax i_m. At each ring iteration (Steps 3.1, ..., 3.K-1, cf. Fig. 3(b)), the current arm node sends updated B_m and i_m to the next arm node cf. σ. Even though B_m and i_m do not change, it is important to re-encrypt $\mathsf{Enc}(B_m||i_m)$ before sending it to the next node to prevent an external observer from knowing when there is a change in the max and argmax (and hence learn information about which arms are pulled more often). Finally, once the ring computation reaches the last arm node relative to σ (i.e., the only one that received $next_i = 0$),

we go to Step 4.

e) Step 4: This step corresponds to the last two lines in Fig. 4(b) (the last arm node in the ring sends $\mathsf{Enc}(i_m)$ to AS), followed by the last two lines in Fig. 4(a) (AS receives and decrypts the index of the arm to be pulled at the next iteration).

f) Step 5: Once the budget is spent and no more arm has to be pulled, each arm node R_i (for $i \in [\![K]\!]$) encrypts with $\mathcal{E}_{\mathsf{DC}}$ its sum of rewards s_i and sends the result $\mathcal{E}_{\mathsf{DC}}(s_i)$ to AS.

g) Step 6: The node AS takes the K ciphertexts $\mathcal{E}_{\mathsf{DC}}(s_i)$ received at Step 5, and computes $\mathcal{E}_{\mathsf{DC}}(R) = \mathcal{E}_{\mathsf{DC}}(\sum_{i=1}^{K} s_i) = \prod_{i=1}^{K}(\mathcal{E}_{\mathsf{DC}}(s_i))$, thanks to the additive homomorphic property of Paillier cryptosystem. Then, AS sends $\mathcal{E}_{\mathsf{DC}}(R)$ to the data client, who is able to decrypt using sk and hence obtains R.

C. Analysis of UCB-DS

Next, we analyze the correctness (Sect. III-C1), security (Sect. III-C2), and complexity (Sect. III-C3) of UCB-DS.

1) Correctness: We point out that UCB-DS outputs exactly the same cumulative reward as UCB. The computations done in Fig. 4 to maximize the reward are the same as the one done in Fig. 2. Indeed, if we take UCB-DS and remove all encryptions/decryptions (both symmetric and asymmetric), and all messages are communicated in clear between participants, then we obtain a protocol that we call UCB-D, which outputs exactly the same result as UCB-DS. This happens because of the consistency property of the chosen cryptographic schemes i.e., if we encrypt a message M using Enc (or $\mathcal{E}_{\mathsf{DC}}$, respectively) to obtain a ciphertext C, then if we decrypt C using Dec (or $\mathcal{D}_{\mathsf{DC}}$, respectively), then we obtain exactly M. Next, to reduce UCB-D to UCB, we simply remove all distributions of tasks among participants and rewrite UCB-D as a sequential algorithm to obtain exactly UCB. In particular, the random permutation σ (that is generated at each round to decide in which order to iterate over arms) reduces to the randomness in the argmax function used in standard UCB cf. Fig. 2 when, if several arms have maximal B_i-value, then the argmax should be randomly picked among those arms.

2) Security: In Table II, we summarize what each participant in UCB-DS knows/does not know. The main properties of our protocol are:

- No cloud node can learn the cumulative reward and additionally:
  - Only AS and the pulled arm know which arm is pulled at each round. Arms that are not pulled can guess the pulled arm with average probability $\frac{1}{2} + \frac{1}{2K}$.
  - Only arm node R_i knows the sum of rewards for arm i.
- Only DC knows the cumulative reward, and she knows nothing else.
- An external observer cannot learn the cumulative reward, the sum of rewards for some arm, or which arm has been pulled at some round.

These properties subsume the list of desirable security properties listed in Sect. III-A. We omit here formal statements and proofs for all these security properties, which are available in [19].

TABLE II
WHAT EACH PARTICIPANT OF UCB-DS KNOWS AND DOES NOT KNOW.

Participant	*Knows*	*Does not know*
AS	• Arm pulled at each round	• Sum of rewards for some arm and cumulative reward
R_i	• Sum of rewards for arm i • Arm pulled at each round, with average probability $\frac{1}{2} + \frac{1}{2K}$	• Sum of rewards of other arm $j \neq i$ and cumulative reward
DC	• Cumulative reward	• Arm pulled at each round • Sum of rewards for some arm
External observer	• Nothing	• Arm pulled at each round • Sum of rewards for some arm and cumulative reward

TABLE III
NUMBER OF CRYPTOGRAPHIC OPERATIONS USED IN UCB-DS.

	Encryptions		*Decryptions*	
AES-CBC	K	(step 0)	K	(step 0)
	$(N-K+1)K$	(step 2)	$(N-K+1)K$	(step 2)
	$(N-K+1)(K-1)$	(step 3)	$(N-K+1)(K-1)$	(step 3)
	$(N-K+1)$	(step 4)	$(N-K+1)$	(step 4)
Paillier	K	(step 5)	1	(step 6)

3) Complexity: We detail in Table III the number of cryptographic operations used in each step of UCB-DS. By summing up, we obtain $O(NK)$ AES-CBC encryptions/decryptions, K Paillier encryptions, and one Paillier decryption. Hence, we have a number of AES-CBC operations linear in N, whereas the number of Paillier operations does not depend on N. These are desirable complexity properties. In particular, the number of Paillier operations (which are quite slow to evaluate in practice) depends only on K that is typically much smaller than N in bandit scenarios. Our implementation (cf. Sect. IV) follows the aforementioned theoretical analysis and confirms the linear time behavior and the scalability of UCB-DS.

D. Refinement

We propose the UCB-DS2 refinement, which adds slightly stronger security guarantees to UCB-DS, for few more cryptographic operations (but the similar asymptotic behavior as UCB-DS). A property of UCB-DS (cf. Table II) is that an arm node R_i knows with average probability of $\frac{1}{2} + \frac{1}{2K}$ what arm is pulled at the next round. This happens because during the ring computation, every arm sees in clear the partial argmax i_m. Our UCB-DS2 refinement removes this leakage.

The idea of UCB-DS2 is that, in addition to UCB-DS, we also encrypt the partial argmax i_m during the ring computation. This modification requires to introduce new keys. We recall that UCB-DS assumes an AES-CBC key that is shared between the data owner and all cloud participants and that is used for the functions $\mathsf{Enc}/\mathsf{Dec}$. For UCB-DS2, if we want that an arm node R_i cannot decrypt the partial argmax i_m received from the previous arm node in the ring, we need to encrypt i_m with some other key. This is why in UCB-DS2 we introduce K new AES-CBC keys, each of them shared between AS and a single R_i arm node. Each such key defines

functions Enc_i/Dec_i. We omit here the pseudocode and the analysis of UCB-DS2 that we include in [19].

IV. Experiments

We show that the overhead due to cryptographic primitives is reasonable, hence our protocols are feasible in practice. More precisely, we show the scalability of our protocols with respect to both parameters N and K through an experimental study using synthetic and real data. We compare:

- UCB = Standard UCB [3], outlined in Fig. 2.
- UCB-D = UCB with distribution of tasks among participants in the spirit of UCB-DS cf. Sect. III-B, but with all messages exchanged in clear among participants (i.e., UCB-D does not use any cryptographic primitive). The only overhead w.r.t. UCB is due to distribution of tasks.
- UCB-DS = Distributed Secure UCB cf. Sect. III-B.
- UCB-DS2 = Refinement of UCB-DS cf. Sect. III-D.

We implemented the algorithms in Python 3. For AES-CBC we used the *PyCryptodome* library[3] and keys of 256 bits. For Paillier, we used the *phe* library[4] in the default configuration with keys of 2048 bits. We did our experiments on a laptop with CPU Intel Core i7 of 2.80GHz and 16GB of RAM, running Ubuntu. Each reported result is averaged over 100 runs. In each run, we executed all algorithms using the same random seeds, needed for drawing arm rewards and for generating the permutation used to iterate in a random order over the arms when choosing the argmax arm to be pulled at the next round.

We make available on a public GitHub repository[5] our source code, together with the data that we used, the generated results from which we obtained our plots, and scripts that allow to install the needed libraries and reproduce our plots.

As expected, in each experiment, all four algorithms output exactly the same cumulative reward. The property that our secure algorithms return exactly the same cumulative reward as standard UCB is in contrast with differentially-private multi-armed bandit algorithms [4], [5], [6], where the returned cumulative rewards are different from that of standard UCB. Consequently, a shallow empirical comparison between these works and ours boils down to comparing apples and oranges: (i) on the one hand, the running time of differentially-private bandit algorithms is roughly the same as for standard UCB and is never reported in their experiments, whereas (ii) on the other hand, for our algorithms the cumulative reward is always the same as for standard UCB and consequently there is no point for us in doing any plot on the cumulative reward. Nevertheless, we carefully analyzed all experimental settings (N, K, μ) used in the related work, that we adapt for our scalability experiments, as we detail next.

a) Scalability with respect to N: In this experiment, we rely on six scenarios from the related work [4], [6] to fix K and μ, and to vary N. In Fig. 5, we show the results only for Scenario 1 [4]. We omit here the other scenarios, which yield similar results, included in [19]. We vary N from 10^2 to 10^5 that is also the maximum budget from [4], [6]. UCB and UCB-D have very close running times, and up to two orders of magnitude smaller than UCB-DS and UCB-DS2, which are also very close. All algorithms have a similar linear time behavior. The overhead between secure and non-secure algorithms comes naturally from the cryptographic primitives. Moreover, the two lines corresponding to the secure algorithms are not parallel with the other two lines because, cf. Sect. III-C3, the overhead due to Paillier encryptions depends only on K (that is fixed in the figure) and not on N (that varies in the figure), hence the Paillier overhead is more visible for small N. The running times of UCB-DS/UCB-DS2 for the largest considered budget N=10^5 is of $\sim$100 seconds, which remains practical. In Fig. 5, we also zoom on the time taken by each participant of UCB-DS for N=10^5. We observe that AS takes the lion's share, which is expected because at each round AS sends encrypted messages to all R_i participants, whereas each R_i sends an encrypted message only to one other participant. As expected, all R_i take roughly the same time. The shares taken by the data owner and the data client are the smallest among all participants, which is a desirable property because we require them to do as few computations as possible, whereas the bulk of the computation is outsourced to the cloud.

b) Scalability with respect to K: In this experiment, we fix N=10^5, and we vary K∈$\{5, 10, 15, 20\}$ and implicitly μ with μ_1=0.9 and $\mu_{2\leq i\leq K}$=0.8. We present results in Fig. 6. We observe, as in the previous experiment, a linear time behavior and a similar zoom on the time taken by each participant.

c) Real-world data: We also stress-tested our algorithms on real-world data, using the same data and setup as [20]. After a pre-processing step (detailed in [19]), we transformed real-world data in three bandit scenarios: Jester-small (K=10) and Jester-large (K=100) based on Jester dataset [21], and MovieLens (K=100) based on MovieLens dataset [22]. We ran each of these scenarios with N=10^5 that is the largest budget considered in [20]. Our results (cf. Fig. 7) essentially confirm the behavior observed in the synthetic experiments i.e., there are roughly two orders of magnitude between non-secure and secure algorithms. In the largest considered scenarios (Jester-large and MovieLens, both with K=100), where standard UCB takes around twenty seconds, both UCB-DS and UCB-DS2 take around one thousand seconds, that we believe acceptable as waiting time for the data client before getting the cumulative reward result for which she pays.

V. Conclusions and Future Work

We tackled the problem of cumulative reward maximization in multi-armed bandits, in a setting where data and computations are outsourced to some honest-but-curious cloud. We proposed UCB-DS, a distributed and secure protocol based on UCB, which yields exactly the same cumulative reward as UCB while enjoying desirable security properties that we precisely characterize. In particular, no cloud node or external observer can learn the cumulative reward, which can be seen only by the data client who pays a budget. We rely on

[3]https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html

[4]https://python-paillier.readthedocs.io/en/develop/

[5]https://github.com/radu1/secure-ucb

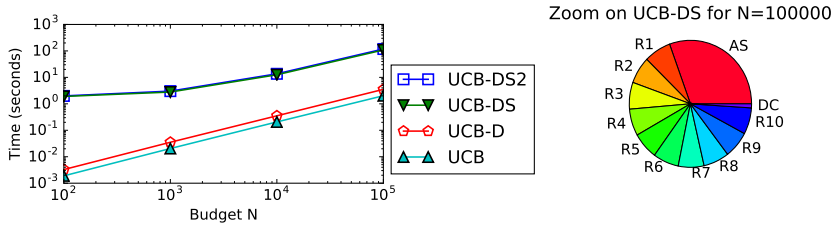


Fig. 5. Scalability with respect to N, for fixed $K = 10$, $\mu_1 = 0.9$, $\mu_2 = \ldots = \mu_{10} = 0.8$. In the zoom, we do not show DO (its share is close to 0).

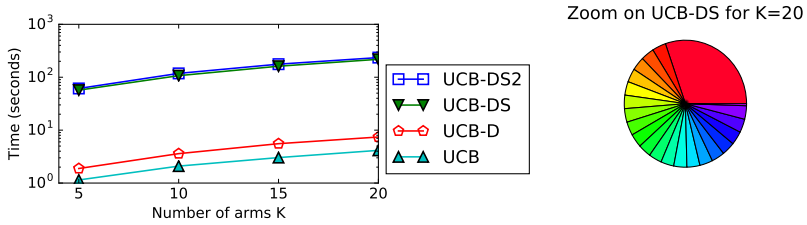


Fig. 6. Scalability with respect to K, for fixed $N = 10^5$. In the zoom (labels not shown because they would be colliding): the AS takes the lion's share, $\mathsf{R}_{1 \leq i \leq 20}$ take the 20 equal shares, DC is barely visible, and DO is not shown since its share is close to 0.

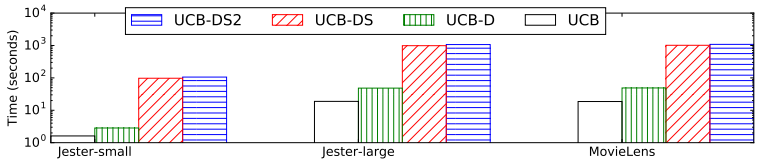


Fig. 7. Running times on three real-world data scenarios from [20].

distribution of tasks and on cryptographic schemes to achieve the security properties of UCB-DS, and we characterize the overhead of cryptography from both theoretical and empirical points of view. Our experiments show the scalability and practical feasibility of UCB-DS, and of its refinement UCB-DS2.

As future work, we plan to extend our scenario such that multiple data clients concurrently submit budgets to the cloud and receive corresponding cumulative rewards. In such a scenario, parallelism between nodes could be leveraged to improve the system's throughput.

References

[1] S. Bubeck and N. Cesa-Bianchi, "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems," *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[2] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast Homomorphic Evaluation of Deep Discretized Neural Networks," in *CRYPTO*, 2018, pp. 483–512.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[4] P. Gajane, T. Urvoy, and E. Kaufmann, "Corrupt Bandits for Preserving Local Privacy," in *ALT*, 2018, pp. 387–412.

[5] N. Mishra and A. Thakurta, "(Nearly) Optimal Differentially Private Stochastic Multi-Arm Bandits," in *UAI*, 2015, pp. 592–601.

[6] A. C. Y. Tossou and C. Dimitrakakis, "Algorithms for Differentially Private Multi-Armed Bandits," in *AAAI*, 2016, pp. 2087–2093.

[7] C. Dwork, "Differential Privacy," in *ICALP*, 2006, pp. 1–12.

[8] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *STOC*, 2009, pp. 169–178.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *ASIACRYPT*, 2017, pp. 409–437.

[10] "Advanced Encryption Standard (AES)," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2001, fIPS Publication 197.

[11] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption," in *FOCS*, 1997, pp. 394–403.

[12] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *EUROCRYPT*, 1999, pp. 223–238.

[13] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare, "Secure Best Arm Identification in Multi-Armed Bandits," in *ISPEC*, 2019, pp. 152–171.

[14] J. Audibert, S. Bubeck, and R. Munos, "Best Arm Identification in Multi-Armed Bandits," in *COLT*, 2010, pp. 41–53.

[15] R. Ciucanu, A. Delabrouille, P. Lafourcade, and M. Soare, "Secure Cumulative Reward Maximization in Linear Stochastic Bandits," in *ProvSec*, 2020.

[16] R. Agrawal, "Sample Mean Based Index Policies with $O(\log(n))$ Regret for the Multi-Armed Bandit Problem," *Advances in Applied Probability*, vol. 27, no. 4, pp. 1054–1078, 1995.

[17] R. Munos, "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning," *Foundations and Trends in Machine Learning*, vol. 7, no. 1, pp. 1–129, 2014.

[18] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[19] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare, "Secure Outsourcing of Multi-Armed Bandits," in *TR at https://hal.inria.fr/ /hal-02953292*, 2020.

[20] P. Kohli, M. Salek, and G. Stoddard, "A Fast Bandit Algorithm for Recommendation to Users With Heterogenous Tastes," in *AAAI*, 2013.

[21] K. Y. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A Constant Time Collaborative Filtering Algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[22] F. M. Harper and J. A. Konstan, "The MovieLens Datasets: History and Context," *ACM TiiS*, vol. 5, no. 4, pp. 19:1–19:19, 2016.