Journal of Artificial Intelligence Research 73 (2022) 737-765 Submitted 07/2021; published 02/2022

SAMBA: A Generic Framework for Secure Federated Multi-Armed Bandits

Radu Ciucanu RADU.CIUCANU@UNIV-GRENOBLE-ALPES.FR
Univ. Grenoble Alpes, CNRS LIG, France

Pascal Lafourcade PASCAL.LAFOURCADE@UCA.FR
Univ. Clermont Auvergne, CNRS LIMOS, France

Gael Marcadet GAEL.MARCADET@UCA.FR
Univ. Clermont Auvergne, CNRS LIMOS, France

Marta Soare MARTA.SOARE@UNIV-GRENOBLE-ALPES.FR
Univ. Grenoble Alpes, CNRS LIG, France

Abstract

The multi-armed bandit is a reinforcement learning model where a learning agent repeatedly chooses an action (pull a bandit arm) and the environment responds with a stochastic outcome (reward) coming from an unknown distribution associated with the chosen arm. Bandits have a wide-range of application such as Web recommendation systems. We address the cumulative reward maximization problem in a secure federated learning setting, where multiple data owners keep their data stored locally and collaborate under the coordination of a central orchestration server. We rely on cryptographic schemes and propose SAMBA, a generic framework for Secure federAted Multi-armed BAndits. Each data owner has data associated to a bandit arm and the bandit algorithm has to sequentially select which data owner is solicited at each time step. We instantiate SAMBA for five bandit algorithms. We show that SAMBA returns the same cumulative reward as the non-secure versions of bandit algorithms, while satisfying formally proven security properties. We also show that the overhead due to cryptographic primitives is linear in the size of the input, which is confirmed by our proof-of-concept implementation.

1. Introduction

Federated learning is a machine learning paradigm where multiple data owners collaborate in solving a learning problem, under the coordination of a central orchestration server (Kairouz, McMahan, & et al., 2021). Each data owner's raw data is stored locally and not exchanged or transferred. The development of machine learning algorithms in federated learning settings is a timely topic, which touches several communities: "*a longstanding goal pursued by many research communities (including cryptography, databases, and machine learning) is to analyze and learn from data distributed among many owners without exposing that data*" (Kairouz et al., 2021). We tackle this goal by relying on cryptographic techniques to develop a secure framework for learning on distributed data.

In particular, we focus on multi-armed bandits, a reinforcement learning model where a learning agent needs to sequentially decide which "arm" to choose among several options (with unknown reward distributions) available in the environment. After each arm selection, the environment responds with a stochastic reward drawn from the reward distribution associated to the chosen arm. To maximize the cumulative reward, the learning agent has

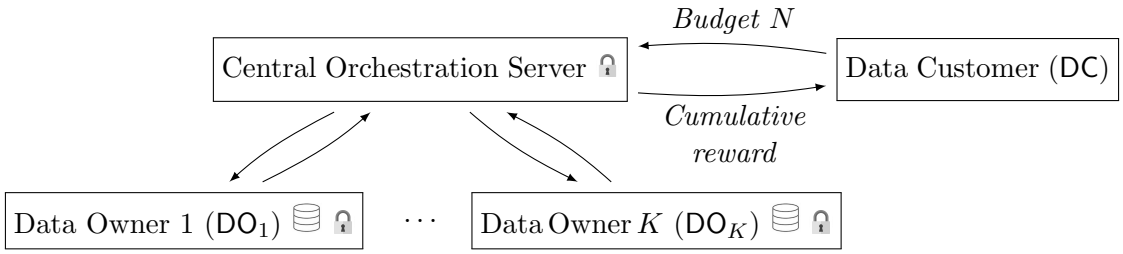


Figure 1: Instantiation of the federated learning paradigm for cumulative reward maximization in multi-armed bandits.

to continuously face the so-called exploration-exploitation dilemma and decide whether to *explore* by choosing arms with more uncertain associated values, or to *exploit* the information already acquired by choosing the arm with the seemingly largest associated value. Bandits have practical applications such as Web recommender systems, where the arms are the recommended items and the rewards are given by the user ratings. More specifically, we tackle the problem of *secure cumulative reward maximization in federated multi-armed bandits*, a problem that to the best of our knowledge has not been previously studied in the literature. Our goal is to propose a generic federated framework that is guaranteed to return exactly the same cumulative reward as standard bandit algorithms (Sutton & Barto, 2018, Chapter 2), while guaranteeing formally proven security properties.

As depicted in Figure 1, we assume that the *data* i.e., the reward functions associated to K bandit arms are stored locally by K *data owners* ($\mathsf{DO}_1, \ldots, \mathsf{DO}_K$). The data is potentially sensitive, hence it should remain stored locally and cannot be seen in clear by any participant other than its owner (this is why we depict locks near each DO_i). As typically done in federated learning, we assume that the learning algorithm is done by some *central orchestration server* (referred to as *server* in the sequel). The *data customer* (DC) sends a budget N to the server and receives the cumulative reward. Moreover, we assume that the participants in Figure 1 (data owners, server, and data customer) are *honest-but-curious* i.e., they correctly do the required computations, but try to gain as much information as possible based on the data that they see. In particular, we aim at minimizing the data leakage to the server (this is why we also depict a lock near the server) e.g., the server cannot see rewards produced by each data owner. Additionally, an external observer that has access to all messages exchanged between the aforementioned participants should not be able to learn any input, output, or intermediate data.

To motivate our problem setting, we present an example based on federated learning in recommendation systems (Shi & Shen, 2021; Li, Song, & Fragouli, 2020). In this case, the K data owners are K *local stores*, each of them being able to recommend items based on potentially sensitive data. Moreover, the data customer is a *parent company* that displays on its Web site recommended items that can come from any of the K local stores. Given a *budget* N (i.e., total number of recommended items that can be sequentially displayed by the parent company), the goal of the parent company is to maximize the *cumulative reward* (i.e., maximize the sum of obtained user ratings on the recommended items). The bandit al-

gorithm has to decide *how* to sequentially choose the N recommended items, which should come from the K local stores. The aforementioned recommendation systems motivating example can be easily adapted to other classical federated learning applications where security is of paramount importance e.g., commercial, financial, and medical domains (Kairouz et al., 2021).

Our goal is to build a generic federated learning framework such that, given some standard bandit algorithm $\mathcal{A}$, we are able to plug $\mathcal{A}$ in our framework and obtain the same cumulative reward as $\mathcal{A}$, while guaranteeing data security. Our goal could be theoretically achieved by relying on a fully homomorphic encryption (FHE) scheme (Gentry, 2009), which allows to compute any function directly in the encrypted domain. Indeed, in theory it would suffice that each data owner encrypts its data with a FHE scheme; then, the server would do the computations needed for cumulative reward maximization directly in the encrypted domain. However, it remains an open question how to build a practical FHE system. Although state-of-the-art FHE systems (SEAL[1] and HElib[2]) have done remarkable progress, computations with real numbers are still limited because of the noise needed for FHE multiplications. Moreover, even simple functions such as comparisons needed in all bandit algorithms (e.g., compute an argmax or a probability matching) require complex and time-consuming computations in FHE systems, even for approximate results and even for recent state-of-the-art algorithms (Cheon, Kim, & Kim, 2020; Garcelon, Perchet, & Pirotta, 2021). Since FHE systems cannot be currently used off-the-shelf to propose secure federated bandit algorithms, our approach is based on *simpler cryptographic schemes*, in conjunction with *secure multi-party computation*.

Summary of Contributions and Paper Organization

In Section 2, we discuss the positioning of our problem setting w.r.t. the related work. In Section 3, we introduce basic notions on bandit algorithms and cryptographic tools.

In Section 4, we present Samba, a generic framework for secure cumulative reward maximization for federated bandits. The key ingredients of Samba are:

- We distribute the server computations between two nodes: Controller (that sees only encrypted messages and distributes computation tasks among participants) and Comp (whose only goal is to compare numbers obtained after permuting and masking bandit arm scores). This distribution technique allows to perform comparisons, without revealing to the server neither the bandit arm scores nor the arm pulled at some time step.
- We exchange only encrypted messages such that an external network observer cannot learn any input, output, or intermediate data. Moreover, each data owner can see in clear the raw data pertaining to its bandit arm and nothing else. The data owners communicate only with Controller, with messages encrypted with *indistinguishable under chosen-plaintext attack (IND-CPA)* cryptographic schemes, namely symmetric AES-GCM (2001, 2007) and asymmetric (Paillier, 1999).
- At the end of Samba, we compute the cumulative reward by summing up the rewards from each data owner directly in the encrypted domain, by relying on the additive

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

Data \ Participant	DO_i	DC	Server	Ext
Cumulative reward		X		
Sum of rewards and number of pulls for DO_i	X			
Sum of rewards and number of pulls for $\mathsf{DO}_{j\neq i}$				
Arm pulled at time step t	X*			
Reward at time step t	X*			

Figure 2: *Security properties.* The X should be read as: the participant can see in clear the concerned piece of data, whereas an empty case means the opposite. The $\star$ should be read as: only if DO_i is pulled at time step t. Ext should be read as: an external network observer having access to all messages exchanged between participants.

homomorphic property of Paillier. Hence, neither the data owners nor the server nodes can see in clear the cumulative reward: only the data customer that invested a budget for computing the cumulative reward is able to decrypt it.

We instantiate Samba to secure five bandit algorithms: ε-greedy, UCB, Thompson Sampling, Softmax, and Pursuit. In Section 5, we provide the theoretical analysis of Samba. In a nutshell, we show that Samba enjoys the following features:

- *Genericity*: Samba can be instantiated with any bandit algorithm that satisfies the properties (i) computing the score of an arm does not depend on the other arms, and (ii) selecting the arm to be pulled at some round can be done in the presence of some random masks and permutations on which Samba relies to hide the real arm scores. In particular, the five aforementioned bandit algorithms satisfy these properties. We also include examples of bandit algorithms that cannot be instantiated in Samba: an existing algorithm (Reinforcement Comparison) that cannot be instantiated because of (i), and an hypothetical algorithm that cannot be instantiated because of (ii) as we are not aware of any off-the-shelf algorithm that does not satisfy (ii).
- *Correctness*: Samba returns exactly the same cumulative reward as the standard (non-secure and non-federated) bandit algorithms because the cryptographic primitives and distribution of tasks do not change the arm selection strategy w.r.t. the standard algorithms.
- *Security*: we summarize the *security properties* in Figure 2. We give a brief intuition for each participant:
  - DO_i can see data concerning arm i and nothing else about other arms, nor about the cumulative reward.
  - Only DC can see the cumulative reward for which she spends a budget. She can see only this piece of information for which she pays, and nothing else.
  - The server nodes (Controller and Comp) and external observers cannot learn any input, output, and intermediate data.
- *Complexity*: the number of cryptographic operations is linear in the input: Samba uses $O(NK)$ AES-GCM operations and $O(K)$ Paillier operations. It is a desirable feature that the number of Paillier operations does not depend on the budget N

because N is typically larger than the number of arms K, and AES-GCM is much faster than Paillier.

In Section 6, we report on a proof-of-concept empirical evaluation that shows the feasibility and scalability of Samba. We present directions of future work in Section 7.

2. Related Work

In Section 2.1, we discuss the positioning of our problem setting w.r.t. the federated learning literature. In Section 2.2, we discuss related works on federated and secure bandits.

2.1 Positioning in the Federated Learning Paradigm

We precisely position our problem setting w.r.t. a state-of-the-art federated learning survey (Kairouz et al., 2021). Our framework is built upon the typical federated learning characteristics:

- *Data distribution.* Data is generated locally and remains decentralized. Each data owner stores its own data and cannot read the data of the other data owners.
- *Orchestration.* A central orchestration server organizes the learning, but never sees raw data.

Moreover, among the main federated learning settings (cross-silo vs cross-device) (Kairouz et al., 2021), our framework pertains to the *cross-silo federated learning setting*, whose typical characteristics are:

- *Distribution scale.* There are rather few data owners (less than 100), which can be different organizations e.g., stores, hospitals.
- *Data availability and reliability.* Each data owner is assumed to be available when it is required to do some computation tasks and there are no machine failures.
- *Primary bottleneck.* In general, it might be the computation and communication costs. In our framework, both costs have the same asymptotic big-O complexity.
- *Addressability.* Each data owner has an id that allows the central orchestration server to access it specifically.
- *Statefulness.* Each data owner is stateful i.e., it maintains local variables throughout the execution of the entire framework.
- *Data partition.* The partition should be fixed. In our case, we assume *feature-partitioned (vertical)* data i.e., each data owner has data pertaining to a single bandit arm.
- *Incentive mechanisms.* There is the need for incentive mechanisms to ensure honest participation of the data owners, since they may also be business competitors e.g., the local stores from our motivating example from the introduction. We assume a monetary incentive derived from data customer's budget.

As federated learning *threat model*, we assume that all participants (data owners, data customer, central orchestration server) are *honest-but-curious*, which means that they can inspect all received messages but cannot tamper the data and computations needed for the learning algorithm. We assume the classical formulation (Goldreich, 2004) (Chapter 7.5, where *honest-but-curious* is denoted *semi-honest*), in particular (i) each 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.

To deal with the honest-but-curious model, we rely on two standard cryptographic techniques (Kairouz et al., 2021):

- *Secure multi-party computation.* The participants collaborate to simulate a fully trusted third party who can: (i) compute a function of inputs provided by all the participants, and (ii) reveal the computed value to a chosen participant (the data customer), with no party learning anything further.
- *Homomorphic encryption.* As discussed in the introduction, it is not currently possible to efficiently rely on a fully homomorphic encryption scheme. We can nonetheless rely on the partially homomorphic Paillier scheme that is additive homomorphic and allows to sum up the rewards from each data owner directly in the encrypted domain.

Being an emerging topic at the intersection of several communities, there are multiple interesting variants of the *cross-silo federated learning setting* that we consider here. Most of them are still open problems, in particular in settings of sequential decision making (Kairouz et al., 2021), such as in the multi-armed bandit model. The very few recent works that we are aware of are on the related best arm identification problem. (i) A recent study focuses on federated learning aspects such as the *incentivization of data owners* (Shi, Xu, Xiong, & Shen, 2021b), but without additionally considering the data security aspect. In their setting, all DOs share the same K arms where an arm i yields different rewards from a DO to another. The goal of each DO is selfish: they want to collect as much reward as possible during a certain time horizon. They assume that the server can observe the rewards from all DOs and use them to compute the arm with the highest average reward over all DOs. (ii) Another study tackles the *robustness with respect to Byzantine adversaries* in federated best arm identification (Mitra, Hassani, & Pappas, 2021). In their setting, each DO has access to a subset of the K arms and to ensure robustness, their methods implies that each subset of arms is sampled by multiple DOs. *In contrast, in our proposed framework* Samba*, the data of a* DO *is locally stored and never exchanged, hence it is hidden from all the other participants.* Furthermore, a main characteristic is that Samba keeps the *same arm selection strategy as the standard (non federated, non secure) bandit algorithm*, a property that cannot be achieved if some DOs are selfish or not robust. While changing some characteristics of our federated learning setting will probably lead to interesting complementary problems, these modifications are outside the scope of Samba.

2.2 Positioning w.r.t. Federated and Secure Bandits

To the best of our knowledge, our work is the first that relies on cryptographic techniques to provide data security guarantees for federated multi-armed bandit algorithms.

Federated multi-armed bandits is an emerging topic, with few recent works that consider the federated learning paradigm for sequential decision making problems, where data is observed in response to interactions with an unknown environment. At each time step, the learner has only limited feedback about the arm that is pulled and this makes the setting more challenging compared to the typical supervised learning scenarios, where all training data is available from the beginning of the learning process. The recent works tackling federated bandits, consider different models: standard stochastic (Shi & Shen,

2021; Li et al., 2020), bandits with graph structure (Zhu, Zhu, Liu, & Liu, 2021), and linear bandits (Dubey & Pentland, 2020; Huang, Wu, Yang, & Shen, 2021). For all these works, the main focus in on adapting bandit algorithm to the federated setting, and some of them additionally rely on differential privacy (Dwork & Roth, 2014) to protect the data.

In particular, the first works on cumulative reward maximization in (private) federated multi-armed bandits (Shi & Shen, 2021; Li et al., 2020; Zhu et al., 2021; Shi, Shen, & Yang, 2021a) focus on the analysis of the gain in sharing data coming from multiple DOs for obtaining better *local* (DO-specific) and respectively *global* cumulative rewards (for all participants in the federated learning process). The typical assumption is that all DOs have access to the same subset of arms, which corresponds to an horizontal data partition. Another typical assumption from all these works is that the DOs *exchange information about the rewards they observe and about the indices of their selected arms* with their neighbors (Li et al., 2020; Zhu et al., 2021), respectively with the central orchestration server (Shi & Shen, 2021; Zhu et al., 2021). Before sharing these pieces of information, DOs apply differential privacy mechanisms to inject noise in their local data to keep it private from the other participants. For the next time steps, the bandit algorithm will continue to select arms based on the differentially-private information that is transmitted between participants.

A differentially-private bandit algorithm takes roughly the same computation time as the standard algorithm, but because of the noise that is injected in the data to ensure differential privacy, the arm selection strategy is altered. Thus, the modified selection strategy leads to a different output and a reduced performance (increased regret) compared to that of the standard bandit algorithm. On the other hand, in a cryptographic approach, the local data of each DO (concerning e.g., their observed rewards) is never exchanged in clear: encryption techniques are used to guarantee that local data maintained by each DO is hidden from the other participants. By relying on a carefully chosen set of primitives (AES-GCM and Paillier in the case of Samba), cryptographic approaches do not change the arm selection and output the same result as the standard algorithm, at the price of an increased computation time due to the use of cryptographic primitives. Although we share the common goal of data protection in federated bandits, the use of different techniques (differential privacy in the related works vs cryptography in our work) leads to complementary systems, whose different architecture and trade-offs are not comparable. In addition, in contrast with all previous federated multi-armed bandit frameworks for cumulative reward maximization, we focus on a vertical data partition (cf. Section 2.1) and our secure framework guarantees that local data maintained by each DO is hidden from the other participants.

There exist only a few cryptography-based secure protocols for bandits, in settings where all data is outsourced to the honest-but-curious cloud (Ciucanu, Lafourcade, Lombard-Platet, & Soare, 2020; Ciucanu, Delabrouille, Lafourcade, & Soare, 2020; Ciucanu, Lafourcade, Lombard-Platet, & Soare, 2019) and no other work proposing cryptography-based secure protocols for federated bandits. The protocol that is the closest to Samba also considers the problem of secure cumulative reward maximization for standard stochastic bandits (Ciucanu et al., 2020). There are two main differences between them (Ciucanu et al., 2020) and Samba. (i) The data distribution assumptions are different: they assume that all data is outsourced to the cloud, whereas Samba focuses on a federated learning setting where data is stored locally by each owner and never exchanged. Consequently, the respective distributed architectures are intrinsically different. (ii) Their protocol is catered

for securing the UCB algorithm, whereas Samba is a generic framework where multiple bandit algorithms can be easily plugged in. Among the algorithms supported in Samba, we have UCB and similar argmax-based algorithms. Moreover, Samba also supports more complex algorithms where arms are pulled based on a probability matching.

3. Preliminaries

We present bandit algorithms in Section 3.1 and cryptographic tools in Section 3.2. Before that, we introduce some useful notation on which we rely throughout the paper:

- By $[x_1, \ldots, x_n]$ we denote the list containing, in order, the elements $x_1, \ldots, x_n$.
- Given integers x and y such that $x \leq y$, by $[\![x, y]\!]$ we denote the list $[x, x+1, \ldots, y]$. By $[\![x]\!]$ we denote the list $[1, \ldots, x]$.
- By $[x_i]_{i \in [\![n]\!]}$ we denote the list $[x_1, \ldots, x_n]$.
- A permutation $\sigma : L \to L$ is a function for which every element occurs exactly once as an image value; by σ^{-1} we denote the inverse of σ.
- Given a permutation σ, by $\sigma([x_i]_{i \in [\![n]\!]})$ we denote the permuted list $[\sigma(x_1), \ldots, \sigma(x_n)]$.

3.1 Bandit Algorithms

The historical motivation (Thompson, 1933) behind the multi-armed bandit model concerns the adaptive design of clinical trials. For a given disease, a doctor can choose among K drugs with probability of success $\mu_1, \ldots, \mu_K$ unknown at the beginning of the clinical trial. At each time step t, the doctor chooses a drug $i \in [\![K]\!]$ for a patient. If the drug i heals the patient, we say that drug generates a reward 1; otherwise, we say that the reward is 0. The K bandit arms model the effectiveness of the K treatments available in the clinical trial. The assumption is that the rewards observed from each arm i are independent samples drawn from a Bernoulli distribution associated to arm i. Maximizing the sum of observed rewards means maximizing the number of healed patients from the clinical trial. The design of efficient multi-armed bandit strategies is a dynamic research topic, also motivated by good empirical performance in a wide range of modern applications, from Web advertisement and recommender systems (Li, Chu, Langford, & Schapire, 2010) to game playing (Kocsis & Szepesvári, 2006).

In this paper, we consider the typical setting of stochastic multi-armed bandits with Bernoulli rewards. Next, we introduce the notation related to bandit algorithms.

A bandit algorithm takes as **input** the budget N and the number of arms K, and gives as **output** the sum of observed rewards for all arms. The **unknown environment** of the bandit algorithm consists of K distributions associated to the K arms. We consider Bernoulli distributions 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. 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 returning 0 is $1–\mu_i$. The agent sequentially selects the N arms to be pulled with the goal of maximizing the sum of rewards.

While Samba can incorporate several cumulative reward maximization **algorithms** with Bernoulli rewards (we refer to Section 5.1 for an analysis of the property needed by bandit algorithms to fit in Samba), in this paper we illustrate Samba using a representative

```
/* Initialization: pull each arm once & initialize variables */
for i ∈ ⟦K⟧
  r ← pull(i)                                   /* Random reward for arm i */
  s_i ← r                                       /* Sum of observed rewards for arm i */
  n_i ← 1                                       /* Number of pulls of arm i */
/* Exploration-exploitation: pull one arm at each time step t */
for t ∈ ⟦K + 1, N⟧                              /* Only a budget of N − K is left */
  Choose i_m according to algorithm A
  r ← pull(i_m)
  s_{i_m} ← s_{i_m} + r
  n_{i_m} ← n_{i_m} + 1
return s_1 + ... + s_K
```

Figure 3: Generic cumulative reward maximization with bandit algorithm $\mathcal{A}$.

selection of five textbook algorithms (Sutton & Barto, 2018; Kuleshov & Precup, 2014; Russo, Van Roy, Kazerouni, Osband, & Wen, 2018), which represent a variety of strategies. To minimize redundancy when presenting the aforementioned collection of algorithms, we present what is common to all of them in Figure 3. In particular, each bandit algorithm needs to store, for each arm $i \in [\![K]\!]$, two variables s_i (sum of rewards) and n_i (number of pulls), based on which it can compute $\hat{\mu}_i = \frac{s_i}{n_i}$ (empirical mean). Each bandit algorithm has its own strategy for choosing i_m for each time t, that we present in Figure 4. We stress that at each time t only one arm is pulled, thus only the corresponding variables s_{i_m} and n_{i_m} will be updated, while the sum of rewards and the number of pulls for all other arms are not affected. To simplify notation, we drop the index indicating the time t whenever the variables to be updated at time t are obvious for the context. In the sequel, by (arm) **score** of a bandit algorithm $\mathcal{A}$ we mean, depending on $\mathcal{A}$, either the argument of the argmax, or the probability needed to compute a probability matching.

3.2 Cryptographic Tools

SAMBA relies on two cryptosystems: Paillier and AES-GCM, which are both IND-CPA secure. In this section, we introduce these cryptographic tools, while aiming to provide enough background to understand how they are useful in SAMBA to have formally proven correctness and security properties. Each of the two cryptographic schemes has a security parameter λ that is input to key generation. By 1^λ we denote the unary representation of λ, which is a standard notation in cryptography. Our security theorems are asymptotic i.e., they describe the behavior when λ becomes infinitely large. In practice, the security parameter is the length of the keys, for both Paillier and AES-GCM.

Paillier Asymmetric Encryption. Paillier's cryptosystem (Paillier, 1999) 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 such that:

- $\mathcal{G}(1^\lambda)$ 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 $(\mathbb{Z}^*_{n^2}, \cdot)$,

Algorithm $\mathcal{A}$	*Strategy for choosing the arm at time* t
ε**-greedy** with fixed or decreasing ε	**return** $\begin{cases} \arg\max_{i\in[\![K]\!]} \hat{\mu}_i \text{ , with probability } 1-\varepsilon & \textit{(exploit)} \\ \text{a random arm, with probability } \varepsilon & \textit{(explore)} \end{cases}$
UCB	**return** $\arg\max_{i\in[\![K]\!]}(\hat{\mu}_i + \sqrt{\frac{2\ln(t)}{n_i}})$
Thompson Sampling	**for** $i \in [\![K]\!]$ **sample** $\theta_i \sim \text{Beta}(s_i+1, n_i - s_i + 1)$ **return** $\arg\max_{i\in[\![K]\!]} \theta_i$
Softmax – Boltzmann distribution with temperature τ	**return** arm i with probability $\frac{e^{\hat{\mu}_i/\tau}}{\sum_{j=1}^{K} e^{\hat{\mu}_j/\tau}}$
Pursuit with learning rate β	**return** arm i with probability $p_{i,t} =$ $\begin{cases} \frac{1}{K}, \text{if } t = 0 \\ p_{i,t-1} + \beta(1 - p_{i,t-1}), \text{if } t > 0 \text{ and } i = \arg\max_{i\in[\![K]\!]} \hat{\mu}_i \\ p_{i,t-1} + \beta(0 - p_{i,t-1}), \text{otherwise} \end{cases}$

Figure 4: Instantiations of Figure 3 for five cumulative reward maximization algorithms. We recall that for each arm $i \in [\![K]\!]$ at time t, n_i is the number of pulls, s_i is the sum of rewards, and $\hat{\mu}_i = \frac{s_i}{n_i}$ is the empirical mean.

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 $(\mathsf{sk}, \mathsf{pk})$.

- $\mathcal{E}_{\mathsf{pk}}(m)$ randomly picks $r \in \mathbb{Z}_n^*$, computes $c = g^m \cdot r^n \mod n^2$, and outputs c.
- $\mathcal{D}_{\mathsf{sk}}(c)$ computes $m = L(c^\Lambda \mod n^2) \cdot M \mod n$, and outputs m.

We say that Paillier is *asymmetric* because it relies on different keys for encryption and decryption. Moreover, Paillier 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 $c_1 = \mathcal{E}_{\mathsf{pk}}(m_1) = g^{m_1} \cdot r_1^n \mod n^2$ and $c_2 = \mathcal{E}_{\mathsf{pk}}(m_2) = g^{m_2} \cdot r_2^n \mod n^2$, is the encryption of the sum of m_1 and m_2. Indeed, we have:

$$\begin{aligned} \mathcal{E}_{\mathsf{pk}}(m_1) \cdot \mathcal{E}_{\mathsf{pk}}(m_2) &= c_1 \cdot c_2 \mod n^2 \\ &= (g^{m_1} \cdot r_1^n) \cdot (g^{m_2} \cdot r_2^n) \mod n^2 \\ &= (g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n) \mod n^2 \\ &= \mathcal{E}_{\mathsf{pk}}(m_1 + m_2)\ . \end{aligned}$$

AES-GCM Symmetric Encryption. AES (AES, 2001) is a NIST standard for symmetric encryption that encrypts messages of 128 bits. To encrypt messages larger than 128 bits, we use AES with a symmetric encryption mode. Among all existing modes, we rely on GCM (*Galois Counter Mode*) (AES, 2007), which has been recently added to Transport Layer Security (TLS[3]), a standard protocol that provides communication security over a

3. `https://datatracker.ietf.org/doc/html/rfc8446`

computer network. The AES-GCM cryptosystem is defined by a triple of polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a security parameter λ such that $\mathsf{Gen}(1^\lambda)$ generates Key, a uniformly random symmetric key of 128, 192 or 256 bits, according to λ. We say that AES-GCM is *symmetric* because it relies on the same key for encryption and decryption. Let $c = \mathsf{Enc}(m)$ be the encryption of m and $m = \mathsf{Dec}(c)$ be the decryption of c, with the same symmetric key Key.

IND-CPA (INDistinguishability under Chosen-Plaintext Attack). Both Paillier and AES-GCM are IND-CPA (Bellare, Desai, Jokipii, & Rogaway, 1997): (i) Paillier is IND-CPA under the decisional composite residuosity assumption (Paillier, 1999), and (ii) AES-GCM is IND-CPA under the assumption that AES is a pseudo-random permutation (Bellare et al., 1997). In the security properties of SAMBA, the notion of "better than random" is consistent with the IND-CPA property. Intuitively, this means that both cryptographic schemes are secure against attackers that have access only to a polynomial number of encrypted messages (and in particular have no access to a decryption oracle). This is a reasonable assumption in the honest-but-curious model introduced in Section 2.1.

More precisely, let $\Pi = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a cryptographic scheme. The *probabilistic polynomial-time (PPT) adversary* Adv tries to break the security of Π. The IND-CPA game, denoted by $\mathrm{EXP}(\mathsf{Adv})$, works as follows: Adv 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. Adv knows m_0, m_1 and c, and 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 $\alpha = \Pr[b' \leftarrow \mathrm{EXP}(\mathsf{Adv}); b = b']$, we denote the probability that Adv 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 γ is negligible in λ, denoted $\mathrm{negl}(\lambda)$, if for every positive polynomial $p(\cdot)$ and sufficiently large λ, $\gamma(\lambda) < 1/p(\lambda)$. We denote by $\mathrm{negl}(\lambda)$ any function negligible in λ. Notably, we have $\mathrm{negl}(\lambda) + \mathrm{negl}(\lambda) = \mathrm{negl}(\lambda)$ and we may write $x + \mathrm{negl}(\lambda)$ instead of $x - \mathrm{negl}(\lambda)$.

All theoretical security properties of SAMBA also hold if we choose any other IND-CPA symmetric scheme instead of AES-GCM, 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-GCM 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.

4. A Generic Framework for Secure Federated Multi-Armed Bandits

As mentioned in the introduction, we have K data owners (DO_i), one data customer (DC), and the server. By looking at the bandit algorithms presented in Section 3.1, we can roughly classify the bandit computations in two categories:

- *Local computations*, which can be done locally by each DO_i e.g., pull an arm to generate a reward, maintain variables needed for all algorithms (n_i and s_i) as well as algorithm-specific variables (e.g., $p_{i,t}$).

- *Global computations*, done by the server, where data from all DO_i is required. The two types of global computations that appear in Figure 4 are:
  - Compute the argmax of the arm scores.
  - Choose an arm according to a probability matching.

We recall that we aim at a secure protocol that leaks as little data as possible to the server. For instance, we want to avoid that the server knows which arm is pulled at some time step. Hence, we choose to distribute the server computations among two nodes, using an idea already known in the literature e.g., in the context of private outsourced sort (Baldimtsi & Ohrimenko, 2015):

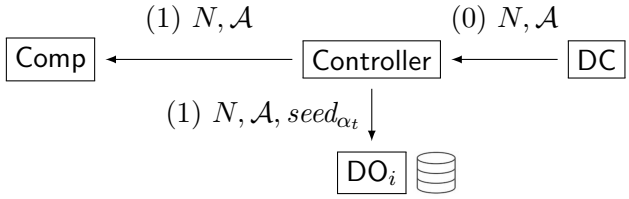
- Controller. This node interacts with DC and DO_i. More precisely, when we need to do some global computation, each DO_i sends to Controller a score encrypted with a secret key unknown to Controller. After receiving the collection of scores from all DO_i, note that Controller cannot compare the scores in order to decide which arm to pull next. Indeed, we encrypt the scores with the key of a second node, whose only task is to compare scores.
- Comp. This node interacts only with Controller, who sends a collection of encrypted scores each time we need to do a global computation. Comp is able to decrypt and compare the scores. The subtlety is that Controller permutes the encrypted scores before forwarding them to Comp, hence Comp is not able to associate the arm pulled at some time step with the real arm index. Moreover, to hide the real scores from Comp, we require each DO_i to mask the real arm scores with a mask that is order-preserving (to be able to correctly compute argmax) and proportion-preserving (to be able to correctly compute a probability matching).

This leads us to the architecture of Samba outlined in Figure 5, where we have $K + 3$ participants: K data owners + 1 data customer + 2 server nodes (Controller and Comp). We give pseudocode in Figure 6.

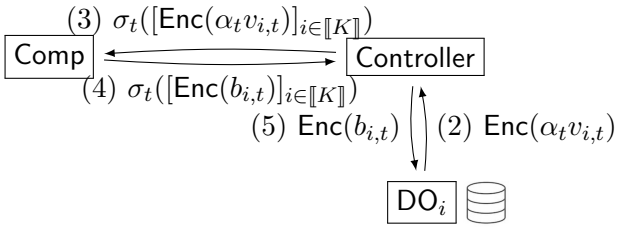
Before detailing the steps of Samba, we briefly introduce the needed cryptographic keys. By $\mathsf{Enc}(\cdot)$ we denote encryption using an AES key shared before starting the actual protocol among Comp and all DO_i (we stress that this key is unknown to Controller). Moreover, we assume that DC generates a Paillier's key pair $(\mathsf{pk}, \mathsf{sk})$ and for sake of clarity we denote by $\mathcal{E}(\cdot)$ the encryption with pk.

4.1 Architecture

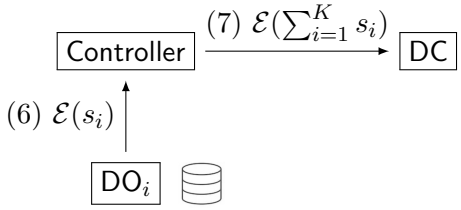
We present the main bricks of Samba's architecture, whose design is motivated by a trade-off between its *genericity* (i.e., Samba should be generic enough to plug in different multi-armed bandit algorithms) and *efficiency* (i.e., Samba should be efficient when executing typical computations needed for multi-armed bandit algorithms). The architecture of Samba presented in Figure 5 and explained hereafter allows to directly plug in four bandit algorithms instantiated earlier (**UCB**, **Thompson Sampling**, **ε-greedy** and **Softmax**). Finally, in Section 4.2 we present the extension needed in Samba such that more complex algorithms such as **Pursuit** can be easily plugged in.



(a) *Parameter setup* (Steps 0 and 1) done only once at the beginning.



(b) *Core of* SAMBA (Steps 2–5) used to select the arm to be pulled. Repeated for each time step t until the end of the budget.



(c) *Cumulative reward computation* (Steps 6 and 7) done once at the end.

Figure 5: Architecture of SAMBA.

4.1.1 PARAMETER SETUP (FIGURE 5(A))

At Step 0, DC sends to Controller the budget N and the name of the algorithm $\mathcal{A}$ that should be used for cumulative reward maximization. We list in Figure 4 the algorithms currently supported in SAMBA, hence we allow DC to send to Controller any value from the column "Algorithm $\mathcal{A}$" in Figure 4 i.e., the name of the algorithm and the algorithm-specific parameters, when applicable. Such parameters are ε (if ε is fixed) or f_ε (if ε is decreasing over time) for ε-**greedy**, τ for **Softmax**, and β for **Pursuit**.

At Step 1, Controller forwards the received input to DO_i and Comp. Additionally, Controller sends to DO_i a randomly generated $seed_{\alpha_t}$, which is used to draw random masks α_t for the arm scores such that Comp cannot see the real arm scores. As detailed later in the sequel, α_t changes at each time step $t \in [\![K+1, N]\!]$.

At the end of Step 1, each DO_i pulls its arm once, and initializes its variables s_i and n_i.

4.1.2 CORE OF SAMBA (FIGURE 5(B))

The core of SAMBA (Steps 2-5) is executed for each time step $t \in [\![K+1, N]\!]$.

We first illustrate the core of SAMBA with the **UCB** algorithm, for $K = 3$. Assume that we are at $t = 68$, when $s_1 = 24, s_2 = 10, s_3 = 2$, and $n_1 = 33, n_2 = 24, n_3 = 10$. At

```
/* Parameter setup */
receive N, A, seed_{α_t} from Controller (1)
r ← pull(i)
s_i ← r
n_i ← 1
/* Core of SAMBA */
for t ∈ [[K+1, N]]
  α_t ← nextValueFromSeed(seed_{α_t}, t)
  v_{i,t} ← ComputeScore(A, t, s_i, n_i)
  send Enc(α_t v_{i,t}) to Controller (2)
  receive Enc(b_{i,t}) from Controller (5)
  if Dec(Enc(b_{i,t})) = 1
    r ← pull(i)
    s_i ← s_i + r
    n_i ← n_i + 1
/* Cumulative reward computation */
send E(s_i) to Controller (6)
```

(a) Pseudocode of DO_i.

```
/* Parameter setup */
receive N, A from DC (0)
send N, A to Comp (1)
seed_{α_t} ← new random integer
for i ∈ [[K]]
  send N, A, seed_{α_t} to DO_i (1)
/* Core of SAMBA */
for t ∈ [[K+1, N]]
  for i ∈ [[K]]
    receive Enc(α_t · v_{i,t}) from DO_i (2)
  σ_t ← new random permutation
  send σ_t([Enc(α_t · v_{i,t})]_{i∈[[K]]}) to Comp (3)
  receive σ_t([Enc(b_{i,t})]_{i∈[[K]]}) from Comp (4)
  for i ∈ [[K]]
    send σ_t^{-1}(σ_t(Enc(b_{i,t}))) to DO_i (5)
/* Cumulative reward computation */
for i ∈ [[K]]
  receive E(s_i) from DO_i (6)
send E(s_1) · ... · E(s_K) to DC (7)
```

(b) Pseudocode of Controller.

```
/* Parameter setup */
receive N, A from Controller (1)
/* Core of SAMBA */
for t ∈ [[K+1, N]]
  receive σ_t([Enc(α_t · v_{i,t})]_{i∈[[K]]}) from Controller (3)
  V ← σ_t([Dec(Enc(α_t · v_{i,t}))]_{i∈[[K]]})
  σ_t(i_m) ← SelectArm(A, t, V)
  send σ_t([Enc(1_{i=σ_t(i_m)})]_{i∈[[K]]}) to Controller (4)
```

(c) Pseudocode of Comp.

```
/* Parameter setup */
send N, A to Controller (0)
/* Cumulative reward computation */
receive E(Σ_{i=1}^{K} s_i) from Controller (7)
```

(d) Pseudocode of DC.

Figure 6: Pseudocode of SAMBA participants. Numbers in () correspond to Steps in Figure 5.

Step 2, each data owner DO_i computes $B_i = \frac{s_i}{n_i} + \sqrt{\frac{2\ln(t)}{n_i}}$, as presented in Figure 4. Hence, we get $B_1 \approx 1.23$, $B_2 \approx 1.01$ and $B_3 \approx 1.12$. The random seed $seed_{\alpha_t}$ is useful such that all DO_i independently draw the same random mask needed to hide the real arm scores when computing the argmax. Using $seed_{\alpha_t}$, assume we draw $\alpha_{68} = 0.5$, hence each DO_i needs to multiply its B_i with α_{68} before encrypting with Enc and sending to Controller. Hence, Controller receives $\mathsf{Enc}(0.62)$ from DO_1, $\mathsf{Enc}(0.5)$ from DO_2, and $\mathsf{Enc}(0.56)$ from DO_3. This ends the Step 2 in Figure 5(b).

When receiving the aforementioned messages, Controller is not able to decrypt because it does not know the AES key used in Enc (we recall that this key is known only by Comp and all DO_i). Before forwarding the messages to Comp that is able to decrypt and compare scores, Controller permutes the messages in order to hide from Comp the correspondence

between a score and the DO_i that produced it. We stress that the permutation σ_t, which is *pseudo-random*, changes at each time step t because we want to avoid that Comp observes patterns where there is an arm much better than the others. By continuing on our example, assume that $\sigma_{68}(1) = 2, \sigma_{68}(2) = 3, \sigma_{68}(3) = 1$. Hence, at Step 3, $\mathsf{Controller}$ sends to Comp the list $[\mathsf{Enc}(0.56), \mathsf{Enc}(0.62), \mathsf{Enc}(0.5)]$. Then, at Step 4, Comp decrypts and computes argmax $= 2$, hence it sends to $\mathsf{Controller}$ the permuted list of pulling bits $[\mathsf{Enc}(0), \mathsf{Enc}(1), \mathsf{Enc}(0)]$. At Step 5, $\mathsf{Controller}$ inverts the permutation and sends $\mathsf{Enc}(1)$ to DO_1, and $\mathsf{Enc}(0)$ to DO_2 and DO_3. Then, each DO_i can decrypt its pulling bit. In particular, DO_1 can decrypt $\mathsf{Enc}(1)$, by computing $\mathsf{Dec}(\mathsf{Enc}(1)) = 1$. Since the pulling bit is 1, DO_1 pulls its arm and gets reward 1 at time step $t = 68$. We update arm variables: $s_1 = 25$ and $n_1 = 34$. No variable is updated for DO_2 or DO_3, who both had pulling bits 0. For $t = 69$ and until the end of the budget, Steps 2-5 are repeated.

Plugging **Thompson Sampling** in SAMBA is done exactly as for UCB, the only change being the arm score function (cf. Figure 4). As for **UCB**, no additional parameter is needed for **Thompson Sampling**.

Plugging **ε-greedy** in SAMBA is also immediate. We recall that for **ε-greedy** the learning agent *explores* by choosing randomly the arm to pull at time t with probability ε and *exploits* by choosing the best arm to pull at time t with probability $1 - \varepsilon$. Next, we also give details of the steps performed in SAMBA for securing **ε-greedy** for a time step t. We recall that we have as specific parameter $\varepsilon \in [0, 1]$ which gives the probability to *explore*, which can be either fixed or decreasing according to a time-depending function f_ε. We need another random seed, denoted $seed_\varepsilon$, that should be generated at Step 0 and send by $\mathsf{Controller}$ to each DO_i at Step 1. The random seed $seed_\varepsilon$ is useful such that at each time step all DO_i independently draw the same random number $x \in [0, 1]$, and compare it to ε in order to decide whether we shall *explore* or *exploit* at some time step i.e., if $x \leq \varepsilon$ then *explore*, else *exploit*.

For a concrete example, suppose that for $K = 3$ we have $\varepsilon = 0.1$ (thus, a learning agent *explores* when choosing the arm to pull at time t with probability 0.1 and *exploits* when choosing the arm to pull at time t with probability 0.9). Using $seed_\varepsilon$, assume we draw $x = 0.7$, hence we need to *exploit*. As shown in Figure 4, for *exploit* in **ε-greedy** we compute the empirical means $\frac{s_i}{n_i}$ for each DO_i, then pull the arm corresponding to the largest empirical mean. This is done in Steps 2-5, exactly as exemplified for **UCB**.

Alternatively, assume that using $seed_\varepsilon$ we draw $x = 0.04$, hence we need to *explore* i.e., randomly choose an arm. To do so, at Step 2, each DO_i encrypts the same number, that we set to 0 without loss of generality. At Step 3, $\mathsf{Controller}$ still permutes the list of encrypted masked values received from DO_i, before sending it to Comp. At Step 4, Comp computes an argmax on a permuted list containing K identical values, then sends the corresponding pulling bits to each DO_i and the protocol continues as previously until the end of Step 5. Due to the random permutation, all arms have equal chance of being selected for this *explore* step and the information about the arm that was chosen does not leak.

We continue with an example for **Softmax**, where the next pulled arm is chosen as a probability matching based on a Boltzmann distribution with temperature parameter τ. As shown in Figure 4, the probability to pull DO_i is $\frac{e^{\hat{\mu}_i/\tau}}{\sum_{j=1}^{K} e^{\hat{\mu}_j/\tau}}$. For the computations needed in the arm selection of **Softmax**, the frontier between local and global computations is

less clear. We remind that to easily plug-in an algorithm in Samba, one must be able to compute the score of a DO_i without seeing the variables of the other $\mathsf{DO}_{j\neq i}$. The Samba implementation of **Softmax** requires: (i) at Step 2, each DO_i computes $e^{\widehat{\mu}_i/\tau}$ and then applies a mask as explained previously, and (ii) at Step 4, Comp computes the sum and subsequent fractions after receiving all masked probabilities before sending the pulling bits to Controller. Thanks to the proportion-preserving mask, this operation does not modify the computed probabilities. Hence, Comp can recover the same probability matching as in the standard algorithm. All other steps are done as in the previous examples.

For clarity, we also provide a concrete example of the functioning of **Softmax** within Samba. For simplicity, we do not explicitly write the cryptographic functions in the following example. For every arm $i \in [\![K]\!]$, we denote the score $e^{\widehat{\mu}_i/\tau}$ by v_i. We assume a setting with $K = 3$ and $\tau = 0.1$. At an arbitrary time $t = 98$, assume we have $s_1 = 49, s_2 = 9, s_3 = 1$ and $n_1 = 68, n_2 = 24, n_3 = 5$, hence $\widehat{\mu}_1 = s_1/n_1 = 49/68 = 0.72, \widehat{\mu}_2 = 0.38$ and $\widehat{\mu}_3 = 0.2$. At Step 2, the DO_i compute $v_1 = e^{\widehat{\mu}_1/\tau} = e^{0.72/0.1} = 1339.43$, $v_2 = 44.7$, and $v_3 = 7.39$, respectively. Each DO_i uses the same mask, say $\alpha_{98} = 0.15$ (coming from seed $seed_{\alpha_t}$), and sends to Controller his masked value $\alpha_{98}v_1 = 200.91$, $\alpha_{98}v_2 = 6.71$, and $\alpha_{98}v_3 = 1.11$, respectively. At Step 3, Controller permutes the list of received values $[200.91, 6.71, 1.11]$ with a randomly drawn σ_{98} and sends the result to Comp, assume $[1.11, 6.71, 200.91]$. At Step 4, Comp divides each received element by $s = 1.11 + 6.71 + 200.91 = 208.73$, which produces the probability matching list $[0.01, 0.03, 0.96]$. Each value in the aforementioned list is the probability of an arm to be chosen e.g., the last arm has probability 96% to be chosen, and assume that this arm is actually chosen. Hence, Comp sends the list of pulling bits $[0, 0, 1]$ to Controller who, at Step 5, inverts the permutation $\sigma_{98}^{-1}([0, 0, 1]) = [1, 0, 0]$, and sends each pulling bit to the right DO_i.

Abstracting Score Computation and Arm Selection. To implement the aforementioned algorithms in the Samba generic protocol, we simply need to instantiate the two abstract functions *ComputeScore* and *SelectArm* from Figure 6:

***ComputeScore*: UCB: return** $\hat{\mu}_i + \sqrt{\frac{2\ln(t)}{n_i}}$
Thompson Sampling: return θ_i cf. Figure 4
ε**-greedy: return** $\hat{\mu}_i$ or **return** 0 (depends on ε, $seed_\varepsilon$)
Softmax: return $e^{\widehat{\mu}_i/\tau}$

***SelectArm*: UCB, Thompson Sampling,** ε**-greedy: return** $\arg\max_{\sigma_t(i)\in[\![K]\!]}(v_{\sigma_t(i),t})$
Softmax: return arm $\sigma_t(i)$ with probability $\frac{v_{\sigma_t(i),t}}{\sum_{j=1}^{K} v_{\sigma_t(j),t}}$

We additionally point out that the multiplicative mask α_t can be replaced by any order- and proportion- preserving function, without changing the theoretical properties of Samba (cf. Section 5). We chose to rely on a multiplicative mask in the presentation of Samba for the sake of simplicity.

4.1.3 Cumulative Reward Computation (Figure 5(c))

When the budget has been spent (that is, after observing N rewards), each DO_i can compute its final cumulative reward s_i. We must now communicate $\sum_{i=1}^{K} s_i$ to the Data Customer (DC) in a secure way. To do so, we use partial homomorphic encryption within the last two steps (Steps 5 and 6) of Samba, as depicted in Figure 5(c).

Namely, at Step 5, each DO_i sends $\mathcal{E}(s_i)$ to the Controller, who does not have the private key needed to decrypt it.

At Step 6, Controller computes $\mathcal{E}(s_1) \cdot \ldots \cdot \mathcal{E}(s_K) = \mathcal{E}(s_1 + \ldots + s_k)$, using the additive homomorphic property, and sends the results to DC, who has the private key needed to decrypt it and thus obtains the cumulative reward.

4.2 Extension

Pursuit is more challenging because it requires both argmax and probability matching to choose the next arm to be pulled cf. Figure 4. Indeed, for all algorithms instantiated until now, we had a single iteration over Steps 2-5, where we computed either an argmax (ε-**greedy**, **UCB**, **Thompson Sampling**) or a probability matching (**Softmax**). Consequently, to instantiate **Pursuit** in SAMBA, we need two iterations at each time step: a first iteration to compute the argmax among all empirical means $\widehat{\mu}_i$, and a iteration to compute a probability matching based on $p_{i,t}$.

To illustrate the SAMBA instantiation of **Pursuit**, we take an example where $K = 3$, and after the initialization (i.e., pull each arm once), $s_1 = 1, s_2 = s_3 = 0, n_1 = n_2 = n_3 = 1$, and $p_{1,3} = p_{2,3} = p_{3,3} = 1/3$. We are at $t = K + 1 = 4$ and let $\beta = 0.1$. The two iterations work as follows:

1. The first iteration computes probabilities $p_{i,4}$. Each DO_i sends its encrypted masked $\widehat{\mu}_i$ to Controller, which then forwards to Comp. Then, Comp computes an argmax exactly as explained in Section 4.1.2. On our example, arm 1 has probability to be pulled: $p_{1,4} = 1/3 + 0.1 \cdot (1 - 1/3) = 0.4$, while the other arms have probabilities to be pulled: $p_{2,4} = p_{3,4} = 1/3 + 0.1 \cdot (0 - 1/3) = 0.3$.
2. The second iteration computes the arm to be pulled according to a probability matching. Each DO_i sends his encrypted masked probability $p_{i,4}$ to Controller, which then forwards to Comp. Then, Comp randomly draws an arm according the the probabilities $p_{i,4}$, exactly as explained in Section 4.1.2 for **Softmax**.

To generalize the aforementioned ideas in order to instantiate arbitrary bandit algorithms that need more iterations over Steps 2–5 of SAMBA, the main required modification is an additional algorithm-dependent parameter $nb_{\mathcal{A}}$. The parameter $nb_{\mathcal{A}} = 2$ for **Pursuit** and was by default =1 for all bandit algorithms seen in Section 4.1.2. In the pseudocode of Controller, Comp, and DO_i in Figure 6, we simply need to add another loop "**for** $l \in [\![nb_{\mathcal{A}}]\!]$" just below each "**for** $t \in [\![K+1, N]\!]$". The semantics of the bits sent by Comp is different compared to the case of a single iteration. More precisely, only at the last iteration the bits are effectively used to pull an arm, whereas for the iterations $l \in [\![nb_{\mathcal{A}} - 1]\!]$ the bits are simply used to compute the score needed for iteration $l + 1$. Finally, we need to add the parameter $l \in [\![nb_{\mathcal{A}}]\!]$ to abstract functions *ComputeScore* and *SelectArm*, which for **Pursuit** are:

***ComputeScore*:**
if l=1 **return** $\widehat{\mu}_i$
else return $p_{i,t}$ cf. Figure 4

***SelectArm*:**
if l=1 **return** $\arg\max_{\sigma_t(i) \in [\![K]\!]}(v_{\sigma_t(i),t})$
else return arm $\sigma_t(i)$ with probability $\frac{v_{\sigma_t(i),t}}{\sum_{j=1}^{K} v_{\sigma_t(j),t}}$

One can plug in Samba virtually any bandit algorithm having an arbitrary number of iterations, as long as each iteration computes an argmax or a probability matching. We discuss the genericity of Samba in Section 5.1.

5. Theoretical Analysis

We analyze the *genericity*, *correctness*, *security*, and *complexity* of Samba in Section 5.1, 5.2, 5.3, and 5.4, respectively. Some proofs are omitted here and are available in the Appendix[4].

5.1 Genericity

In Section 4.1.2, we explained how to instantiate in Samba the algorithms ε**-greedy**, **UCB**, **Thompson Sampling**, and **Softmax**, each of them requiring an iteration over the core of Samba; then, in Section 4.2, we explained how to instantiate Samba with the **Pursuit** algorithm, which requires two such iterations. While presenting the Samba instantiations of the aforementioned algorithms, we introduced the abstract functions *ComputeScore* (executed by each DO_i) and *SelectArm* (executed by Comp). Intuitively, any cumulative reward maximization algorithm can be instantiated in Samba as long as it can be reduced to a federated version where only functions *ComputeScore* and *SelectArm* need to be specified. To characterize the genericity of Samba, we detail the properties that an algorithm should satisfy in order to be instantiated in Samba.

Theorem 1. *A standard cumulative reward maximization algorithm $\mathcal{A}$ can* be instantiated in Samba *if the computations of $\mathcal{A}$ can be distributed using $nb_{\mathcal{A}}$ iterations over the core of* Samba *such that the following two properties hold:*

1. Arm score locality*: At each time step $t \in [\![K+1, N]\!]$ and iteration $l \in [\![nb_{\mathcal{A}}]\!]$, DO_i can evaluate the function* ComputeScore *based only on its local variables n_i and s_i.*
2. Oblivious arm selection*: At each time step $t \in [\![K+1, N]\!]$ and iteration $l \in [\![nb_{\mathcal{A}}]\!]$, Comp can evaluate the function* SelectArm *based only on the current list $\mathcal{V}$ of permuted masked arm scores.*

Intuitively, the first property states that each arm score depends only on the local variables of its data owner and not on data stored by other data owners. Moreover, the second property states that the only node that has access to non-encrypted data (i.e., Comp) sees such data distorted with permutations and masks that change at each iteration. Thus, Comp cannot learn the local variables pertaining to some arm. The aforementioned properties are coherent with the secure federated learning paradigm because the data remains locally stored and the central orchestration server cannot learn more than the minimum amount of data needed to perform its tasks. We detail the security properties in Section 5.3.

Non-instantiable Algorithms. For each of the two conditions from Theorem 1, we give an example of a bandit algorithm that cannot be instantiated in Samba.

For the first condition, the example is **Reinforcement Comparison** (Sutton & Barto, 2018), which maintains a preference $\pi_{i,t}$ for each arm $i \in [\![K]\!]$. At time step t, the probability to select arm i is given by $p_{i,t} = \frac{e^{\pi_{i,t}}}{\sum_{j=1}^{K} e^{\pi_{j,t}}}$. Suppose that at time step t, the arm i has

4. https://raw.githubusercontent.com/gamarcad/paper-samba-code/master/pdf/appendix.pdf

been selected and generated a reward r_t. Then, the preference for arm i is updated as $\pi_{i,t+1} = \pi_{i,t} + \beta(r_t - \bar{r}_t)$, where $\bar{r}_t$ is the *reference reward* that depends on all arms. At the end of each time step t, $\bar{r}_{t+1}$ is updated as $\bar{r}_{t+1} = (1-\alpha)\bar{r}_t + \alpha r_t$. Both α and β are learning rates between 0 and 1. To sum up, the probability of selecting an arm i in **Reinforcement Comparison** depends on $\bar{r}_t$, which is obtained by averaging over the observed rewards $r_1, \ldots, r_t$, which come from all K arms. Therefore, computing the score of an arm does not respect the *Arm score locality* property required by Theorem 1.

For the second condition, we are not aware of any off-the-shelf cumulative reward maximization algorithm for stochastic Bernoulli bandits that cannot be instantiated in SAMBA. Such an algorithm should decide on the selected arm at time step t in a way that is not compatible with the random masks and/or permutations from SAMBA. For the sake of example, imagine an arm selection strategy that (i) filters the list of arms by keeping only the ones with odd indices for odd time step t and even indices for even time step t, (ii) filters the remaining arms by keeping only those with empirical means $\hat{\mu}_i$ in the interval [0.5, 1], and (iii) returns the argmax of $\hat{\mu}_i$ among the arms that survived the filters (i) and (ii). Note that (i) cannot be ensured by Comp in the presence of the random permutations, whereas (ii) cannot be ensured by Comp in the presence of random masks.

We stress that all algorithms mentioned in this section (instantiable in SAMBA or not) consider Bernoulli reward distributions cf. Section 3.1. We refer to Section 7 for a discussion on the challenges to build a potential SAMBA-inspired framework for different bandit reward models e.g., linear bandits.

5.2 Correctness

We show that the instantiation of a cumulative reward maximization algorithm in SAMBA does not change its output.

Theorem 2. *Take a standard cumulative reward maximization algorithm $\mathcal{A}$ and its federated version $\mathcal{A}'$ instantiated in* SAMBA *(cf. Theorem 1), both initialized with the same seeds for the needed randomness (e.g., pull function, argmax when ties have to be broken). For every N, K, and $\mu_1, \ldots, \mu_K$, algorithms $\mathcal{A}$ and $\mathcal{A}'$ return the same cumulative reward.*

Proof. If we take $\mathcal{A}'$ and remove all encryption/decryptions (both AES-GCM and Paillier), we obtain $\mathcal{A}^{\text{no-enc}}$ where all messages are communicated in clear between participants. Algorithms $\mathcal{A}'$ and $\mathcal{A}^{\text{no-enc}}$ output the same result because of the consistency property of the chosen cryptographic schemes i.e., if we encrypt a message M using Enc (or $\mathcal{E}$, respectively) to obtain a ciphertext C, then if we decrypt C using Dec (or $\mathcal{D}$, respectively), then we obtain exactly M.

Take $\mathcal{A}^{\text{no-enc}}$ and remove the distribution of data computation among participants, which yields $\mathcal{A}^{\text{no-dist}}$ that returns the same result as $\mathcal{A}^{\text{no-enc}}$ because both algorithms do exactly the same computations in the same order.

Take $\mathcal{A}^{\text{no-dist}}$ and remove the use of the permutation σ_t, which yields $\mathcal{A}^{\text{no-perm}}$ that returns the same result as $\mathcal{A}^{\text{no-dist}}$ because the random permutation σ_t reduces to the randomness in the argmax function used in standard bandit algorithms when ties have to be broken among arms with the same score. As pointed out in the theorem statement, this is ensured by fixing the same random seed.

Data \ Participant	DO_i	DC	Server		Ext
			Comp	Controller	
Cumulative reward		X		$\mathcal{E}$	$\mathcal{E}$
Sum of rewards and number of pulls for DO_i	X		α_t	$\mathcal{E}$	$\mathcal{E}$
Sum of rewards and number of pulls for $\mathsf{DO}_{j\neq i}$			α_t	$\mathcal{E}$	$\mathcal{E}$
Arm pulled at time step t	$\mathsf{X}^\star$		σ_t	Enc	Enc
Reward at time step t	$\mathsf{X}^\star$				

Figure 7: *Security properties of* Samba. The X means that the participant can see in clear the concerned piece of data, with $\star$ = only if DO_i is pulled at time step t. Ext means an external network observer having access to all messages exchanged between participants. In the cells without X or $\mathsf{X}^\star$, we indicate the technique that we used to prevent the participant from seeing in clear the concerned data: Paillier encryption ($\mathcal{E}$), AES-GCM encryption (Enc), random masks (α_t), and random permutations (σ_t). A grayed cell means that the concerned participant does not see any message about the concerned piece of data.

Take $\mathcal{A}^{\text{no-perm}}$ and remove the use of the multiplicative random mask α_t, which yields $\mathcal{A}$ that returns the same result as $\mathcal{A}^{\text{no-perm}}$ because the mask has no impact on the arm selection strategy. Indeed, this holds $\forall \alpha_t \in \mathbb{R}_+^*$ regardless of the computations needed in *SelectArm*:

- *Order-preserving* (needed if *SelectArm* does an argmax): given arm scores v_1 and v_2, it holds that $v_1 \leq v_2$ iff $\alpha_t v_1 \leq \alpha_t v_2$.
- *Proportion-preserving* (needed if *SelectArm* does a probability matching): given arm scores $v_1, \ldots, v_K$, for every $i \in [\![K]\!]$ it holds that

$$\frac{\alpha_t v_i}{\sum_{j=1}^{K}(\alpha_t v_j)} = \frac{\alpha_t v_i}{\alpha_t(\sum_{j=1}^{K} v_j)} = \frac{v_i}{\sum_{j=1}^{K} v_j}$$

Through the aforementioned sequence of reductions, we reduced the federated algorithm $\mathcal{A}'$ to its standard non-federated version $\mathcal{A}$, such that at each time step t, both $\mathcal{A}$ and $\mathcal{A}'$ select exactly the same arm to be pulled. In addition, as pointed out in the theorem statement, we fix the same random seed for generating random rewards, hence pulling some arm i at time step t in all aforementioned algorithms ($\mathcal{A}', \mathcal{A}^{\text{no-enc}}, \mathcal{A}^{\text{no-dist}}, \mathcal{A}^{\text{no-perm}}, \mathcal{A}$) returns exactly the same reward. By summing up all these rewards, we obtain exactly the same cumulative reward. □

5.3 Security

In Figure 7, we summarize the *security properties* of Samba, which subsume their previous presentation in Figure 2. In particular, note that the server is divided in two independent nodes Comp and Controller.

Theorem 3. *Take a standard cumulative reward maximization algorithm $\mathcal{A}$ and its federated version $\mathcal{A}'$ instantiated in* Samba *(cf. Theorem 1). It holds that $\mathcal{A}'$ satisfies the security properties summarized in Figure 7 if the primitives used in* Samba *are secure.*

To give an intuition on why some security property holds, we indicate in Figure 7 the technique that we used to prevent some participant from seeing in clear some concerned data. We make available in the Online Appendix the formal statements and proofs of the aforementioned security properties for each participant. We give a few more details next.

Security of DO_i. By construction of SAMBA, each DO_i knows data about its arm (e.g., its sum of rewards s_i) and knows the time step and the reward when its arm is pulled by the cumulative reward maximization algorithm. Also by construction of SAMBA, DO_i does not see any other data about the cumulative reward except its s_i, or about other arms at the time steps when DO_i is not pulled. This means that at a time step when it is not pulled, DO_i cannot guess the pulled DO_{i_m} with probability better than random. Moreover, DO_i cannot guess with probability better than random the sum of rewards of any other arm.

Security of DC. By construction of SAMBA, only DC sees in clear the cumulative reward for which she spends a budget. This happens after Step 7 when she decrypts the cumulative reward computed directly in the encrypted domain by Controller. Moreover, DC sees no other data concerning the execution of the cumulative reward maximization algorithm. In particular, DC cannot guess with probability better than random the arm that is pulled at some time step or the partial sum of rewards produced by some arm.

Security of Comp. By construction of SAMBA, Comp is only involved at Steps 3 and 4, when it sees in clear arm scores that have been (i) multiplied with a random mask α_t and (ii) permuted using a permutation σ_t, where both α_t and σ_t change at each iteration. Hence, Comp cannot guess with probability better than random the real arm scores (hence neither the number of pulls nor the rewards produced by some arm) because of (i), or which is the real arm index pulled at some time step because of (ii).

Security of Controller and of external observers. By construction of SAMBA, Controller knows the permuted masked scores of all DO_i given at Step 2, the selection bits of all DO_i sent by Comp at Step 4, both encrypted with AES-GCM. Additionally, at Step 6, Controller knows the partial cumulative rewards encrypted with Paillier's cryptosystem. By seeing only data encrypted with AES-GCM (Enc) or Paillier ($\mathcal{E}$), Controller cannot guess with probability better than random the cumulative reward, the sum of rewards and number of pulls for some arm, or the arm pulled at some time step. The security proofs of Controller are the most technical among all participants because they require proofs by contradiction as typically done in cryptographic proofs that rely on the hypothesis that the primitives (in our case AES-GCM or Paillier) are secure. To give an idea on how this kind of proof works, we need to introduce some additional notation, such as:

- $s_{i,t}$ = the sum of rewards obtained by arm i until time step t.
- $data_x^t$ = the data to which participant x has access until time step t, where x can be virtually any participant from Figure 6 or the external observer (ext). If t is omitted, this denotes the data to which x has access at the end of the execution of SAMBA.
- $\mathsf{Adv}^{pb(.)}(d)$ = the answer of a Probabilistic Polynomial-Time (PPT) adversary Adv (we defined such an adversary in Section 3.2) that knows d and tries to solve the problem pb. The problem pb may also take some *input*.

An example of security theorem for Controller is: *For an arm $i \in [\![K]\!]$ and a time step $t \in [\![K+1, N]\!]$, an honest-but-curious* Controller *cannot guess $s_{i,t}$, given $data^t_{\mathsf{Controller}}$, with*

a probability better than random. More precisely, for all PPT adversaries Adv*,*

$$\left|\Pr\left[(i,\hat{s}_{i,t}) \leftarrow \mathsf{Adv}^{sum(t)}(data^t_{\mathsf{Controller}}); \hat{s}_{i,t} = s_{i,t}\right] - p_S(s_{i,t})\right|$$

is negligible in λ*, where* $\mathsf{Adv}^{sum(t)}(data^t_{\mathsf{Controller}})$ *returns* $(i,\hat{s}_{i,t})$ *in which* $\hat{s}_{i,t}$ *is* Adv*'s guess on* $s_{i,t}$ *for the arm* i *(chosen by* Adv*), and* $p_S(s_{i,t})$ *is the probability of obtaining a sum of rewards* $s_{i,t}$ *for an arm* i *at time step* t*.* The subtlety is that the sum guessed by Adv is based on input data consisting of all the (encrypted) data that Controller has access to, whereas p_S is just the probability of randomly guessing a correct sum without having access to any data. To prove such a theorem, we need to assume toward a contradiction that there exists a PPT adversary Adv who is able to guess a correct sum of rewards from $data^t_{\mathsf{Controller}}$ with a non-negligible advantage. After quantifying the implications of such an assumption, we find that there exists a PPT adversary able to break the security of the cryptographic primitive used to encrypt $data^t_{\mathsf{Controller}}$, which contradicts the fact that the employed primitives are secure (in our Samba theorems, we rely on AES-GCM or Paillier).

Regarding an external observer (*ext*) that has access to all messages exchanged over the network between Samba participants, the proofs are very similar to the case of Controller because all the exchanged messages are encrypted with AES-GCM or Paillier.

5.4 Complexity

We quantify the cryptographic overhead as well as the computation cost of Samba to conclude that both complexity measures are linear in N and K.

Theorem 4. *Take a standard cumulative reward maximization algorithm* $\mathcal{A}$ *and its federated version* $\mathcal{A}'$ *instantiated in* Samba *(cf. Theorem 1).*

1. $\mathcal{A}'$ *requires* $O(NK)$ *AES-GCM encryptions and decryptions,* K *Paillier encryptions, and one Paillier decryption.*
2. $\mathcal{A}'$ *has communication cost of* $O(NK)$ *messages, assuming as unit of measurement the size of an encrypted number.*

Proof. To prove *1)*, we quantify the number of cryptographic operations required by Samba. The parameter setup (Steps 0, 1) requires no cryptography. The core of Samba (Steps 2, 3, 4, 5) requires $2K$ AES-GCM encryptions and $2K$ AES-GCM decryptions and is repeated $(N-K)nb_{\mathcal{A}}$ times. Since the number of iterations is constant (=1 or 2 in all algorithms instantiated so far in Samba), by summing up we obtain $O(NK)$ AES-GCM encryptions and decryptions at the core of Samba. Finally, the cumulative reward computation (Steps 6, 7) requires K Paillier encryptions and one Paillier decryption.

To prove *2)*, we quantify the size of messages communicated among Samba participants. The parameter setup (Steps 0, 1) requires a constant number of messages: ≥ 7 (cf. Figure 5), but may additionally include algorithm-specific parameters (e.g., $\varepsilon, f_\varepsilon, \tau, \beta$ cf. Section 4.1.1). The core of Samba (Steps 2, 3, 4, 5) communicates 4 lists of size K and is repeated $(N-K)nb_{\mathcal{A}}$ times. Finally, the cumulative reward computation (Steps 6, 7) communicates $K+1$ messages. In total, the communication cost is of $O(NK)$. □

It is a desirable feature that the number of Paillier operations does not depend on the budget N because N is typically larger than the number of arms K, and AES-GCM is much

faster than Paillier. Moreover, when quantifying the computation time, we looked only at the cryptographic operations because they dominate the time required for the other computations. Our proof-of-concept empirical evaluation in Section 6 confirms the aforementioned observations.

Algorithm-Specific Optimizations. We point out that the practical complexity (both cryptographic and communication) of some bandit algorithm instantiated in SAMBA can be diminished by leveraging algorithm-specific properties, while guaranteeing the same correctness and security properties. Among the algorithms instantiated so far in SAMBA, this is true for the exploration time steps of ε-**greedy**, which occur with probability ε (either fixed or decreasing). In the exploration time steps of ε-**greedy**, the next arm to be pulled is selected randomly. Hence, it is not necessary that DO_i send K encrypted messages to Controller at Step 2. Instead, we can let Controller draw a random number as score for all arms before sending a list to Comp at Step 3. In this case, Comp randomly chooses the next arm to be pulled, the randomness in the argmax being simulated by the randomness from the permutation σ_t. Then, Steps 4 and 5 are as usually, in particular Controller cannot decrypt which arm has to be pulled because the pulling bits are encrypted with an AES key known only by Comp and DO_i. Although such an optimization of $O(K)$ does not reduce the overall asymptotic complexity of $O(NK)$, we implemented it in our prototype to be able to be slightly more efficient without jeopardizing the theoretical properties of SAMBA.

6. Experiments

We present a proof-of-concept empirical study of SAMBA. We explain the experimental setting in Section 6.1 and we discuss the results in Section 6.2.

6.1 Setting

We study the feasibility and scalability of SAMBA through a proof-of-concept experimental study using two datasets that contain user ratings for movies i.e., MovieLens (Harper & Konstan, 2016) and jokes i.e., Jester (Goldberg, Roeder, Gupta, & Perkins, 2001), respectively. The use of ratings for testing bandit algorithms is natural since the items (movies or jokes) in the dataset correspond to the bandits arms, and a user rating given for a particular item corresponds to the reward obtained when the corresponding arm is chosen. We preprocess the datasets similarly to an existing technique (Kohli, Salek, & Stoddard, 2013). Namely, for each item i, we compute the mean reward μ_i = (#of ratings above a threshold for item i) / (#of ratings received for item i). Without loss of generality, we consider the K items with largest mean rewards. Note that this experimental setting is typical for stochastic multi-armed bandit papers (Kohli et al., 2013; Ciucanu et al., 2020; Kuleshov & Precup, 2014; Shi & Shen, 2021). We tuned the algorithm-specific parameters $(\varepsilon, \tau, \beta)$ similarly to an existing technique (Kuleshov & Precup, 2014). The budget varies from $N = 5 \times 10^4$ to $N = 10^5$, and the number of arms from $K = 10$ to $K = 100$, which is coherent with the typical distribution scale of cross-silo federated learning (cf. Section 2.1).

We implemented the five algorithms from Figure 4, with both fixed and decreasing ε-greedy, which implies that we have a total number of six lines in each reported plot. We

used Python 3. For AES-GCM we used the *Cryptography* library[5] and keys of 256 bits. For Paillier, we used the *phe* library[6] in the default configuration with keys of 2048 bits. We did our experiments on a virtual machine running Ubuntu, located in a server with 8GB of RAM and 24 cores Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz. The reported points in the plots are averaged over 50 runs.

Given the related work positioning from Section 2.2, a direct empirical comparison with existing works on federated bandit algorithms with security/privacy guarantees would not be very useful, given the fundamental differences between our work and existing approaches. Indeed, (i) on the one hand, the running time of differentially-private federated bandit algorithms is roughly the same as for the standard algorithms and is not reported in their experiments; this is why such differentially-private works focus on quantifying the impact on the cumulative reward, (ii) on the other hand, our Samba instantiations of bandit algorithms are guaranteed to return the same cumulative rewards as the standard algorithms.

All details concerning our Samba prototype are available on a public GitHub repository[7], including our source code, the data, the generated results from which we obtained our plots, and scripts that allow to install the needed libraries and reproduce the entire workflow to generate our plots. We stress that due to the genericity of Samba, the prototype can be easily extended to other bandit algorithms that satisfy Theorem 1.

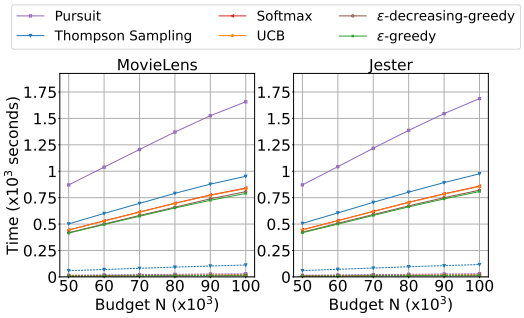
6.2 Results

We discuss the *scalability* of Samba w.r.t. the budget N (Figure 8(a)) and the number of arms K (Figure 8(b)). Our experimental results show that the execution time is linear in N and K, which matches the theoretical complexity introduced in Section 5.4. The overhead between the standard algorithms (shown in dotted lines) and their Samba instantiation is of at most an order of magnitude and comes from the cryptographic primitives. In particular, we recall that we have $O(NK)$ AES-GCM operations and $O(K)$ Paillier operations. Since Paillier operations are longer to evaluate that AES-GCM, we naturally observe that the lines from the scalability w.r.t. K (Figure 8(b)) increase more steeply than those from the scalability w.r.t. N (Figure 8(a)).

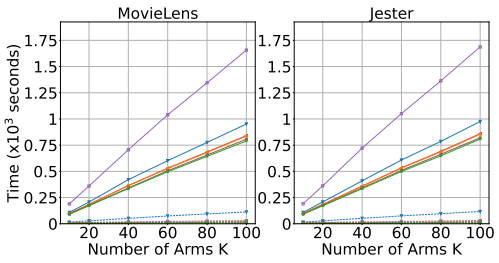
For both datasets, for the maximal considered input (i.e., budget $N = 10^5$ and $K = 100$ arms), the majority of the Samba algorithms take around 15 minutes. The only exception is **Pursuit**, whose execution time is double, which is expected because its Samba instantiation requires two iterations (as explained in Section 4.2). We believe that such a waiting time is reasonable for a data customer to obtain the cumulative reward, while ensuring security guarantees for all participants in the federated learning setting.

By comparing the left vs right plots in each of Figure 8(a) and 8(b), we observe that regardless the dataset, the shapes of the lines are exactly the same and in precisely the same hierarchy. This is also a consequence of the theoretical properties of Samba because its complexity depends on N and K, but not on the arm values $\mu_{1 \leq i \leq K}$. As explained in Section 6.1, only the μ_i change from a dataset to the other. The hierarchy between the times taken by the 5 bandit algorithms that require only one iteration (i.e., all except Pursuit)

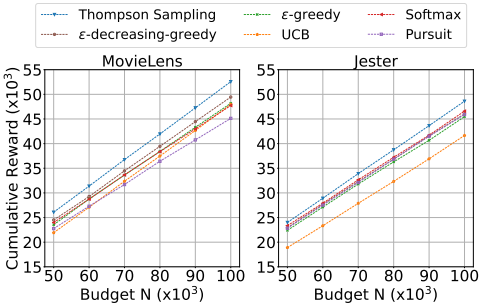
5. `https://pypi.org/project/cryptography/`
6. `https://pypi.org/project/phe/`
7. `https://github.com/gamarcad/paper-samba-code`



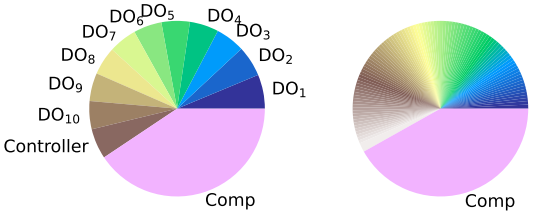
(a) Execution time: fixed K=100 and varying N.



(b) Execution time: fixed N=10^5 and varying K.



(c) Cumulative reward: fixed K=100 and varying N.



(d) Zoom on the execution time of SAMBA participants: N=10^5, and K=10 (left) and K=100 (right). On both plots, the share of DC is not shown because is close to 0. On the right, the 101 equal non-labeled shares are taken by $\mathsf{DO}_{1\leq i\leq 100}$ and Controller.

Figure 8: Experimental results of SAMBA.

is rather easy to explain based on their standard (non-secure non-federated) versions cf. Figure 4: **Thompson Sampling** is the slowest because sampling according to a Beta distribution is costly, ..., **ε-greedy** with fixed ε is the fastest because it does the simplest computations.

Even though the hierarchy of execution times is rather clear and predictable, we cannot state that there is a cumulative reward maximization algorithm that should always be chosen. Indeed, besides execution time, there is also a qualitative measure that is the actual cumulative reward that is returned, and which depends on the μ_i. We recall that cf. Theorem 2, the SAMBA instantiation of an algorithm is guaranteed to return the same cumulative reward as its standard (non-secure non-federated) version, hence SAMBA does not yield any quality loss w.r.t. the standard versions.

The two different datasets (MovieLens and Jester) yield different cumulative rewards (Figure 8(c)) for the instantiated algorithms, and different rankings in their performance, according to the associated ratings of items (aka arm values). Hence, we cannot say that there is algorithm clearly better or worst than the others. However, the SAMBA instantiation of **Thompson Sampling** seems to provide a good trade-off because it always produces the best rewards and its execution time is quite close to the other algorithms that require only one iteration over the core of SAMBA.

Lastly, we show in Figure 8(d) the shares of each Samba participant. Our observations are coherent with the role played by each participant, as visible from their involvement in the Samba's architecture in Figure 5. Comp is the most time-consuming node since it performs the most work, and for the same N, the share of Comp stays the same, even for different orders of magnitude of K. Each DO_i has an equal share, since they all perform the same amount of work. The share of Controller is roughly equivalent to that of an DO_i, whereas the share of DC is negligible.

To sum up, our proof-of-concept experiments confirm the theoretical complexity and show that the overhead of Samba is reasonable, hence our protocols are feasible.

7. Conclusions and Future Work

We tackled the problem of secure cumulative reward maximization in multi-armed bandits in a cross-silo federated learning setting. Under the orchestration of a central server, each data owner participating at the cumulative reward computation has the guarantee that its raw data is not seen by some other participant. We proposed Samba, a generic secure protocol that is able to easily transform multi-armed bandit algorithms in their secure federated version, while yielding the exact same cumulative reward as their standard (non-secure non-federated) version. To achieve Samba's security properties, we relied on secure multi-party computations and cryptographic schemes under the honest-but-curious threat model. Through a theoretical analysis and proof-of-concept experiments, we showed that the cryptographic overhead implied by Samba remains reasonable in practice.

System Demonstration and Web Interface. In addition to the fundamental contributions presented in this paper, we also implemented a Samba system demonstration (Marcadet, Ciucanu, Lafourcade, Soare, & Amer-Yahia, 2022) that is based on a Web interface simulating the Samba federated components. The user-friendly Samba Web interface allows data scientists to configure the end-to-end workflow of deploying a federated bandit algorithm, by examining the interaction between three key dimensions of federated bandits: cumulative reward, computation time, and security guarantees.

Future Work. We plan to extend Samba such that it provides security guarantees in more complex threat models and for more complex multi-armed bandit frameworks. More in general, using cryptography to ensure data security for machine learning algorithms is a promising, timely direction. We plan to pursue this direction and to design secure protocols useful for other machine learning models and applications.

According to a recent keynote from a security conference (Yung, 2015), an important factor that one should take into account when designing a secure multi-party computation protocol is "*Generality vs. specificity: Secure computation is a general scheme; in reality one has to choose an application, starting from a very real business need, and build the solution from the problem itself choosing the right tools, tuning protocol ideas into a reasonable solution, balancing security and privacy needs vs. other constraints.*" We stress that federated learning is a timely, dynamic topic pertaining for both artificial intelligence/machine learning and security/cryptography communities. One may see a particular class of AI/ML problems as the "*business need*" from the aforementioned keynote. For Samba, the considered problem setting that is relevant and novel for all concerned communities is cumulative

reward maximization in federated stochastic multi-armed bandits. We relied on secure multi-party computations in order to guarantee formally proven correctness and security properties, while yielding a reasonable cryptographic overhead. Moreover, Samba is generic in the sense that multiple algorithms can be easily plugged in, and we already show five such algorithms in the paper. Designing a generic framework with theoretical properties similar to Samba but pertaining for other bandit problems (e.g., for best arm identification) or different bandit models (e.g., for linear bandits) are completely different problem settings, which require the development of new specific secure protocols, as also emphasized above. Naturally, this is also the case for finding optimal strategies in the standard (non-federate, non-secure) version of bandits algorithms designed for a specific setting, which are no longer optimal for a different bandit problem or model.

For instance, in linear bandits, the bandit arms are vectors that are known by the learner, hence there is no need in hiding them as it was the case in Samba, where each bandit arm is known precisely by a data owner. Moreover, in linear bandits, the unknown environment is given by a parameter vector θ (unknown to the learner), which is common to all arms and is used to generate rewards by computing noisy scalar products with the arm vectors. Hence, a potential Samba-inspired framework for linear bandits should make data federation hypothesis different than Samba because the considered learning data are genuinely different. Towards this goal, we first need to find techniques for federating the storage of θ among several data owners before designing secure protocols that (i) enjoy security and correctness guarantees similar to those of Samba and (ii) are able to run cumulative reward maximization algorithms for linear bandits. The properties from Samba's genericity theorem (i.e., Arm score locality and Oblivious arm selection) will no longer be fitted for linear bandits because by construction, the linear bandit arms are connected by the global linear structure, whereas such connections between arms are not present in standard stochastic bandits as considered in Samba. Even without considering security and federated aspects, a linear bandit algorithm cannot be optimal if it does not leverage the global linear structure when designing the arm selection strategy. The challenge is thus to find the right genericity properties, specific to the linear bandit setting, that will allow cumulative reward maximization algorithms for linear bandits to be plugged in an adapted secure, federated framework that enjoys theoretical properties similar to those of Samba.

Acknowledgments

We thank the anonymous reviewers, whose suggestions helped us to improve our paper.

This work was mostly done while Radu Ciucanu, Gael Marcadet, and Marta Soare were affiliated with INSA Centre Val de Loire / Univ. Orléans / LIFO, France.

This work has been partially supported by MIAI@Grenoble Alpes (ANR-19-P3IA-0003), two projects funded by EU Horizon 2020 research and innovation programme (TAILOR under GA No 952215 and INODE under GA No 863410), and the French BPI project D4N.

References

AES (2001). Advanced Encryption Standard (AES). `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf`. FIPS Publication 197.

AES (2007). Recommendation for BlockCipher Modes of Operation:Galois/Counter Mode (GCM) and GMAC. `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf`. NIST Special Publication 800-38D.

Baldimtsi, F., & Ohrimenko, O. (2015). Sorting and Searching Behind the Curtain. In *Financial Cryptography*, pp. 127–146.

Bellare, M., Desai, A., Jokipii, E., & Rogaway, P. (1997). A Concrete Security Treatment of Symmetric Encryption. In *Symposium on Foundations of Computer Science (FOCS)*, pp. 394–403.

Cheon, J. H., Kim, D., & Kim, D. (2020). Efficient Homomorphic Comparison Methods with Optimal Complexity. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pp. 221–256.

Ciucanu, R., Delabrouille, A., Lafourcade, P., & Soare, M. (2020). Secure Cumulative Reward Maximization in Linear Stochastic Bandits. In *International Conference on Provable and Practical Security (ProvSec)*, pp. 257–277.

Ciucanu, R., Lafourcade, P., Lombard-Platet, M., & Soare, M. (2019). Secure Best Arm Identification in Multi-Armed Bandits. In *International Conference on Information Security Practice and Experience (ISPEC)*, pp. 152–171.

Ciucanu, R., Lafourcade, P., Lombard-Platet, M., & Soare, M. (2020). Secure Outsourcing of Multi-Armed Bandits. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 202–209.

Dubey, A., & Pentland, A. (2020). Differentially-Private Federated Linear Bandits. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Dwork, C., & Roth, A. (2014). The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, *9*(3-4), 211–407.

Garcelon, E., Perchet, V., & Pirotta, M. (2021). Homomorphically Encrypted Linear Contextual Bandit. *CoRR*, *abs/2103.09927*.

Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. In *Symposium on Theory of Computing (STOC)*, pp. 169–178.

Goldberg, K. Y., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, *4*(2), 133–151.

Goldreich, O. (2004). *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press.

Harper, F. M., & Konstan, J. A. (2016). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *5*(4), 19:1–19:19.

Huang, R., Wu, W., Yang, J., & Shen, C. (2021). Federated Linear Contextual Bandits. In *Conference on Neural Information Processing Systems (NIPS)*.

Kairouz, P., McMahan, H. B., & et al. (2021). Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning*, *14*(1–2), 1–210.

Kocsis, L., & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning (ECML)*.

Kohli, P., Salek, M., & Stoddard, G. (2013). A Fast Bandit Algorithm for Recommendation to Users With Heterogenous Tastes. In *AAAI Conference on Artificial Intelligence*.

Kuleshov, V., & Precup, D. (2014). Algorithms for Multi-Armed Bandit Problems. *CoRR*, *abs/1402.6028*.

Li, L., Chu, W., Langford, J., & Schapire, R. E. (2010). A Contextual-bandit Approach to Personalized News Article Recommendation. In *International Conference on World Wide Web (WWW)*.

Li, T., Song, L., & Fragouli, C. (2020). Federated Recommendation System via Differential Privacy. In *International Symposium on Information Theory (ISIT)*, pp. 2592–2597.

Marcadet, G., Ciucanu, R., Lafourcade, P., Soare, M., & Amer-Yahia, S. (2022). SAMBA: A System for Secure Federated Multi-Armed Bandits. In *IEEE International Conference on Data Engineering (ICDE) – Demo*.

Mitra, A., Hassani, H., & Pappas, G. (2021). Exploiting Heterogeneity in Robust Federated Best-Arm Identification. *CoRR*, *abs/2109.05700*.

Paillier, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pp. 223–238.

Russo, D., Van Roy, B., Kazerouni, A., Osband, I., & Wen, Z. (2018). A Tutorial on Thompson Sampling. *Foundations and Trends in Machine Learning*, *11*(1), 1–96.

Shi, C., & Shen, C. (2021). Federated Multi-Armed Bandits. In *AAAI Conference on Artificial Intelligence*.

Shi, C., Shen, C., & Yang, J. (2021a). Federated Multi-armed Bandits with Personalization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 2917–2925.

Shi, C., Xu, H., Xiong, W., & Shen, C. (2021b). (Almost) Free Incentivized Exploration from Decentralized Learning Agents. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second edition). The MIT Press.

Thompson, W. R. (1933). On the Likelihood that One Unknown probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, *25*, 285–294.

Yung, M. (2015). From Mental Poker to Core Business: Why and How to Deploy Secure Computation Protocols?. In *Conference on Computer and Communications Security (CCS)*, pp. 1–2.

Zhu, Z., Zhu, J., Liu, J., & Liu, Y. (2021). Federated Bandit: A Gossiping Approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *5*(1).