Design of Secure Multi-User Protocols : Application to Bandits, Ticketing and File Transfer

Gaël Marcadet

Design of Secure Multi-User Protocols

Application to Bandits, Ticketing and File Transfer

Thèse de Doctorat de l'Université Clermont-Auvergne
Spécialité: Informatique

École Doctorale Sciences Pour l'Ingénieur (SPI)
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
CNRS, UMR 6158, LIMOS, 63173 Aubière, France

Thèse présentée et soutenue le 5 Juillet 2024 par:

Gaël MARCADET

Composition du jury:

Jean-Guillaume DUMAS Professeur des Universités, LJK, Université Grenobles-Alpes	Président du Jury
Benjamin NGUYEN Professeur des Universités, LIFO, INSA Centre-Val de Loire	Rapporteur
Melek ÖNEN Maître de Conférences HDR, EURECOM	Rapporteuse
Ioana BOUREANU Professeure des Universités, SCCS, University of Surrey	Examinatrice
Chloé HÉBANT Cryptographe, COSMIAN	Examinatrice
Pascal LAFOURCADE Professeur des Universités, LIMOS, Université Clermont-Auvergne	Directeur de Thèse

LIMOS

UNIVERSITÉ Clermont Auvergne

Abstract

A cryptographic protocol establishes a series of interactions among users to deliver a given functionality while ensuring various properties, a protocol being considered secure when it successfully ensures all intended properties. Accomplishing these properties requires the need of cryptographic primitives, whose usage may entail computation overhead, limiting the scalability of the protocol. Throughout this manuscript, we focus on three problems dealing with multiple users.

The first contribution focuses on the design of a federated multi-armed bandits framework where a federation server, acting as a learning agent, sequentially pulls a bandit arm, the environment responding with a reward coming from an unknown distribution associated with the chosen bandit. In this contribution, we introduce Tango, a secure federated multi-armed bandits protocol fixing and extending our initial attempt Samba shown to be insecure. Tango is proved to prevent the federation server to learn the reward distribution, the obtained rewards and the pulled bandit arm, at the cost of a large computation overhead due to the usage of expensive cryptographic primitives. In the second part of this contribution, we introduce Salsa a secure federated multi-armed bandits protocol moving away from the blueprint of Samba and Tango, still preventing the federation server to learn sensitive data while achieving high-performance.

The second contribution of this manuscript addresses a problem involving a large number of users, since it concerns the design of a ticketing system. Indeed, despite the high-demand, these systems provide very restricted guarantees. For instance, one may easily resell a ticket twice. To go further, tickets are nominative, revealing the identity of the ticket's owner. Using standard cryptographic primitives, we propose two scalable ticketing systems called Applause and Spotlight, ensuring anonymity of users while featuring ticket purchasing, ticket refunding, ticket validation and ticket transferability. The difference between Applause and Spotlight lies in the ability to recover the identity of an attendee: In Applause, the anonymity of every user is guaranteed at any time, a property that still hold with Spotlight except for an additional third-party able to recover the identity of an attendee, at the cost of a slightly longer ticket validation.

Our third and final contribution deals with the problem of file transfer by broadcasting, which involves sharing a file with a group of users. The trivial solution of storing files on a single, publicly accessible server falls short for instance when the server is down or when the server handles a high number of requests. In this contribution, we introduce a universally composable and efficient protocol allowing one to share a file with a specified group of users while ensuring confidentiality, integrity of the file and sender authentication.

Résumé

Un protocole cryptographique établit une série d'interactions parmi des utilisateurs pour fournir une fonctionnalité donnée tout en garantissant diverses propriétés. Un protocole est considéré comme sécurisé lorsqu'il assure avec succès toutes les propriétés attendues. Pour ce faire, il utilise de primitives cryptographiques dont l'usage peut entraîner un surcoût de calcul et donc limiter la scalabilité du protocole. Tout au long de ce manuscrit, nous nous concentrons sur trois problèmes impliquant plusieurs utilisateurs.

La première contribution se concentre sur la conception d'un framework de bandits fédérés, où un serveur de fédération agissant en tant qu'agent d'apprentissage, tire successivement un bandit, ce qui produit une récompense provenant d'une distribution inconnue associée au bandit choisi. Dans cette contribution, nous introduisons Tango, un protocole de bandits fédérés sécurisé corrigeant et étendant notre première tentative Samba, qui s'est révélée être non sécurisée. Nous prouvons que Tango empêche le serveur de fédération d'apprendre les récompenses générées ainsi que le bandit choisi, au prix d'un surcoût de calcul important dû à l'utilisation de primitives cryptographiques coûteuses. Dans la deuxième partie de cette contribution, nous introduisons Salsa, un protocole de bandits fédérés sécurisé s'éloignant de l'esprit de Samba et Tango, empêchant le serveur de fédération d'apprendre des données sensibles tout en obtenant des performances élevées.

La deuxième contribution de ce manuscrit traite de la conception d'un système de billetterie, impliquant un grand nombre d'utilisateurs. Malgré la forte demande, ces systèmes offrent des garanties très restreintes. Par exemple, il est facile de revendre un billet deux fois. Pire, la majorité des billets sont nominatifs, permettant d'identifier le propriétaire d'un billet. En utilisant des primitives cryptographiques standards, nous proposons Applause et Spotlight, deux systèmes de billets garantissant l'anonymat des utilisateurs tout en proposant l'achat, le remboursement, la validation et le transfert d'un billet. La différence entre Applause et Spotlight réside dans la capacité de lever l'anonymat d'un utilisateur : dans Applause, l'anonymat d'un utilisateur est garanti. Ceci est toujours valable dans Spotlight sauf pour une partie externe capable de lever l'anonymat d'un utilisateur. Toutefois, lever l'anonymat dans Spotlight est possible au prix d'une validation de billet plus longue.

Notre troisième contribution se concentre sur le problème de diffusion d'un fichier à un groupe d'utilisateurs. La solution triviale consistant à stocker des fichiers sur un seul serveur disponible publiquement est insuffisante, notamment lorsque le serveur est hors service. Dans cette contribution, nous introduisons un protocole efficace et universellement composable permettant à un utilisateur de partager un fichier avec un groupe d'utilisateurs, tout en garantissant la confidentialité et l'intégrité du fichier ainsi que l'authentification de l'expéditeur.

Remerciements

Le point culminant de la thèse est sans aucun doute la soutenance, un moment où se rassemblent proches et experts, marquant l'un des jours les plus importants de la vie d'un doctorant. Ne faisant pas exception, j'ai eu l'honneur de recevoir un jury exceptionnel, que je remercie chaleureusement pour leur présence. Ce jury était composé de Ioana Boureanu, Chloé Hébant, Melek Önen, Benjamin Nguyen et présidé par Jean-Guillaume Dumas. Mes remerciements les plus sincères vont à Melek Önen et Benjamin Nguyen pour avoir accepté d'être les rapporteurs de ma thèse.

La thèse peut aisément se comparer à un long fleuve agité. Durant ce périple, j'ai eu la chance d'être très bien accompagné, notamment par mon directeur de thèse, Pascal Lafourcade, sans qui rien de tout cela n'aurait été possible. Sa bonne humeur, sa bienveillance, son énergie débordante ainsi que son excellence auront été un guide précieux. Pour toutes ces raisons, je lui adresse mes remerciements les plus chaleureux.

Cette thèse n'aurait pas été possible sans le soutien du projet Datalake For Nuclear financé par la Banque Publique d'Investissement, que je remercie sincèrement.

Au-delà de l'aspect scientifique, ces trois années m'ont permis de rencontrer des personnes formidables, capables de transformer une journée de travail ordinaire en souvenir précieux. Je pense notamment à mes collègues de bureau, toujours présents pour partager de bons moments, en F203 et ailleurs. Anthony, Charles, Dhekra, Frédéric, Léo, Lola-Baie, merci à vous. Un merci particulier à Valérie pour avoir facilité notre quotidien et pour égayer nos journées avec son bel accent du sud. Je tiens à exprimer une reconnaissance spéciale à Léo, qui a joué officieusement un rôle de mentor et d'encadrant tout au long de cette thèse.

La sphère personnelle a sans doute joué le rôle le plus discret, mais aussi le plus crucial et nécessaire de tout mon parcours. Je ne serai jamais assez reconnaissant envers mes proches pour leur soutien indéfectible. J'adresse un sincère remerciement à mon oncle et à ma tante pour tout ce qu'ils m'ont apporté. Depuis ma tendre enfance, la "première femme de ma vie", qui se reconnaîtra, s'est battue sans relâche malgré les nombreuses difficultés pour me permettre de grandir dans un environnement sain et d'accomplir mes rêves, sans rien attendre en retour. Tu ne peux qu'être fière de tout ce que tu as accompli.

Sous prétexte d'une thèse à Clermont-Ferrand, le destin m'a fait rencontrer ce qui est aujourd'hui mon rayon de soleil. Elisa, il est difficile de quantifier ton impact positif sur mon moral tant il est important. Mais une chose est sûre : tout aurait été bien plus difficile sans toi. Ta place dans ma vie est précieuse et rassurante, au point où tu m'as fait repenser l'avenir.

Contents

Chapter 1

Introduction

Just as a reader needs to comprehend English to understand this manuscript, computers require a clearly defined set of interactions to understand requests coming from other computers, and act. A *protocol* corresponds to this definition: Defining interactions and their content between two or more computers, in order to offer a functionality. As an example, suppose an employee having an important document to share with a group of coworkers. A protocol corresponds to the interactions between the employee, potentially one or more servers, and the coworkers. Ultimately, at the end of the rotocol execution, the group of coworkers have access to the document.

A functionality ensured by the protocol comes with a set of constraints. Back to our previously mentioned example, the document has to be shared only with a group of coworkers. Hence, coworkers outside of the specified group should not access the document. These constraints constitute the *security properties* that the protocol aims to respect. This manuscript is dedicated to the design of protocols respecting security properties. Before going any further, it is important to understand that absolute security, in the sense that no practical attack exists against a protocol, is *hard* to obtain. Even protocols that are widely used around the world might be attacked, as demonstrated again, recently, with the Secure Shell protocol, leading to more than ten millions of vulnerable servers [FB23]. Back in the past, a protocol was considered secure (*i.e.,* respecting the security properties) if no attack was discovered. This proof methodology falls short when an attack exists but is kept secret and used as a backdoor. The most striking historical example was Enigma, broken by British cryptographers in 1942, subsequently enabling all Enigma-encrypted communications to be decrypted.

In the wake of successive failures, cryptography has gradually moved closer to the field of mathematics. This new approach requires the protocol designer to formally prove that his protocol is correct and safe *i.e.,* ensuring all the desired security properties. This paradigm is better known as *provable security*. The crucial point is that the security of a protocol is only valid under certain assumptions. Indeed, the security of a protocol is studied under the prism of a so-called *security model*, delimiting the capabilities of an adversary trying to attack the protocol. Its capabilities include for instance the corrupted parties involved in the protocol and thus its access to some knowledge and its possible actions on the protocol. Hence, one of the most important tasks for the protocol designer, as done in this manuscript, is to carefully model the desired security, otherwise limiting the interest of the protocol.

The security brought by the usage of cryptographic primitives in a protocol is achieved at the cost of additional computation time due to cryptographic operations and, most of the time, to a greater amount of information transmitted between the parties involved in the protocol. In addition to the security considerations, a protocol

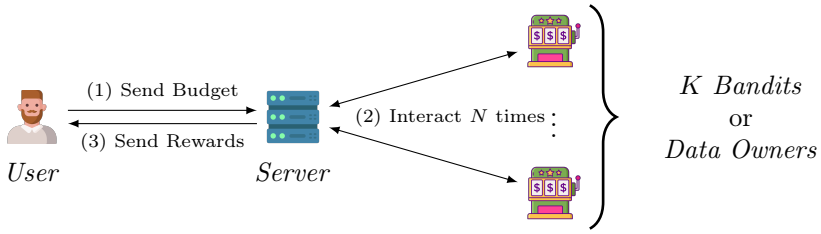


Figure 1.1: Federated Multi-Armed Bandits Architecture.

designer has to study the context in which the protocol is used, including for instance the number of users. The holy grail for a protocol designer is to be able to build a protocol that meets security requirements, while being effective even for large scale to reach scalability. In this manuscript, we particularly focus on the *multi-users* setting in which several users are involved in the running protocol. The considered number of users may drastically impact the performance of the protocol: A scalable protocol naturally supports a large number of users and hence is more suitable for large use-cases. This manuscript covers three subjects operating at different scales. In our first subject, at most 100 users are interacting with a federation server, in order to solve a reward maximisation problem. Our second and third case of study operates with potentially thousands of users, focusing respectively on the design of secure anonymous ticketing system and secure efficient distributed file transfer. Whereas the involvement of thousands of users is clear in the ticketing system context, the distributed file transfer needs more explanations: Our file transfer protocol addresses the case of a distributed company, in which a file is sent from a sender to a potentially large group of employees. We explain in details the problems we consider in this manuscript.

Secure Federated Multi-Armed Bandits. A bandit is a casino machine equipped of an arm. Once a coin has been inserted in the bandit and the arm pulled, the bandit produces a reward with a fixed probability ε or not reward with probability $1-\varepsilon$. In the sequel, a bandit is called a *Data Owner* ($\mathcal{DO}$) without distinction, since it owns the reward probability ε. The *reward maximisation problem* asks to a stateful user, having a finite budget of N coins, and facing K bandits, to maximise its reward. This problem, intensively used in recommender systems [LCLS10] to game playing [KS06], belongs to the line of *reinforcement learning* research, in which the user has to choose the best among a finite set of actions, in our case by pulling the arm of its choice adaptively, following a stochastic approach.

Numerous algorithms have been introduced in the literature, allowing a user to maximise its rewards. In this work, we focus on the *federated* multi-armed bandits, depicted in Figure 1.1: The user initially sends the budget N to a new party referred as the *server*, in charge to execute a rewards maximisation algorithm. For each coin of the budget, the server pulls a bandit of its choice following the reward maximisation algorithm strategy. At the end, the server sends back the obtained rewards to the user.

In practice, the reward probability ε_i is unknown by the data owner $\mathcal{DO}_i$. To estimate it, $\mathcal{DO}_i$ has at its disposal the number of obtained rewards s_i and the number of pulls

n_i, both with respect to data owner $\mathcal{DO}_i$. The estimation, computed following the reward maximisation strategy, is called a *score*. The estimation of ε_i, computed by $\mathcal{DO}_i$ is referred as the *score* and denoted v_i. During an interaction between the server and data owners, each score v_i is sent to the server, and has to chose the best machine, still following the reward maximisation strategy. The response from the server to the data owners consists of K bits $b_1, \ldots, b_K$, where exactly one bit b_i equals one, meaning that the i-th data owner has been chosen by the server. The constraints, or the security properties, that we aim to achieve are the following: The server should not learn any score v_i, as well as any obtained rewards r_i. In addition, the *chosen* data owner, should not be identified by the server.

Few works related to this problem have been proposed in the literature [TD16, CLLS19, CLLS20, SS21]. Almost all of these works are not specific to the targeted federated learning architecture. In the order to fill the gap, we introduced in 2022 a federated learning framework called Samba [CLMS22], designed to solve the reward maximisation problem in the federated learning architecture, using simple yet efficient cryptographic primitives including secret-key encryption and masking. However, after investigation, it appears that Samba suffers from correctness and security issues. In addition, no security definition has been provided in the literature yet addressing the reward maximisation problem in the federated learning setting, later referred as the secure federated multi-armed bandits problem.

The presented contribution starts by a formal security definition of a secure federated multi-armed bandits protocol, based on the real-ideal paradigm. In a nutshell, our security definition models that a party involved in the protocol should not learn any information on the score v_i provided by a data owner, on any selection bit b_i but also on the cumulative rewards s_i, and by extension on the total cumulative rewards s sent to the user. Equipped with our security definition, we present two secure federated multi-armed bandits protocols Tango and Salsa achieving our security definition. Our first protocol extends and fixes the flawed Samba protocol using homomorphic encryption, keeping in mind to respect the initial blueprint of Samba. This protocol relies on two federation servers, and is constructed using secret-key and public-key encryptions, as well as additive and fully homomorphic encryption. Compared to Samba, Tango ensures correctness but also genericity, in the sense that any standard multi-armed bandits algorithm, traditionally executed by a single party, can be plugged into Tango and hence enjoying the security offered by Tango, without impacting the total cumulative rewards returned to the user. In addition, Tango is said resistant against data owner failures, allowing to return the same total cumulative rewards to the user, even in case where a data owner is offline until the end of the protocol.

Due to the homomorphic computations, we have shown that Tango suffers from a high computation overhead. To overcome this issue, we have proposed a second protocol Salsa, respecting all the aforementioned properties including genericity, correctness, security and resistance against data owner failure, while being particularly efficient. For the sake of concreteness, whereas Tango requires more than 2.5 seconds to select the best among 9 arms, Salsa requires 0.15 millisecond. The efficiency of Salsa is achieved thanks to secure two-party computations, in which computations are performed over a so-called share. Briefly, given a data v, a data owner derives two shares $\langle v \rangle^0$ and $\langle v \rangle^1$

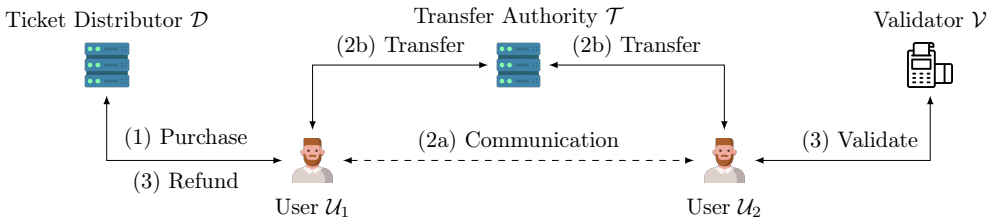


Figure 1.2: Ticketing System Architecture.

that are later distributed to two distinct and non-colluding servers. Given only one share, it is infeasible to recover the data v, ensuring the privacy of v. Despite that the data v has been shared, it remains possible for the two servers to obtain the shares $\langle f(v)\rangle^0$ and $\langle f(v)\rangle^1$ where $f(v)$ corresponds to an arbitrary circuit f evaluated on the data v. In Salsa, we take advantage of this ability of running arbitrary computations while being privacy-preserving to solve the reward maximisation problem efficiently.

Anonymous Auditable Transferable E-Ticket. Ticketing is a commonly used system where a user obtains a ticket for a show. During the Covid period, many shows have been either cancelled, or more commonly, postponed to a later date. Due to the date change, many users were unable to attend the event, for various reasons. Users had two options for obtaining a refund: Either to contact the agency from which the ticket was purchased, or to resell the ticket to another user, usually via a third-party platform. Unfortunately, most of these third-party platforms are not ticket-specific and do not check the validity of tickets. As a result, a number of malicious users have managed to resell the same ticket several times. From the point of view of the user who is the victim of such a scam, the consequences can be considerable when you take into account the price of the ticket, transport and accommodation, but also the psychological aspect, a concert being awaited sometimes for several months.

In this manuscript, our aim is to design a ticketing protocol secure against the aforementioned attack. As any traditional ticketing system, our construction proposes purchase, refund of a ticket, but also transfer of a ticket between two users. An overview of the protocol is depicted in Figure 1.2: First, a user is allowed to purchase a ticket at a dedicated entity called Ticket Distributor and denoted $\mathcal{D}$. If the user cannot attend the event anymore, the user is allowed to proceed to a refund, again with the Ticket Distributor $\mathcal{D}$. The keystone of the protocol is the ability, for the user, to transfer its own ticket to another user. Using our protocol, a user having received (and kept) a ticket via the transfer procedure is guaranteed to be accepted at the event. This is solved by preventing the same ticket to be transferred twice, thanks to the Ticket Transfer Authority $\mathcal{T}$, whose the role is to guarantee the fairness of the ticket transfer. Finally, when the user attends the event, the protocol allows a user to validate its ticket at a validation point, physically located at the entrance of the event, called a validator and denoted $\mathcal{V}$.

In addition to the traditional functionalities, in contrast to all other ticketing systems, we have focused our attention on a ticketing system ensuring *anonymity of users*. This anonymity feature is crucial to ensure a complete privacy of users, and should be

preserved against the system (composed of the Ticket Distributor $\mathcal{D}$, the Ticket Transfer Authority $\mathcal{T}$ and Validators $\mathcal{V}$), but also against other users, particularly during the ticket transfer. Our first ticketing system called Applause includes all traditional functionalities. It is secure against double-spending of the same ticket, but also provides anonymity for users during every interaction with the system. We stress that anonymity in Applause is strong, in the sense that the identity of a user is never involved during an interaction. But more importantly, each individual interaction (*e.g.,* purchase of a ticket) cannot be linked with another interaction as coming from the same user.

In addition to the design of Applause, we have focused our attention on some edge cases such as fire or terrorism in which an external authority would be allowed to identify users attending the event for civil security purposes. This interesting feature, however, is facing anonymity of users preventing a user to be identified by the system. Observe that in such setting, anonymity of users still holds against all entities, except for an additional *explicitly identified* authority allowing to reveal the identity of users. In this context, we propose Spotlight combining Applause and this identity revealing feature. In this protocol, a user is still able to purchase, refund and transfer a ticket while preserving its anonymity against other users and the system. However, this additional authority called the Judge, denoted $\mathcal{J}$, is able to reveal the identity of the user.

Our motivation for creating two distinct protocols lies on two important observations: First, the ability to reveal identity of users introduces a risk of excessive of misuses. For instance, in our model in which Spotlight is considered secure, the identity revealing key must be kept secret by the judge. Otherwise, anonymity is trivially broken for any user. In contrast, this judge-leakage scenario never happens with Applause. Second, the auditability feature comes at the price of an additional trusted third-party as well as a slightly slower ticket validation, requiring 165 milliseconds instead of 45 milliseconds, which is still practical. Let elaborate more on the practicality of these two protocols. Thanks to our proof of concept, written in Rust, we have shown that Applause and Spotlight are efficient. A complete interaction including a ticket purchase, transfer and validation requires 100 milliseconds for Applause at most 250 milliseconds for Spotlight, to perform all the cryptographic operations (communication times excluded). Therefore, Applause and Spotlight are guaranteed to be suitable for events dealing with a large number of attendees.

From a security standpoint, we have proved that Applause and Spotlight ensure ticket unforgeability, ticket privacy, resistance against double-spending of the same ticket anonymity of users using the game-based computational model.

Secure Distributed File Transfer Protocol. Sending a file to a server through internet is a very simple and common action. A typical file transfer architecture is composed of several clients and a central server, handling file download and upload. This architecture, however, suffers from drawbacks: Suppose a world-scale company having several offices. A user localised far away from the server, attempting to download a given file, may suffer from a longer communication delay, inherent of the network size, used to contact the server. Second, a server is, by definition, limited by its computational resources and hence is able to handle a finite number of requests at the same time. Last but not least, in case where the server is down, the file cannot be downloaded anymore.

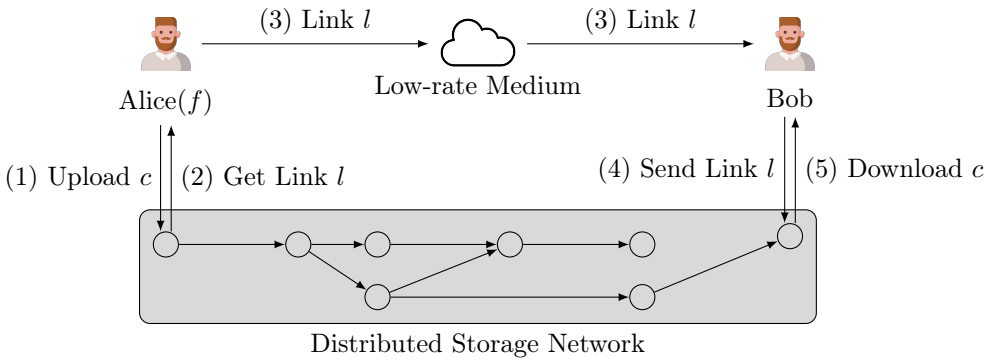


Figure 1.3: Representation of our system where a sender shares a file f to a receiver.

A *Distributed file storage system*, solving all of these issues, acts similarly to a graph, in which nodes are servers and edges are connection between servers. When a file is uploaded on a server, the file is propagated through the network. The direct advantage of a distributed file storage system as presented below, is a fault-tolerance system, since the crash of a server does not impact files accessibility. More importantly, since a file is propagated, it can be accessed using a closer server, reducing communication delay, leading to a more efficient file access.

In this manuscript, we focus our attention on the specific case of a world-scaled company, and study a design of a file transfer protocol ensuring file accessibility using a distributed file storage system. The security properties of a company-focused file transfer system are identical to secure mail exchange: First, the file transfer system may be in charge of transferring sensitive files, accessible only by precisely defined users or groups, defined as the confidentiality of the file. Second, the integrity of the file property prevents any modification during the file transfer, a useful property, for instance, when a juridic-valued contract is exchanged. Third and last, authentication of the sender prevents, for example identity theft.

In this work, we have constructed a *simple but efficient* file transfer protocol addressing all the aforementioned properties. The general overview of our protocol, described in Figure 1.3, works as follow: Suppose Alice having a file f, which she wants to send to Bob. First, Alice stores the encryption of f, denoted c, at the closest storage server in a distributed file system. As a response, Alice obtains a link l, associated to the file. Next, we assume a low-rate communication channel between all users of the system, used by Alice to transfer her link l to Bob. We stress that this communication channel is *not* intended to send a file, but *short* messages having a size being independent of the (potentially large) file f. Later, Bob contacts the closest storage server of the distributed file system with the link l, and obtains as a response c. Finally, Bob recovers f from c using a dedicated secret decryption key (previously obtained from an authority).

In contrast with the two first works, in which the security has been proven secure under the game-based variant of the computational model, the security of this protocol has been proven under the *Universal Composability* (UC) model, introduced for the first time by Canetti [Can20]. These two approaches, while sharing some similarities, lead to a radically different protocol modelisation and formalisation. Putting aside technical

details, UC models all security properties as a *single* modelisation, referred in UC as the *ideal functionality*, shown later to be indistinguishable to the *real protocol*. The major feature of UC is the composability feature allowing ideal functionalities to be composed together in order to construct protocols, realising more complex ideal functionalities. In contrast to existing works providing no implementation, based on a proof-of-concept in Rust, we claim that our solution is efficient, thanks to the simplicity of our approach.

Manuscript Organisation. The manuscript is organised following the same order of the work presented above: In Chapter 3, we present our work introducing the two secure federated multi-armed bandits protocols Tango and Salsa. These two protocols are still under submission, and must be seen as the following works of our initial contribution of Samba published at the *Journal of Artificial Intelligence Research* in [CLMS22]. In Chapter 4, we present our two secure ticketing systems Applause and Spotlight, being part of our contribution accepted to the *19th ACM ASIA Conference on Computer and Communications Security* (AsiaCCS 2024). Finally, in Chapter 5, we introduce our work accepted to the *15th International Conference on Cryptology in Africa* (AFRICACRYPT 2024), focusing the design of iUC-secure file transfer system based on standard attribute-based encryption and distributed file storage system. To make the manuscript more accessible, we introduce in Chapter 2 the formalism required to grab the introduced works.

Published Works. Below are presented the title and the abstract of the five works published during this thesis, including the works of Samba [CLMS22] being the starting point of the work introduced in Chapter 3, as well the work of Applause and Spotlight published at the AsiaCCS conference.

- *Ciucanu, R., Lafourcade, P., Marcadet, G., and Soare, M. (2022). SAMBA: A Generic Framework for Secure Federated Multi-Armed Bandits. Journal of Artificial Intelligence Research, 73.*

  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 maximisation 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 primi-

tives is linear in the size of the input, which is confirmed by our proof-of-concept implementation.

- *Lafourcade, P., Marcadet, G., Olivier-Anclin, O., and Mahmoud, D. (2024). Auditable, Anonymous, Transferable Ticketing System. The 19th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2024).*

  Abstract. Digital ticketing systems typically offer ticket purchase, refund, validation, and, optionally, anonymity of users. However, it would be interesting for users to transfer their tickets, as is currently done with physical tickets. We propose Applause, a ticketing system allowing the purchase, refund, validation, and transfer of tickets based on trusted authority, while guaranteeing the anonymity of users, as long as the used payment method provides anonymity. To study its security, we formalise the security of the transferable E-Ticket scheme in the game-based paradigm. We prove the security of Applause computationally in the standard model and symbolically using the protocol verifier ProVerif. Applause relies on standard cryptographic primitives, rendering our construction efficient and scalable, as shown by a proof-of-concept. In order to obtain Spotlight, an auditable version of the protocol that we also proved to be secure, users will remain anonymous except for a new third party, which will be able to disclose their identity in the event of a disaster.

- *Marcadet, G., Ciucanu, R., Lafourcade, P., Soare, M., and Amer-Yahia, S. (2022) Samba: A System for Secure Federated Multi-Armed Bandits. 38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022, 3154–3157.*

  Abstract. The federated learning paradigm allows several data owners to contribute to a machine learning task without exposing their potentially sensitive data. We focus on cumulative reward maximisation in Multi-Armed Bandits (MAB), a classical reinforcement learning model for decision making under uncertainty. We demonstrate Samba, a generic framework for Secure federAted Multi-armed BAndits. The demonstration platform is a Web interface that simulates the distributed components of Samba, and which helps the data scientist to configure the end-to-end workflow of deploying a federated MAB algorithm. The user-friendly interface of Samba, allows the users to examine the interaction between three key dimensions of federated MAB: Cumulative reward, computation time, and security guarantees. We demonstrate Samba with two real-world datasets: Google Local Reviews and Steam Video Game.

- *Lafourcade, P., Marcadet, G. and Robert, L. (2023). RMC-PVC: A Multi-Client Reusable Verifiable Computation Protocol. In J. Hong, M. Lanperne, J. W. Park, T. Cerný, and H. Shahriar (Eds.), Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023, Tallinn, Estonia, March 27-31, 2023 (pp. 1558–1565). ACM.*

Abstract. The verification of computations performed by an untrusted server is a cornerstone for delegated computations, especially in multi-clients setting where inputs are provided by different parties. Assuming a common secret between clients, a garbled circuit offers the attractive property to ensure the correctness of a result computed by the untrusted server while keeping the input and the function private. Yet, this verification can be guaranteed only once.

Based on the notion of multi-key homomorphic encryption (MKHE), we propose RMC-PVC a multi-client verifiable computation protocol, able to verify the correctness of computations performed by an untrusted server for inputs (encoded for a garbled circuit) provided by multiple clients. Thanks to MKHE, the garbled circuit is reusable an arbitrary number of times. In addition, each client can verify the computation by its own. Compared to a single-key FHE scheme, the MKHE usage in RMC-PVC allows to reduce the workload of the server and thus the response delay for the client. It also enforces the privacy of inputs, which are provided by different clients.

- *Dailly, A., Lafourcade, P., and Marcadet, G. (2024). How did they design this game? Swish: complexity and unplayable positions, 12th International Conference on Fun with Algorithms, FUN 2024, June 4 to June 8, 2024, Maddalena Island, Sardinia, Italy.*

  Abstract. Swish is a competitive pattern recognition card-based game, in which players are trying to find a valid cards superposition from a set of cards, called a "swish". By the nature of the game, one may expect to easily recover the logic of the Swish's designers. However, even with a reverse engineering of Swish, no justification appears to explain the number of cards, of duplicates, but also under which circumstances no player can find a swish. In this work, we formally investigate Swish. In the commercial version of the game, we observe that there exist large sets of cards with no swish, and find a construction to generate large sets of cards without swish. More importantly, in the general case with larger cards, we prove that Swish is NP-complete.

- *Lafourcade, P., Marcadet, G. and Robert, L. (2024). iUC-Secure Distributed File Transfer From Standard Attribute-based Encryption. 15th International Conference on Cryptology in Africa, AFRICACRYPT 2024, July 10-12, 2024, Douala, Cameroon.*

  Abstract. Attribute-Based Encryption (ABE) stands as a cryptographic cornerstone, enabling access control to messages based on user attributes. The security definition of standard ABE is shown to be impossible in Universal Composability (UC) against an *active* adversary. To overcome this issue, existing formal UC security definitions of ABE rely on additional properties for ABE, necessary to prove security against an active adversary, excluding standard ABE by definition. In light of the composability feature offered by UC and the absence of ideal functionality tailored for standard ABE, we propose the two following contributions: (1) We

construct the first ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ for ABE which, under reasonable hypothesis against static corruption, can be realised using an IND-CCA2-secure ABE scheme; and (2) our $\mathcal{F}_{\mathsf{ABE}}$ leads us to propose a protocol solving a simple yet highly practical, world-scaled company-focused problem: Efficient file transfer. The proposed construction provides data integrity, sender authentication, attribute-based file access, featured with *constant* data size transferred between users. This is achieved by relying on two efficient building blocks: ABE and signature, which are layered atop of the hash-based distributed storage system IPFS. Our protocol, strengthened by a formal security definition and analysis under the Universally Composable (UC) framework called iUC, is proved to realise our problem-oriented authenticated attribute-based file transfer ideal functionality. Finally, we implement our proposal with a proof-of-concept written in Rust, and show it is practical and efficient.

Chapter 2

Background

Contents

Throughout this manuscript, we focus on cryptographic protocols built using cryptographic primitives. For the sake of clarity, we present in this chapter our notation as well as the essential primitives and hypothesis. we also introduce tools used in our proofs of security.

2.1 Notations

We introduce the most common notations used in this manuscript. More specific notations are introduced when required.

- By $a \leftarrow A()$, we denote the affectation of the output of the algorithm A to the variable a.
- By $a \leftarrow\!\!\$\ \mathcal{S}$, we denote the random sampling from the set $\mathcal{S}$, affected to the variable a.
- By λ, we denote the security parameter.
- 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 $[1, \ldots, x]$.
- An algorithm is written as $\mathsf{Alg}(\cdot)$, whereas a protocol between parties $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is written as $\mathsf{Proto}\langle \mathcal{A}_1(\cdot), \ldots, \mathcal{A}_n(\cdot)\rangle \rightarrow \mathcal{A}_1(\cdot), \ldots, \mathcal{A}_n(\cdot)$. Parties which do not receive an output from the protocol are sometimes omitted.
- By $[x_i]_{i \in [\![n]\!]}$ we denote the list $[x_1, \ldots, x_n]$.
- By $\mathbb{Z}$, we denote the set of integers $\{\ldots, -1, 0, 1, \ldots\}$. By $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$, we denote the set $\{0, \ldots, n-1\}$ and $\mathbb{Z}_n^*$ corresponds to the set of invertible integers in $\mathbb{Z}_n$ *i.e.,* integers $a \in \mathbb{Z}_n$ for which there exist an integer $b \in \mathbb{Z}_n$ where $a \cdot b = 1$.

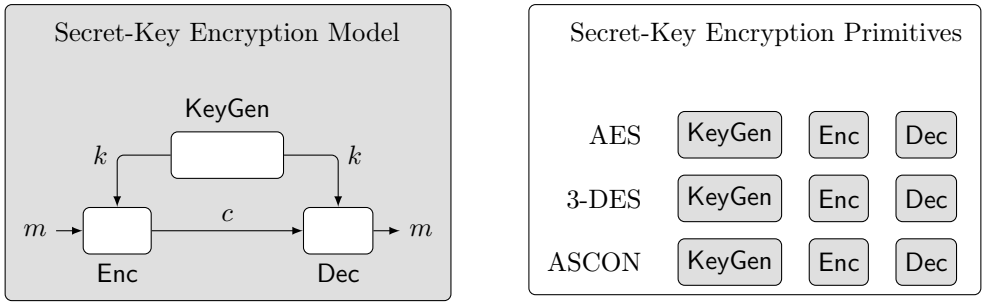


Figure 2.1: Encryption Model and Encryption Primitives.

- By $X \triangleq Y$, we express the perfect indistinguishability between the two distributions X and Y. Similarly, the notation $X \overset{s}{\approx} Y$ expresses the statistical indistinguishability between the two distributions X and Y.

2.2 Black-Box Model

A protocol designer constructing protocols can be viewed as an architect, drawing up a plan of a house. The architect first needs a set of available materials, which when assembled will produce the desired result. The toolbox of a cryptographic protocol designer is composed of essential constructions, called *cryptographic primitives*, later combined to construct protocols. For the sake of clarity, we will first focus on *secret-key encryption*, allowing to respectively encrypt and decrypt a message, using the *same key* (justifying the "secret-key" term) to preserve the confidentiality of the message.

Unifying Cryptographic Primitives. Numerous encryption primitives have been proposed in the literature. We will focus on three well-studied secret-key encryption primitives: Triple-DES (or 3-DES for short) [BM17], Advanced Encryption Standard (AES) [DBN+01] and ASCON [STMC+23]. Each of these primitives have a different internal behaviour: Triple-DES generates three distinct DES keys of 56 bits each, and then encrypts a message m three times sequentially using DES encryption algorithm, producing a ciphertext c. Decryption is the exact inverse operation. AES, in contrast, generates a single key of 128 or 256 bits depending on the desired security level, and encrypts a message m by applying sequentially four distinct and invertible operations during multiple rounds, producing a resulting ciphertext c. Finally, ASCON generates a seed s as a key, used to generate a pseudo-random bitstring r (a bitstring that seems completely chosen at random), used to mask the message m, producing a ciphertext c defined as $m + r$. Decryption requires to recompute the pseudo-random bitstring r based on the seed s, and then to remove the random from the ciphertext with $c - r$, returning m. Even with a strictly minimal level of details, the difference between these three secret-key encryption algorithms is striking. In addition, due to the distinct internal behaviour, the performance of these three primitives are not the same [AEAKH10]. Suppose a protocol designer constructing an arbitrary protocol using AES as the symmetric encryption algorithm. Will the protocol become insecure if we replace AES with ASCON ? Hopefully, replacing a secret-key encryption with another does not affect the

security. Therefore, a protocol designer does not select a particular secret-key encryption; instead, he considers a generic *secret-key encryption model*, viewed as a "black-box", leading to a more general protocol. In Figure 2.1, we have depicted a visual representation of the three mentioned secret-key encryption primitives, fitting the secret-key encryption model defined as three algorithms: The key generation algorithm KeyGen, the encryption algorithm Enc, and the decryption algorithm Dec. A direct advantage happens when a new effective attack is found on a primitive. Indeed, replacing a broken primitive with a secure one does not affect the overall behaviour of the protocol. In the following part of the manuscript, every cryptographic primitive is viewed as black-box, hiding the internal behaviour. This approach to construct protocol is referred as the *black-box model*. In the following of this manuscript, the "primitive" term directly refers to its black-box representation, unless specified.

2.3 Cryptographic Primitives

In this manuscript, the security of the presented protocols relies on cryptographic primitives, whose definition are introduced.

2.3.1 Secret-Key and Public-Key Encryption

Encryption allows one to send a message over an untrusted communication channel, while preserving the confidentiality of the sent messages. In the literature, we distinguish two paradigms for encryption. The first paradigm corresponds to *secret-key encryption*, denoted SKE throughout this work, in which encrypting and decrypting messages requires the same (secret) key. These kinds of encryption are very efficient [AEAKH10, Pan16] and hence are designed to encrypt a large size of data.

Definition 1 (Secret-Key Encryption, SKE)**.** A secret-key encryption scheme is defined by the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of polynomial-time algorithms where:

- $\underline{\mathsf{KeyGen}(1^\lambda) \to k}$: Given the unary representation of the security parameter λ, outputs the secret key k.
- $\underline{\mathsf{Enc}(k, m) \to c}$: Given the secret key k and a plaintext m, outputs a ciphertext c.
- $\underline{\mathsf{Dec}(k, c) \to m}$: Given the secret key k and a ciphertext c, outputs a plaintext m.

Communicating over an insecure channel using secret-key encryption requires beforehand to agree on the used secret key. Transmitting a message securely to another user without having agreed on a secret key is done by considering a public encryption key, say pk, accessible by anyone, while having a private decryption key, say sk, used by the message recipient to decrypt every sent message. This is exactly the aim of the second paradigm, called *public-key encryption*, denoted PKE.

Definition 2 (Public-Key Encryption, PKE)**.** A public-key encryption scheme is defined by the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of polynomial-times algorithms where:

- $\underline{\mathsf{KeyGen}(1^\lambda) \to (sk, pk)}$: Given the unary representation of the security parameter λ, outputs the public key pk and the secret key sk.
- $\underline{\mathsf{Enc}(pk, m) \to c}$: Given the public key pk and a plaintext m, outputs a ciphertext c encrypting the plaintext m under the public key pk.

- $\underline{\mathsf{Dec}(sk, c) \rightarrow m}$: Given the private key sk and a ciphertext c encrypting the plaintext m, outputs m.

Correctness of Encryption. For both secret-key and public-key encryption, the decryption procedure applied on a ciphertext encrypting a message m should result with the message m. This property is called *decryption correctness*, and is formally defined as follows: For every security parameter λ, any secret-key $k \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and any key pair $(sk, pk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, any message m, correct decryption is ensured if we have:

$$m = \mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, m)) = \mathsf{PKE.Dec}(sk, \mathsf{PKE.Dec}(pk, m))$$

This definition considers the case of perfect decryption in which the decryption procedure always succeeds. A more general definition called correct decryption with probability ϵ, modelling a decryption procedure that succeeds with probability $1 - \epsilon$.

Definition 3 (ϵ-decryption correctness of SKE and PKE)**.** For every security parameter λ, any secret key $k \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$ and any key pair $(sk, pk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, any message m, we say that SKE and PKE ensure ϵ-decryption correctness if we have:

$$\Pr\left[m = \mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, m))\right] = \Pr\left[m = \mathsf{PKE.Dec}(sk, \mathsf{PKE.Dec}(pk, m))\right] = 1 - \epsilon$$

The Security Definition Formalism. As we will see in a moment, the security of a cryptographic primitive are most of the time expressed following a *game-based* formalism. A game takes the form of an algorithm executed by a polynomial-time algorithm denoted $\mathcal{C}$ called the *challenger*, playing against a polynomial-time algorithm denoted $\mathcal{A}$ and called the *adversary*. A standard structure of a game is described as follows: The challenger performs some computations, potentially provides some inputs to the adversary and, ultimately awaits an output from the adversary. Eventually, the adversary responds to the challenger. If the response fulfils the *winning conditions*, then we say that the adversary *wins* the game. If necessary, the challenger allows the adversary to call polynomial-time algorithms called *oracles* executed by the challenger. By $y \leftarrow \mathcal{A}^{\mathsf{OFunc}(a,\cdot)}(x)$, we denote the challenger inputting x to the adversary and awaiting for y. When the adversary $\mathcal{A}$ performs computation, it is allowed to call the oracle OFunc, taking two arguments where the first one is set to a by the challenger, for every execution of the oracle. For clarity, we assume that any variable provided by the challenger can be modified by the oracle, useful for example to add or remove elements from some set. In the provided security definitions, we will intensively use the notion of "negligible advantage" for an adversary to win a given game. This notion highlights that for any (polynomial-time) adversary $\mathcal{A}$, it has intuitively of very low probability to win a game. Formally, a function $f : \mathbb{N} \rightarrow \mathbb{R}$ is said negligible if for every polynomial p, there exist an integer n such that for every $x \geq n$:

$$|f(x)| \leq \frac{1}{p(x)}$$

$\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda)$

1 : $k \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$
2 : $m_0, m_1 \leftarrow \mathcal{A}^{\mathsf{Enc}(k,\cdot)}(\lambda)$
3 : $b \leftarrow\!\!\$\ \{0,1\}$
4 : $c \leftarrow \Pi.\mathsf{Enc}(k, m_b)$
5 : $b' \leftarrow \mathcal{A}(c)$
6 : **return** $b = b'$

$\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(\lambda)$

1 : $(sk, pk) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$
2 : $m_0, m_1 \leftarrow \mathcal{A}(\lambda, pk)$
3 : $b \leftarrow\!\!\$\ \{0,1\}$
4 : $c \leftarrow \Pi.\mathsf{Enc}(pk, m_b)$
5 : $b' \leftarrow \mathcal{A}(c)$
6 : **return** $b = b'$

$\mathsf{Exp}^{\text{IND-CCA2}}_{\mathcal{A},\Pi}(\lambda)$

1 : $(sk, pk) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$
2 : $m_0, m_1 \leftarrow \mathcal{A}^{\mathsf{ODec}(sk,\perp,\cdot)}(\lambda, pk)$
3 : $b \leftarrow\!\!\$\ \{0,1\}$
4 : $c \leftarrow \Pi.\mathsf{Enc}(pk, m_b)$
5 : $b' \leftarrow \mathcal{A}^{\mathsf{ODec}(sk,c,\cdot)}(c)$
6 : **return** $b = b'$

Oracle $\mathsf{ODec}(sk, c, c')$

1 : **if** $c = c'$: **return** $\perp$
2 : $m \leftarrow \Pi.\mathsf{Dec}(sk, c')$
3 : **return** m

Figure 2.2: Security experiments for the IND-CPA property using secret-key encryption (experiment at the top left corner) and public-key encryption (experiment at the top right corner), and for the IND-CCA2 property (experiment at the bottom left corner).

Along this manuscript, we denote by $\mathsf{negl}(\lambda)$ a polynomial negligible in the security parameter λ, defining the desired security level (*e.g.,* 128 or 256 bits of security). We are now ready to explore the different security definitions introduced in this manuscript, starting with the security definition for secret-key and public-key encryption.

Security of Secret-Key and Public-Key Encryption. Both secret-key and public-key encryptions are used to communicate over an insecure communication channel. The expectation from a security standpoint should be the same. Intuitively, an encryption scheme is said to be secure if for any polynomial-time adversary, it preserves the confidentiality of the encrypted message. In other words, it cannot distinguish between the encryption of a message, say m_0, from another message, say m_1, even if m_0 and m_1 are *chosen* by the adversary. More than choosing the challenge messages, we have to assume that the adversary has the ability to obtain the encryption of *any* message of his choice, before and after obtaining the challenge ciphertext. Under the prism of the secret-key encryption, this encryption oracle perfectly models the ability for an attacker to send a request to a running server accepting some message and returning its encryption. The formal definition of this experiment is called *Indistinguishability under Chosen-Plaintext Attack* (IND-CPA) that we have depicted in Figure 2.2.

Definition 4 (Indistinguishability under Chosen-Plaintext Attack, IND-CPA)**.** Let Π be an encryption scheme defined by the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Then, Π is said IND-CPA secure if for every adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\Pi} = \left| \Pr\left[\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{A},\Pi}(1^\lambda) \rightarrow 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Suppose now that the adversary has still access to any encryption of the plaintext of its choice, but now has access to a *decryption oracle*, allowing to decrypt the ciphertext of its choice to recover the underlying message of a given ciphertext (obtained or computed) by the adversary, excluding the challenger ciphertext. This setting is commonly known as the *Indistinguishability under Chosen-Ciphertext Attack* (IND-CCA). We can refine this setting by allowing or not the adversary to decrypt any ciphertext of its choice *after* having obtained the challenge ciphertext. The setting where the adversary does *not* have access the decryption oracle after obtaining the challenge ciphertext is called IND-CCA1, whereas the setting where the adversary has access to the decryption oracle after obtaining the challenger ciphertext is called IND-CCA2. In this work, we are particularly interested by the IND-CCA2 security, whose formal security game is depicted in Figure 2.2 (Note that we present chosen-ciphertext attack experiment only for public-key encryption).

Definition 5 (Indistinguishability under Chosen-Ciphertext Attack, IND-CCA2)**.** Let Π be an encryption scheme defined by the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Then, Π is said IND-CCA2 secure if for every adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}^{\text{IND-CCA2}}_{\mathcal{A},\Pi} = \left| \Pr\left[\mathsf{Exp}^{\text{IND-CCA2}}_{\mathcal{A},\Pi}(1^\lambda) \to 1 \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

2.3.2 Homomorphic Encryption

During the 70s, an intuition arises in the cryptography community, formulated for the first time by Rivest, Adleman and Dertouzos [RLRD78], stating that there exist encryption schemes allowing to perform operations directly on the encrypted data, without relying on the decryption key. Until now, numerous encryption schemes have been developed, commonly called *Homomorphic Encryption*. Over the years, many homomorphic encryption schemes has been developed, improving efficiency [Gao18, BV14, CGGI16] and functionality [CZW17, AJJM20].

As any (single-key) encryption scheme, a *homomorphic* encryption scheme is composed of a key generation algorithm KeyGen, an encryption algorithm Enc and a decryption algorithm Dec. In contrast, homomorphic encryption schemes are augmented with an *evaluation* algorithm denoted Eval, allowing to evaluate a given circuit $\mathcal{C}$ over one or more ciphertexts. The supported class of circuits and the depth of these circuits highly depends on the scheme.

Partially Homomorphic Encryption. Most older schemes supporting homomorphic property typically allow only a *single* operation on the ciphertext an arbitrary number of times *e.g.,* addition or multiplication. These schemes supporting a single homomorphic operation are commonly called partially homomorphic encryption schemes. In this work, we are interested in partially homomorphic encryption schemes having the *additively homomorphic* property *i.e.,* performing only addition over encrypted integers. We formally refer to these schemes as Additively Homomorphic Encryption (AHE), formalised below.

Definition 6 (Additively Homomorphic Encryption, AHE)**.** Let Π be an additively homomorphic encryption scheme defined by $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Dec})$ the tuple of polynomial-time algorithms where:

- $\underline{\mathsf{KeyGen}(1^\lambda) \to (sk, pk)}$: Given the unary representation of the security parameter λ, returns (sk, pk).
- $\underline{\mathsf{Enc}(pk, m) \to c}$: Given the public encryption key encryption pk and a message m, outputs the ciphertext c.
- $\underline{\mathsf{Add}(c_1, \ldots, c_n) \to c}$: Given n ciphertext $c_1, \ldots, c_n$ encrypting respectively messages $m_1, \ldots, m_n$, outputs a ciphertext c encrypting the sum $m_1 + \cdots + m_n$.
- $\underline{\mathsf{Dec}(sk, c) \to m}$: Given the secret decryption key sk and a ciphertext c, outputs the message m.

Correctness of Additively Homomorphic Encryption. In contrast with secret-key and public-key encryption, an additively homomorphic encryption scheme has to ensure correctness for freshly computed ciphertext *i.e.,* ciphertext produced by the Enc algorithm, but also for ciphertext derived during the computation via the Add algorithm. Following the notion from [ABC+15], we distinguish between these two correctness properties by referring to the first one as *decryption correctness*, while we refer to the second one as *evaluation correctness*.

Definition 7 (ϵ-decryption correctness for AHE [ABC+15])**.** An additively homomorphic encryption scheme Π defined as the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Dec})$ ensures correct decryption with probability ϵ if for every security parameter $\lambda \in \mathbb{N}$, every $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$, every message m, we have:

$$\Pr\left[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m\right] = 1 - \epsilon$$

Definition 8 (ϵ-evaluation correctness for AHE [ABC+15])**.** An additively homomorphic encryption scheme Π defined as the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Dec})$ ensures correct evaluation with probability ϵ if for every security parameter $\lambda \in \mathbb{N}$, every ciphertexts $c_1, \ldots, c_n$ encrypting messages $m_1, \ldots, m_n$, every $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$, we have:

$$\Pr\left[\mathsf{Dec}(sk, \mathsf{Add}(c_1, \ldots, c_n)) = m_1 + \cdots + m_n\right] = 1 - \epsilon$$

A well-known additively homomorphic encryption scheme is the Paillier cryptosystem [Pai99], introduced in 1999 and allowing to perform an arbitrary number of additions, whose plaintext space is defined in $\mathbb{Z}_n$:

- $\underline{\mathsf{KeyGen}(1^\lambda) \to (sk, pk)}$: Generates two prime numbers p and q according to λ, sets $n \leftarrow p \cdot q$ and $\Lambda \leftarrow \mathrm{lcm}(p-1, q-1)$ (*i.e.,* the least common multiple), generates the group $(\mathbb{Z}^*_{n^2}, \cdot)$, randomly picks $g \in \mathbb{Z}^*_{n^2}$ such that $M = (L(g^\Lambda \mod n^2))^{-1} \mod n$ exists, with $L(x) = (x-1)/n$. It sets $sk \leftarrow (\Lambda, M)$, $pk \leftarrow (n, g)$, it returns (sk, pk).
- $\underline{\mathsf{Enc}(pk, m) \to c}$: Randomly picks $r \in \mathbb{Z}^*_n$ and outputs $c \leftarrow g^m \cdot r^n \mod n^2$.
- $\underline{\mathsf{Add}(c_1, c_2) \to c_3}$: To compute the homomorphic addition of two plaintexts m_1 and m_2 in $\mathbb{Z}_n$, one can compute the product of the two associated ciphertexts with the public key $pk = (n, g)$, denoted $c_1 = \mathsf{Enc}(pk, m_1) = g^{m_1} \cdot r_1^n \mod n^2$ and

$c_2 = \mathsf{Enc}(pk, m_2) = g^{m_2} \cdot r_2^n \mod n^2$. Indeed, we have:

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

This function can handle n ciphertexts by applying the two-by-two addition consecutively as well.

- $\underline{\mathsf{Dec}(sk, c) \to m :}$ Computes $m \leftarrow L(c^\Lambda \mod n^2) \cdot M \mod n$, and outputs m.

Fully Homomorphic Encryption. Introduced for the first time by Craig Gentry [Gen09], fully homomorphic encryption stands as a generalisation of partially homomorphic encryption, by introducing the ability to evaluate an *arbitrary* function, instead of a single operation such as addition [Pai99] or multiplication [Elg85a]. A fully homomorphic encryption shares similarities with partially homomorphic encryption with the key generation, encryption and decryption algorithms. The major difference lies on the expressivity of the evaluation algorithm, called Eval in FHE instead of Add in AHE.

Definition 9 (Fully Homomorphic Encryption, FHE)**.** Let Π be a fully homomorphic encryption scheme be defined by the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ where:

- $\underline{\mathsf{KeyGen}(1^\lambda) \to (sk, pk, evk)}$: Given the unary representation of the security parameter λ, outputs the public key pk, the secret key sk and a public evaluation key evk.
- $\underline{\mathsf{Enc}(pk, m) \to c}$: Given the public key pk and a plaintext m, outputs a ciphertext c encrypting m.
- $\underline{\mathsf{Eval}(evk, f, c_1, \ldots, c_n) \to c}$: Given the evaluation key evk, a function f, as well as n ciphertexts, outputs an encrypted evaluation of the function f. Sometimes, we denote the evaluation of the function f over the ciphertexts $c_1, \ldots, c_n$ by $f(c_1, \ldots, c_n)$.
- $\underline{\mathsf{Dec}(sk, c) \to m}$: Given the secret key sk and a ciphertext c, outputs a plaintext m.

Correctness of Fully Homomorphic Encryption. Similarly to additively homomorphic encryption, a fully homomorphic encryption scheme has to respect two definitions to be said correct: The first definition, decryption correctness, states that the decryption of a fresh ciphertext encrypting a message m must output m.

Definition 10 (ϵ-decryption correctness for FHE [ABC+15])**.** A fully homomorphic encryption scheme Π defined as the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ ensures correct decryption with probability ϵ if for every security parameter $\lambda \in \mathbb{N}$, every $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$, every message m, we have:

$$\Pr\left[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m\right] = 1 - \epsilon$$

The evaluation correctness is similar to the decryption correctness except that the ciphertext inputted in the decryption procedure is the result of the evaluation of a function f over a set of ciphertexts. Note that the definition of evaluation correctness for a FHE scheme is a generalisation of the evaluation correctness for an AHE scheme, a meaningful statement since FHE is a generalisation of AHE by definition.

Definition 11 (ϵ-evaluation correctness for FHE [ABC+15])**.** A fully homomorphic encryption scheme Π defined as the tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ ensures correct evaluation with probability ϵ if for every security parameter $\lambda \in \mathbb{N}$, every ciphertexts $c_1, \ldots, c_n$ encrypting messages $m_1, \ldots, m_n$, every function f, every $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$:

$$\Pr\left[\mathsf{Dec}(sk, \mathsf{Eval}(evk, f, c_1, \ldots, c_n)) = f(m_1, \ldots, m_n)\right] = 1 - \epsilon$$

Security of Additively and Fully Homomorphic Encryption. Even if homomorphic encryption supports computations over encrypted data, its standard security definition remains unchanged compared to the encryption security. In other words, both AHE and FHE are supposed to be IND-CPA-secure, as stated in Definition 4.

2.3.3 Attribute-based Encryption

A practical problem in real-time streaming faced by the industry consists of sending the streamed content from a broadcasting server, to all customers having paid to receive the content. A trivial solution is to encrypt the streamed content for each user. However, in a scenario with a large number of users, this solution is impractical.

A more elegant solution should allow the broadcasting server to compute the encryption of the streamed content *once*, and then allowing *only* the subscribers to decrypt the content. Attribute-based Encryption (ABE) is a generalisation of the above scenario, allowing a fine-grained access to the plaintext: On one hand, each decryption key sk is associated to a set of attributes x. On the other hand, the encryption algorithm is now equipped with an access-policy y, producing a ciphertext c_y. Remark that a policy can be viewed as a set of *all* accepted attributes. We represent the statement "the attribute x respects the policy y" by $x \in y$.

Definition 12 (Attribute-Based Encryption, ABE)**.** An attribute-based encryption scheme is defined by the tuple $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ where:

- $\underline{\mathsf{Setup}(1^\lambda) \to (msk, mpk)}$: Given the unary representation of the security parameter λ, outputs a master key pair.
- $\underline{\mathsf{Enc}(mpk, y, m) \to c_y}$: Given the master public key mpk, the policy y and a message m, outputs the ciphertext c_y.
- $\underline{\mathsf{KeyGen}(msk, x) \to sk_x}$: Given the master secret key msk and an attribute x, outputs the secret key sk_x associated to the attribute x.
- $\underline{\mathsf{Dec}(sk_x, c_y) \to m \cup \perp}$: Given the secret key sk_x associated to the attribute x and the ciphertext c_y encrypting the message m associated to the policy y, outputs m if and only if $x \in y$, or $\perp$ otherwise.

Correctness of Attribute-Based Encryption. The correctness of an attribute-based encryption scheme states that a ciphertext c_y associated to an access policy y and

encrypting a message m, that is inputted to the decryption procedure along a secret decryption key sk_x whose attribute x satisfies y should result into the message m.

Definition 13 (ϵ-decryption correctness for ABE)**.** An attribute-based encryption scheme Π defined as the tuple $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ ensures correct decryption with probability ϵ if for every master key pair $(msk, mpk) \leftarrow \mathsf{Setup}(1^\lambda)$, every message m and access policy y, every ciphertext $c_y \leftarrow \mathsf{Enc}(mpk, y, m)$, every attribute x satisfying y and every decryption key $sk_x \leftarrow \mathsf{KeyGen}(msk, x)$, the following probability holds:

$$\Pr\left[\mathsf{Dec}(sk_x, c_y) = m\right] = 1 - \epsilon$$

Security of Attribute-Based Encryption. The security of an attribute-based encryption must reflect the same intuition compared to a traditional encryption scheme: Every polynomial-time adversary $\mathcal{A}$ trying to guess if a ciphertext encrypts either m_0 or m_1 must have a negligible chance to succeed. However, attribute-based encryption follows a different architecture with an administration owning the master secret key msk and the master public key mpk, allowed to distribute decryption keys associated to some provided attribute. Therefore, an attribute-based encryption has to be considered secure even if decryption keys are provided, except decryption keys allowed to decrypt the challenge ciphertext. In this work, we particularly focus on the IND-CCA2 security of attribute-based encryption, defined in Figure 2.3.

$\mathsf{Exp}^{\text{IND-CCA2}}_{\mathcal{A}}(\lambda)$	**Oracle** $\mathsf{OKeyGen}(msk, \mathcal{X}, y^*; x)$
$(msk, mpk) \leftarrow \mathsf{Setup}(1^\lambda)$	**if** $x \in y^*$: **return** $\perp$
$\mathcal{X} \leftarrow \emptyset$	$\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$
$\mathcal{O} \leftarrow \{\mathsf{OKeyGen}(msk, \mathcal{X}, \perp; \cdot), \mathsf{ODec}(msk, \perp; \cdot, \cdot)\}$	$sk_x \leftarrow \mathsf{KeyGen}(msk, x)$
$(y^*, m_0, m_1), \mathsf{state} \leftarrow \mathcal{A}_1^{\mathcal{O}}(mpk)$	**return** sk_x
$b \leftarrow\!\!\$\ \{0, 1\}$	**Oracle** $\mathsf{ODec}(msk, \psi_{y^*}; \psi_y, x)$
$\psi_{y^*} \leftarrow \mathsf{Enc}(mpk, y^*, m_b)$	**if** $\psi_{y^*} = \psi_y$ **then return** $\perp$
$\mathcal{O}' \leftarrow \{\mathsf{OKeyGen}(msk, \mathcal{X}, y^*; \cdot), \mathsf{ODec}(msk, \psi_{y^*}; \cdot, \cdot)\}$	$sk_x \leftarrow \mathsf{KeyGen}(msk, x)$
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}'}(\psi_{y^*}, \mathsf{state})$	$m \leftarrow \mathsf{Dec}(sk_x, \psi_y)$
return $b = b' \wedge (\nexists x \in \mathcal{X} : x \in y^*)$	**return** m

Figure 2.3: Experiment of the IND-CCA2 security for an ABE scheme.

2.3.4 Digital Signature

Digital signature is a useful cryptographic primitive allowing a user to authenticate messages. In a signature, the signer of a message holds a secret signature key sk used to compute a signature of a given message m, denoted σ_m along this manuscript. The public verification key pk, as its name suggests, is public and used to authenticate the message via the signature σ_m. Since the verification of the signature relies on the public key, the verification procedure is public. Moreover, in case where the public key is certified to be associated to a specific user, say $\mathcal{U}$, then the signature σ_m proves that $\mathcal{U}$ authenticates m.

Definition 14 (Digital Signature, Sig)**.** A digital signature scheme is defined by the tuple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ where:

- $\underline{\mathsf{KeyGen}(1^\lambda) \to (sk, pk)}$: Given the unary representation of the security parameter λ, outputs a key pair.
- $\underline{\mathsf{Sign}(sk, m) \to \sigma_m}$: Given the secret signature key sk and a message m, outputs a signature σ_m.
- $\underline{\mathsf{Verif}(pk, \sigma_m, m) \to b}$: Given the public signature key pk, a message m and a signature σ_m, outputs a bit b at one if the signature σ_m authenticates m, zero otherwise.

Correctness for Digital Signature. A digital signature scheme ensures correctness if a message authenticated by a signer can be verified by a verifier having the valid public verification key.

Definition 15 (ϵ-correctness for Sig)**.** A digital signature scheme Π defined by the tuple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ is ϵ-correct if for all security parameter λ, every $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$, every message m and every signature $\sigma_m \leftarrow \mathsf{Sign}(sk, m)$, we have:

$$\Pr\left[\mathsf{Verif}(pk, \sigma_m, m) = 1\right] = 1 - \epsilon$$

Note that the above correctness definition of digital signature is a generalisation in the sense that ϵ can be set as close to zero as desired. Indeed, the signature algorithm Sign produces a signature using the private signature key sk from which the public verification key pk is derived. Hence, the signature algorithm is able to check the consistency of the computed signature locally. If the computed signature is invalid, it suffices for the signature algorithm to restart the signature generation algorithm again until the signature verification succeeds. Assuming the probability ϵ_1 be the probability that the computed signature is invalid for a single iteration, then by iterating the signature algorithm n times, the probability ϵ_n to obtain an invalid signature after n iterations equals $\epsilon_n = \epsilon_1 \cdot ... \cdot \epsilon_1 = \epsilon_1^n$ which tends to zero. Therefore, given a carefully chosen n, the probability to obtain an invalid signature ϵ in the definition can be set as ϵ_1^n.

Security of Digital Signature. The security of a digital signature scheme relies on the hardness for an adversary to produce a signature authenticating a message m', *not* signed by the user having the secret key sk. This security is formally known as *Existential-Unforgeability under Chosen-Message Attack* (EUF-CMA) security and is defined in the Figure 2.4.

$\mathsf{Exp}^{\text{EUF-CMA}}_{\mathcal{A},\Pi,b}(\lambda)$	Oracle $\mathsf{OSign}(\mathcal{M}, sk, m)$
1 : $(sk, pk) \leftarrow \Pi.\mathsf{KeyGen}(\lambda)$	1 : $\sigma_m \leftarrow \Pi.\mathsf{Sign}(sk, m)$
2 : $\mathcal{M} \leftarrow \emptyset$	2 : **add** m **in** $\mathcal{M}$
3 : $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{OSign}(\mathcal{M}, sk, \cdot)}(\lambda)$	3 : **return** σ_m
4 : **return** $m^* \notin \mathcal{M} \wedge \Pi.\mathsf{Verif}(pk, m^*, \sigma^*) = \top$	

Figure 2.4: Security experiment for the EUF-CMA property.

Definition 16 (Existential-Unforgeability under Chosen-Message Attack, EUF-CMA)**.** Let Π be a digital signature scheme. Then, Π is said EUF-CMA if for every adversary $\mathcal{A}$ we have:

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A},\Pi} = \Pr\left[\mathsf{Exp}^{\text{EUF-CMA}}_{\mathcal{A},\Pi}(\lambda) \to 1\right] \leq \mathsf{negl}(\lambda)$$

2.3.5 Pseudo-Random Permutation

A pseudo-random permutation is a very intuitive and interesting tool in cryptography, allowing one to randomly permute an arbitrary sequence of values $v_1, \ldots, v_n$, producing the permuted sequence $v_{k(1)}, \ldots, v_{k(n)}$ for some permutation k.

Definition 17 (Pseudo-Random Permutation, PRP)**.** A pseudo-random permutation scheme is defined by the tuple $(\mathsf{Gen}, \mathsf{Perm}, \mathsf{Inv})$ where:

- $\underline{\mathsf{Gen}(n) \to k}$: Given the unary representation of the security parameter λ, outputs the permutation key k.
- $\underline{\mathsf{Perm}(k, m_1, \ldots, m_n) \to m_{k(1)}, \ldots, m_{k(n)}}$: Given the permutation key k and n elements $m_1, \ldots, m_n$, it outputs the same n elements in a permuted order, the permutation being defined by the permutation key k.
- $\underline{\mathsf{Inv}(k, m_{k(1)}, \ldots, m_{k(n)}) \to m_1, \ldots, m_n}$: Given the permutation key k and n elements in a permuted order, it outputs the same n elements in the initial order.

Security of Pseudo-Random Permutation. In contrast with other primitives, a pseudo-random permutation does not require a game formalisation. Indeed, the argument is statistical: Given a value $v_{k(i)}$ from a permuted sequence $v_{k(1)}, \ldots, v_{k(n)}$, the probability to recover i is $\frac{1}{n}$, even having access to the full permuted sequence.

2.3.6 Hash Function and Random Oracle Model

Hash functions are intensively used in cryptography, for instance to ensure integrity. Mathematically, a hash function H is a function mapping an arbitrary-sized element in $\{0,1\}^*$, to a constant-sized element in $\{0,1\}^\lambda$. The value of $H(m)$ for a message $m \in \{0,1\}^*$ is called the *hash*. Formally, a hash function is defined as follows:

$$\begin{aligned} H\colon \{0,1\}^* &\mapsto \{0,1\}^l \\ m &\to H(m) \end{aligned}$$

Due to the nature of the source ensemble, a hash function has *collision i.e.,* there always exist two distinct messages m_0 and m_1 in $\{0,1\}^*$ where $H(m_0) = H(m_1)$. To prove security, it is generally common to model hash functions in the so-called Random Oracle Model (ROM) introduced in [BR93]. This model is an idealisation of the hash function, taking the form of an oracle denoted RO, working as follows: This publicly-available oracle accepts any request containing a message m. If it is the first time that m has been seen, the oracle *randomly* chooses a value h from $\{0,1\}^\lambda$, and internally stores the couple (m, h) and returns h. In case where m has already been received during a previous request, then the oracle retrieves the couple (m, h) from its internal storage

and returns h. A direct remark is that given $h \leftarrow \mathsf{RO}(m)$, we have the guarantee that m cannot be learned, since h has been sampled at random from $\{0,1\}^\lambda$.

2.4 Tools for Security Proofs

A security property modelled in the game-based model is represented as a game, in which the adversary is asked to solve a given problem, most of the time consisting of breaking a given property. A security proof in the game-based model starts from this security game, denoted G^0. In general, proving the security in G^0 is *hard i.e.,* the claimed security is not straightforward. The security proofs consist of applying a *small* modification in G^0, obtaining G^1, and to prove that G^0 and G^1 are indistinguishable. We repeat the same operation on G^1, until reaching a game, say G^n, where it is *easy* to prove the security. This proof methodology is commonly called a *sequence of games*. The most difficult part in realising a sequence of games is to prove that for every game G^i with $i \in [\![n]\!]$, G^i is indistinguishable from the previous game G^{i-1}. From the probability to distinguish between to G^{i-1} and the G^i, it is possible to construct a so-called upper-bound on the probability to distinguish between the initial game G^0 in which proving the security is hard, and the last game G^n in which proving the security is easy. To obtain this upper-bound on the probability to distinguish, a calculation method is used on the sequence of games where probability to distinguish between each intermediate game is the same *i.e.,* on a *uniform* sequence of games. In general, the sequence of games is not uniform, but is composed of uniform sub-sequences of games. Assuming sequence of games $\mathsf{G}^0, \ldots, \mathsf{G}^n$ for a constant n, whose value does not vary during the protocol execution, then the probability for an adversary to distinguish between the first game G^0 and the game G^n, denoted by $\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^0,\mathsf{G}^n}$, is defined as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^0,\mathsf{G}^n} &= \left|\Pr\left[\mathcal{A}(1^\lambda, \mathsf{G}^0(1^\lambda)) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathsf{G}^n(1^\lambda)) = 1\right]\right| \\ &\leq \sum_{i=1}^{n} \left|\Pr\left[\mathcal{A}(1^\lambda, \mathsf{G}^i(1^\lambda)) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathsf{G}^{i-1}(1^\lambda)) = 1\right]\right|\end{aligned}$$

To obtain the probability to distinguish between a game G^i and the previous game G^{i-1}, several approaches exist, being dependent on the transition we have applied in G^i. Based on [Sho04], we distinguish three types of transitions: *Transition based on indistinguishability, transition based on failure events* and *bridging transitions*. In transition based on indistinguishability, the view of the adversary differs from the previous game G^{i-1} and the game G^i, but the adversary notices the difference only with a negligible probability. Generally, this type of transition is based on the indistinguishability assumption of some cryptographic primitive such as IND-CPA security. The transition based on failure events consists to add a failure event in the game G^i. When this happens, the adversary will be able to distinguish between the two games. This transition is particularly useful with signatures: To prove the authenticity of a message, we add a failure event in G^i aborting if a received signature is considered valid but has not been produced by the challenger running G^i. An adversary being able to forge a signature would trivially distinguish between these two games. However, if the best probability

for any adversary to forge a signature is negligible, then the probability to distinguish between G^{i-1} and G^{i} is negligible as well. This intuition is commonly known as the Difference Lemma [PS00] and is formalised as follows:

Definition 18 (Difference Lemma [PS00])**.** Let A, B and F be three events associated with the probabilities $\Pr[A], \Pr[B]$ and $\Pr[F]$. Assuming $A \wedge \neg F \iff B \wedge \neg F$ then $|\Pr[A] - \Pr[B]| \leq \Pr[F]$.

The bridging transition is crucial during sequence of games to prepare the future transitions, by integrating, removing or more generally modifying a game without modifying the view of the adversary. This last point is essential: In such transitions, the probability for any polynomial-time adversary to win the game G^{i-1} should equal the probability to win the game G^{i}, or more formally $\Pr\left[\mathsf{G}^{i-1}_{\mathcal{A}}(1^{\lambda}) \to 1\right] = \Pr\left[\mathsf{G}^{i}_{\mathcal{A}}(1^{\lambda}) \to 1\right]$.

Chapter 3

Secure Federated Multi-Armed Bandits

Contents

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 [KMea21]. 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 analyse and learn from data distributed among many owners without exposing that data*" [KMea21]. 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 arms (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 with the chosen arm. To maximise the cumulative reward, the learning agent has to continuously face the so-called exploration-exploitation dilemma: Based on values provided by the data owners referred later as the scores, it decides whether to *explore* by choosing arms with more uncertain associated scores, or to *exploit* the information already acquired by choosing the arm with the seemingly largest associated scores. The historical application of bandits was to discover the best drug within a limited number of trials, the reward drawn from the environment being the effect of the drug on the patient. Bandits have also 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 maximisation in federated multi-armed bandits*, a problem that has not been extensively studied in the literature.

We describe the federated learning architecture we are considering, depicted in Figure 3.1, starting with the localisation of the data at the heart of the federated learning model. In the federated multi-armed bandits, we assume that the *data*, distributed over K *data owners* $(\mathcal{DO}_1, \ldots, \mathcal{DO}_K)$, takes the form of reward functions associated to K bandit arms. The data and all related values are potentially sensitive, hence should not be seen in clear by any participant other than its owner. As typically done in federated learning, we assume that the learning algorithm is done by a server mentioned here as the *controller* $(\mathcal{C})$. The *user* $(\mathcal{U})$ sends a budget N to the controller and, later, receives the cumulative reward s. Moreover, we assume that the participants in Figure 3.1 (data owners, controller, and user) 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 minimising the data leakage to the controller *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.

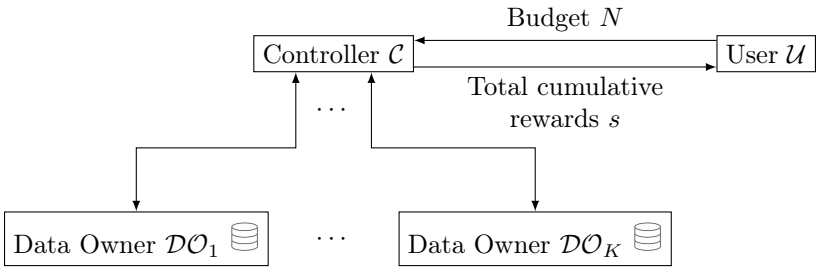


Figure 3.1: Instantiation of the federated learning paradigm for cumulative reward maximisation in multi-armed bandits.

To motivate our problem setting, we present an example based on federated learning in recommendation systems [SS21, LSF20] applied on financial investment. Suppose a user having a *budget* N (*i.e.,* the total money to be invested). The user selects K actions and lets the controller $\mathcal{C}$ decide which actions to select in order to optimise its income. For example, a positive income would be a positive reward whereas a negative income would not be rewarded. Ideally, the chosen arm should be unknown to the system, even if the chosen arm is the most optimised one. The aforementioned investment system example can be easily adapted to other classical federated learning applications where security is of paramount importance *e.g.,* commercial and medical domains [KMea21].

Our goal is to build a generic federated learning framework such that, given a bandit algorithm Alg, we are able to plug Alg in our framework and obtain the same cumulative reward as Alg, while guaranteeing data security. This interesting genericity property has already been put forward in the literature by our initial publication [CLMS22], introducing the Samba framework focusing on the reward maximisation problem specifically in the architecture we are considering in this work. Theoretically, any multi-armed

bandits algorithm satisfying a precise definition is guaranteed to be plugged within the Samba framework, enjoying all the claimed security properties without loss of correctness. However, as we will show later, Samba suffers from correctness and security issues, allowing for instance the federation server to learn the scores of the data owners. Badly, Samba is currently the most recent and state-of-the-art protocol for secure federated multi-armed bandits. Furthermore, to the best of our knowledge, no security definition has yet been suggested for secure federated multi-armed bandits. Such a definition is necessary in order to be able to compare existing protocols.

Summary of Contributions

Our contribution starts with the introduction of the formal security definition of a secure federated multi-armed bandits in the computational model following the real-ideal paradigm. Equipped of our security definition, we propose two secure federated multi-armed bandits protocols Tango and Salsa, following two distinct philosophies.

Our first secure federated multi-armed bandits protocol Tango aims to fix the correctness and security issues of Samba, with the ambition to remain close to the Samba framework. In a nutshell, our fix consists of replacing a flawed mask technique being at the heart of the security of Samba, with fully homomorphic encryption. Briefly, this approach allows us to delegate the best arm identification task to a server called the proxy, over encrypted scores, instead of a second server, solving at the same time the correctness and the security issues found in Samba. Hence, thanks to our fix, Tango enjoys the following properties: *Genericity, correctness and security.* In addition, we have identified an easy-to-achieve and interesting property that we refer in this work as the *resistance against failures* property, which allows a protocol to return (almost) the same total cumulative rewards, even in a scenario where a data owner fails and leaves the protocol before the end. Despite all these interesting properties, Tango suffers from a high latency due to the homomorphic computations, making it less valuable for applications whose arm selection is expected to be fast.

Our second secure federated multi-armed bandits protocol Salsa drastically moves away from the blueprint of Samba and Tango. Indeed, whereas Samba and Tango rely on cryptographic primitives such as encryption, Salsa bases its correctness and security on the secure two-party computations, in which two servers evaluate an arbitrary circuit on private input without learning any information on the input. Briefly, each input is split into two shares, one for each server involved in the computations. Given a share of every input, the two servers are able to evaluate the circuit directly on the shares. The evaluation on the shares results into new shares which once combined, provide the desired output. Compared to homomorphic encryption, computation over shares offers high-performance. Interestingly, in Salsa a data owner does not have to setup any key.

In contrast with Samba, we have chosen to design Tango and Salsa to deal with *discrete* multi-armed bandits. Indeed, the scores in multi-armed bandits algorithms are traditionally represented as floating-point values. Discretizing a problem is an already studied and approved solution to make it easier to solve using standard techniques, for instance applied on discrete deep neural network inference [BMMP18, BGGJ19, SFB+23, TTS+24]. Both protocols Tango and Salsa ensure the same properties mentioned ear-

Data \ Participant	$\mathcal{DO}_i$	$\mathcal{DO}_{j\neq i}$	User ($\mathcal{U}$)	Controller ($\mathcal{C}$)
Total cumulative rewards s			X	
Cumulative rewards s_i for $\mathcal{DO}_i$	X			
Number of pulls n_i for $\mathcal{DO}_i$	X			
Score of $\mathcal{DO}_i$ at time t	X			
Pulled arm index M at time step t	X*			

Figure 3.2: Security properties that we aim to respect. By X, we mean that the participant can see in clear the concerned piece of data, whereas an empty case means the opposite. The $\star$ means that the $\mathcal{DO}_i$ only knows the pulled arm index M if and only if it is pulled at time step t.

lier, namely genericity, correctness, security and resistance against failures. We describe how Tango and Salsa achieve all these properties.

Genericity. The genericity property enjoyed by Tango and Salsa allows any discrete bandit algorithm that satisfies the two following properties to be plugged: (1) Computing the score of an arm does not depend on the other arms, and (2) selecting the arm to be pulled is performed using an arg max function. We stress that these two properties, and more particularly the second one actually, reject multi-armed bandits algorithms initially supported by Samba. This choice is motivated by our security definition being designed specifically for arg max-based multi-armed bandits. In practice, other multi-armed bandits might be plugged both in Tango and Salsa but would require a dedicated or a more generalised security definition that goes beyond this initial contribution. In the literature, three multi-armed bandits meet these conditions, namely UCB, Thompson Sampling and ϵ-greedy.

Correctness. The correctness property offered by Tango and Salsa ensures that the returned total cumulative rewards of a multi-armed bandits algorithm plugged in Tango is the same as the total cumulative rewards returned by the standalone execution of the discrete multi-armed bandits algorithm. In Tango, correctness is achieved via homomorphic encryption, allowing to operate directly over the encrypted data. Once decrypted, the obtained result equals the desired output with very high probability. In Salsa, the computations are no more performed on the encrypted data whose computation incurs a large execution time overhead. Rather, computations are performed directly over the so-called shares using secure two-party computation techniques [DSZ15], offering correctness and high-performance.

Security. The presented protocols are strengthened by a formal security analysis in the computational model, whose security properties are summarised in Figure 3.2. In details, for each time step, the score of each data owner remains hidden, a crucial property since a score can be deterministic (for instance with the ϵ-greedy algorithm) following a small and non-uniform distribution. In addition, the pulled arm index M should not be identified both by the controller but also by other data owners (different of the one that has been pulled). In other words, only the pulled data owner knows that it has been chosen, otherwise has one chance over $K-1$ to identify the pulled data

owner. Similarly, the cumulative rewards s_i as well as the number of pulls n_i for a data owner $\mathcal{DO}_i$ should remain private and unknown to all other parties. In contrast, the total cumulative rewards s calculated as the sum of all cumulative rewards s_i, should be unknown to all parties except to the user $\mathcal{U}$ expecting the total cumulative rewards at the end of the protocol execution.

Resistance Against Failures. By the nature of federated learning, a data owner is sensitive to failure and hence to be disconnected from the protocol before the end of its execution. Badly, due to the considered federated learning architecture and data distribution, a data owner leaving a running protocol execution directly impacts the total cumulative rewards returned to the user $\mathcal{U}$. Surprisingly, this interesting property has never been studied in the context of federated multi-armed bandits. To motivate our interest to tackle this issue, we introduce the following example with three data owners having the following cumulative rewards s_1, s_2 and s_3 respectively set at 4, 6 and 10. Suppose that the third data owner has left the protocol before the end of the protocol. The worst case for the user consists of receiving a total cumulative rewards s equal to $s_1 + s_2$ which equals 10, half of the expected total cumulative rewards of 20 in our example, due to the data owner $\mathcal{DO}_3$ being offline and hence not able to share its cumulative rewards s_3. Thanks to the design of Tango or Salsa, we are able to handle this case and still to return the exact total cumulative rewards 20. We stress that this case in the most desirable. As we will see later, the mechanism we have put in place to solve this issue consists of regularly and privately updating a register maintained by the federation server. We do not want to oversell our reward saving mechanism: There is a case where a reward is definitively lost, if a chosen data owner $\mathcal{DO}_i$ lefts the protocol before registering its cumulative rewards s_i. This edge case is impossible to avoid since in our setting $\mathcal{DO}_i$ is the only one party allowed to obtain a reward based on its reward probability μ_i. That is, using this reward saving mechanism, the returned total cumulative rewards s equals $\sum_{i=1}^{K} s_i - h$ where h is the number of data owners *having left the protocol without updating the register*. This lost of reward h is independent of every cumulative rewards s_i which constitutes a substantial improvement compared to the lost of every cumulative rewards s_i. Tango and Salsa support dynamic joins as well. Indeed, in both protocols we do not leverage on any commonly-shared secret which stands in contrast with Samba for which a shared secret-key is required to join the protocol execution, as we will see in a moment.

Overall Description of Tango. A secure federated multi-armed bandits protocol is divided into two distinct parts: The *arm selection* and the *total cumulative rewards sending*. As its name suggests, the arm selection, replayed as many times as the budget provided by the user, consists of identifying the next arm being pulled. In this phase, every data owner inputs the arm selection protocol with a score, the arm selection protocol being played between all the data owners and the controller $\mathcal{C}$, and expects as a response a selection bit. At time step $t \in [\![N]\!]$, the selection bit received by the data owner $\mathcal{DO}_i$ is denoted as $b_{i,t}$. The constraint is that only one selection bit is supposed to equal one in order to model the fact that a single arm can be chosen at each time step. In Tango, this arm selection protocol relies on the controller $\mathcal{C}$ which is splitted

into two nodes, namely a proxy node $\mathcal{P}$ and a reflector node $\mathcal{R}$. Briefly, the proxy node $\mathcal{P}$, observing only encrypted messages, receives K scores along K encrypted secret-keys, K being the number of data owners. The proxy node $\mathcal{P}$ performs a homomorphic comparison on the K encrypted scores resulting into the encryption of the best arm index M used to construct K encrypted selection bits. These K encrypted selection bits and these K encrypted secret-keys are later shared with the reflector node $\mathcal{R}$ able to decrypt each of these ciphertexts. The only goal of the reflector node $\mathcal{R}$, as its name suggests, is to decrypt the obtained selection bits and to re-encrypt them again, with the appropriate received and decrypted secret-key. All these selection bits, encrypted using the provided secret-key, are sent back to the proxy node $\mathcal{P}$ before being shared with the appropriate data owner. The interesting point is that none of these two nodes are able to identify either the bandit arm scores or the pulled arm at some time, identified by the single positive selection bit. This is achieved for the proxy node $\mathcal{P}$ thanks to the indistinguishability of the used FHE scheme, but also against the reflector node $\mathcal{R}$ thanks to a permutation, randomly chosen and applied by the proxy node $\mathcal{P}$, permuting the encrypted selection bits and on the encrypted secret-keys. In a sense, these two nodes are preventing each other to learn any information while performing the desired arm selection functionality.

At the end of the protocol execution, when all the budget has been spent, the total cumulative rewards sending is executed. This phase is designed to securely send the total cumulative rewards s corresponding to the sum of all the cumulative rewards s_i locally stored by each data owner to the final user $\mathcal{U}$. Since the correctness and security issues of Samba [CLMS22] does not affect this phase, we have decided to not modify the total cumulative rewards sending phase, relying on additively homomorphic encryption such as the Paillier cryptosystem [Pai99]. Hence, the total cumulative rewards s is obtained by summing up the cumulative rewards s_i from each data owner directly in the encrypted domain. Hence, neither the data owners nor the server nodes can see in clear the total cumulative rewards: Only the user $\mathcal{U}$ that invested a budget for computing the total cumulative rewards is able to decrypt it.

Overall Description of Salsa. In contrast with Tango, Salsa does not rely on cryptographic primitives such as (homomorphic) encryption but on secure two-party computations. In the vein of the ABY framework [DSZ15, PSSY21], every computations in Salsa are performed over the so-called shares. In few words, in secure two-party computations, two shares are derived from a private data, one share being insufficient to recover the initial private data. Each of these shares are sent to a different computation server. Then, both servers are executing an arbitrary circuit in a synchronised manner to update their respective shares with respect to the executed circuit. As a result, each server obtains a share which once combined, recovers the desired output of the executed circuit on the initial private data.

In our setting, the computation servers correspond to the controller $\mathcal{C}$ being splitted into two nodes $\mathcal{C}_0$ and $\mathcal{C}_1$. During the arm selection protocol, every data owner $\mathcal{DO}_i$ computes the two following shares denoted $\langle v_{i,t} \rangle^0$ and $\langle v_{i,t} \rangle^1$ of its score $v_{i,t}$, the first share being sent to $\mathcal{C}_0$ and the second share to $\mathcal{C}_1$. Given the shares of every score, the computation servers evaluate an arm selection algorithm which is essentially an arg max

producing a best arm index M followed by K equality tests, one for each arm index i ranging from 1 to K. And since the equality test between the best arm index M and the arm index i results into a bit 1 (*i.e.,* M equals i) or a bit 0 (*i.e.,* M not equals i), this circuit results into K bits that we refer as the selection bits. By construction, only the bit of the chosen arm, b_M, equals one. Since every computation in SALSA is performed on shares, the resulting selection bits are shared among the two servers $\mathcal{C}_0$ and $\mathcal{C}_1$, and hence remain unknown from the computation servers, under the assumption that $\mathcal{C}_0$ and $\mathcal{C}_1$ do not collude. In particular, the first server $\mathcal{C}_0$ knows $\langle b_{i,t} \rangle^0$ for every data owner $\mathcal{DO}_i$ whereas the second server $\mathcal{C}_1$ knows $\langle b_{i,t} \rangle^1$ for every data owner $\mathcal{DO}_i$. The shares of each $b_{i,t}$ is sent to the i-th data owner, which now has access to the two shares $\langle b_{i,t} \rangle^0$ and $\langle b_{i,t} \rangle^1$ and hence is able to reconstruct $b_{i,t}$. We have used a similar efficient sharing technique to perform the total cumulative rewards sending, which stands in contrast with SAMBA and TANGO relying on additively homomorphic encryption.

Related Work

Our work takes position to the federated learning setting in which many parameters can be adjusted, including for example the data distribution or the behaviour of the data owners. We first precise our federated learning setting in which this work takes place. Then, we compare our work with existing approaches developed by the cryptographic community which can be used for the reward maximisation problem. Finally, we focus on existing secure federated multi-armed bandits protocols.

Positioning in the Federated Learning Paradigm. Several parameters can be adjusted in the federated learning setting. We position our work for each of these parameters defined in the influential federated learning survey published in [KMea21]:

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

Moreover, among the main federated learning settings (cross-silo vs cross-device), our framework pertains to the *cross-silo federated learning setting* [KMea21], whose typical characteristics are:

- *Distribution scale.* There are rather few data owners, which can be different organisations *e.g.,* stores, hospitals.
- *Data owner failure.* Each data owner is allowed to be offline at some point during the protocol execution.
- *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 a federated learning *threat model*, we assume that all participants (data owners, user and controller) 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 [Gol04] (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 exchanged messages.

Positioning w.r.t. Federated and Secure Bandits. Being an active research 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 [KMea21], 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* [SXXS21], but without considering the data security aspect. In their setting, all $\mathcal{DO}$s share the same K arms where an arm i yields different rewards from a $\mathcal{DO}$ to another. The goal of each $\mathcal{DO}$ 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 $\mathcal{DO}$s and use them to compute the arm with the highest average reward over all $\mathcal{DO}$s. (ii) Another study tackles the *robustness with respect to Byzantine adversaries* in federated best arm identification [MHP21]. In their setting, each $\mathcal{DO}$ has access to a subset of the K arms and to ensure robustness, their method implies that each subset of arms is sampled by multiple $\mathcal{DO}$s. In contrast, in our proposed framework TANGO, the data of a $\mathcal{DO}$ is locally stored and never exchanged, hence it is hidden from all the other participants. Furthermore, a main characteristic is that TANGO does not affect the arm selection strategy compared to the standard (non federated, non secure) discretized bandit algorithm.

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 [SS21, LSF20], bandits with graph structure [ZZLL21], and linear bandits [DP20, HWYS21]. 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 [DR14] to protect the data.

In particular, the first works on cumulative reward maximisation in (private) federated multi-armed bandits [SS21, LSF20, ZZLL21, SSY21] focus on the analysis of the gain in sharing data coming from multiple $\mathcal{DO}$s for obtaining better *local* ($\mathcal{DO}$-specific) and respectively *global* cumulative rewards (for all participants in the federated learning process). The typical assumption is that all $\mathcal{DO}$s 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 $\mathcal{DO}$s *exchange information about the rewards they observe and about the indices of their selected arms* with their neighbours [LSF20, ZZLL21], respectively with the central orchestration server [SS21, ZZLL21]. Before sharing these pieces of information, as done in [RBCS23], $\mathcal{DO}$s 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 discretized bandit algorithm. Even more, the injected noise has to be carefully computed to ensure differential privacy while limiting the regret. In our cryptographic-based approach, the security is independent of the used multi-armed bandits algorithm. 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 maximisation, we focus on a vertical data partition and our secure framework guarantees that local data maintained by each $\mathcal{DO}$ 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 [CLLS20, CLLS19, CLMS22]. The protocols of [CLLS19, CLLS20] also consider the problem of secure cumulative reward maximisation for standard stochastic bandits, but in a considerably different architecture. The protocol of [CLLS19] considers a single data owner holding all the arms. In a nutshell, the protocol of [CLLS20] supposes a cloud divided into $K+1$ parties: K arm nodes, each arm node denoted R_i having access to its own reward probability μ_i in clear, provided by the data owner; but also an orchestration node. The arm selection protocol proceeds using a ring communication round: Each arm node is inputted from another randomly chosen arm node with the current best arm index M and its score B_M. It inputs the next arm node either with the couple (i, B_i) when B_i is higher than B_m, or it simply forwards the couple (M, B_M) to the next arm node otherwise. The last arm node in the chain only forwards the index M to the orchestration node, which responds to each arm node with the appropriate selection bit. The order of the arm nodes in the chain is randomly chosen by the orchestration server and is changed at each round. We note major differences between [CLLS20] and our protocol Tango. First, the data distribution assumptions are different: They assume that all data is outsourced to the cloud, whereas Tango focuses on a federated learning setting

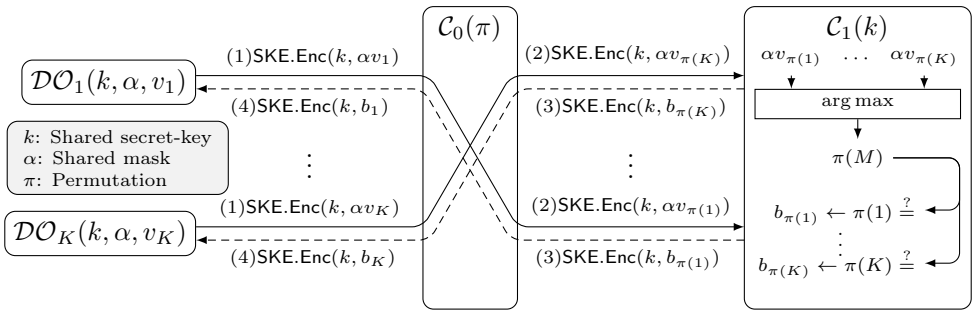


Figure 3.3: Overview of SAMBA for arg max-based multi-armed bandits.

where data is stored locally by each owner and never exchanged. In addition, due to the federated learning setting, communication between the data owners (corresponding to the arm nodes in [CLLS20] but being now independent of the cloud) are no more allowed. Consequently, the respective distributed architectures are intrinsically different. Second, their protocol is catered for securing the UCB algorithm, whereas TANGO is a generic framework where multiple bandit algorithms can be plugged in. Among the algorithms supported in TANGO, we have UCB and similar argmax-based algorithms. Last but not least, the protocol of [CLLS20] as well as the work of SAMBA [CLMS22] may not return the *exact* output compared to traditional UCB algorithm, whereas our construction does. We elaborate on this correctness issue below.

Positioning w.r.t. SAMBA. Introduced in [CLMS22] and depicted in Figure 3.3, SAMBA is the closest work to our protocol TANGO. Indeed, TANGO aims to fix and improve SAMBA in several points, which explains the proximity between these two protocols. Briefly, SAMBA combines two-party computation and (additively only) homomorphic encryption, following the same cross-silo data partition. SAMBA solves the best arm identification for federated multi-armed bandits by using two nodes for the server (as done in this work), that we denote simply as $\mathcal{C}_0$ and $\mathcal{C}_1$. The arm selection protocol starts with every data owner $\mathcal{DO}_i$ having its score v_i. To hide its score v_i to the system, SAMBA assumes the existence of *shared mask* denoted α known by every data owners and different at each time step. This mask α is multiplied with the score v_i, leading to the value αv_i referred later as the *masked score*. Once computed, every masked score is encrypted under a secret-key encryption scheme SKE using a secret-key k, carefully shared between all the data owners and the server $\mathcal{C}_1$. Observe that at each round, *the used mask is the same for all data owners* thanks to a previously shared seed that we omit for simplicity. The first node $\mathcal{C}_0$ receives an encrypted masked score from each data owner, constructs a list of these ciphertexts (encrypting the masked score) and permutes this list using a permutation, denoted π in Figure 3.3. The resulting permuted list of ciphertexts is then shared with the second node $\mathcal{C}_1$ having the shared secret key (unknown from $\mathcal{C}_0$). After the decryption of each ciphertext, $\mathcal{C}_1$ obtains a permuted list of masked scores that are compared to identify the best arm. Let M be the index of the best arm, and $\pi(M)$ be the permuted best arm index observed by $\mathcal{C}_1$. From $\pi(M)$, $\mathcal{C}_1$ constructs a list of K selection bits $b_1, \ldots, b_K$ where only $b_{\pi(M)}$ equals one. These selection bits are forwarded to the first node $\mathcal{C}_0$ which invert the permutation before to

forward each selection bit to the appropriate data owner, having the shared decryption key to recover the selection bit.

We omit the other part of SAMBA since two problems arise here. In SAMBA, the (multiplicative) mask is used in the spirit of the one-time pad to prevent $\mathcal{C}_1$ to recover the real value of the scores. However, recall that the mask is the same for every data owners and hence corresponds to a K-time pad, insecure by definition. Indeed, $\mathcal{C}_1$ can easily derive $\frac{v_1}{v_2}$ by evaluating $\frac{\alpha v_1}{\alpha v_2}$. For this reason, SAMBA does not provide secrecy of the scores thus does not provide secrecy of both the number of pulls n_i and the cumulative rewards s_i. In addition, due to the multiplicative property of the mask, a score at zero is always revealed, since the mask as once chance over the large mask space to equal zero.

The second issue in SAMBA occurs when at least two scores v_i and v_j are equal in the list of scores. For the sake of clarity, we exemplify this issue with three data owners having respectively the reward probabilities $\mu_1 = 0.75, \mu_2 = 0.25$ and $\mu_3 = 0.1$. Suppose that we have chosen the ϵ-greedy algorithm, leading at the first time step $t = 1$ to this list of the scores $v = [v_1 = 1, v_2 = 1, v_3 = 0]$. Following the standard ϵ-greedy definition in its exploitation stage selecting an arm based on the arg max function, the chosen arm should be the first one. However, recall that in SAMBA, $\mathcal{C}_0$ permutes the list of scores, so suppose the permuted list $\pi(v) = [v_2 = 1, v_3 = 0, v_1 = 1]$. In this case, the arg max evaluation results into the second arm instead of the first arm v_1. That is, instead of sampling a reward from the Bernoulli distribution parameterised by the reward probability $\mu_1 = 0.75$, the reward distribution is parameterised by the reward probability $\mu_2 = 0.25$. In the general case, this invalid arm selection does not result in a significant reward difference, but it is sufficient to invalid the claim stating that SAMBA returns the *exact* same reward compared to the standard ϵ-greedy. We stress that this correctness issue also happens in [CLLS20] again due to a random permutation defining the chain of data owners in the ring. In addition to these two issues, due the shared seed and shared secret key, a data owner trying to join the protocol needs first to obtain these shared values. This stands in contrast with our protocols TANGO and SALSA, which at the cost of a reduced set of three supported algorithms (instead of five for SAMBA), we provide at the same time correctness with respect to a discretized multi-armed bandits algorithm, privacy-preserving with respect to the score, but also supports the dynamic join and failure of a data owner during the protocol execution.

Positioning w.r.t. Homomorphic Encryption. Theoretically, our reward maximisation problem itself related to the best arm identification problem, could be solved by a single-server running all the best arm identification algorithm homomorphically. Homomorphic encryption allows a natural and non-interactive class of protocols in which each data owner sends an encrypted score to the server to perform the computation. In contrast to multi-party computations, homomorphic encryption needs encryption and decryption keys which have to be correctly managed to prevent data leakage. The initial definition of homomorphic encryption as suggested by Gentry [Gen09] is referred here as a *single-key* fully homomorphic encryption, perfectly fitting the client-server setting where the client wants to delegate the computation of a function f over a private data x to an honest-but-curious and powerful server. However, our federated learning setting

supposes several clients *i.e.,* the K data owners, and it is not desirable to assume a shared secret decryption key between all the data owners, as supposed in the SAMBA framework. Homomorphic encryption for multiple clients precisely solves this issue and is an active research area [CDKS19, CCS19, Par21, KMS22, AKO24]. Based on the work of Kwak *et al.* [KLSW24], we distinguish two paradigms for multi-clients homomorphic encryption, namely Multi-Parties Homomorphic Encryption (MPHE) and Multi-Key Homomorphic Encryption (MKHE). The difference between these two paradigms occurs at the key generation for the different parties: In MPHE, a key generation protocol is executed between all clients resulting for the i-th client to obtain the secret key sk_i and a joint public key pk while in MKHE each client generates its own key pair (sk_i, pk_i) independently of the other clients. In both paradigms, the decryption of a ciphertext requires all the decryption keys [KLSW24] which requires the collaboration of all the data owners to decrypt the result. Since we have assumed the potential failure of a data owner, a secret-key sk_i may be missing, making the resulting ciphertext undecipherable for the other data owners. In addition, multi-clients homomorphic encryption leads to a reduced performance compared to the single-key homomorphic encryption. For instance, the NAND gate evaluation using single-key variant of TFHE [CGGI19] takes approximately 8 milliseconds, while the multi-clients variant [KMS22] takes approximately 270 milliseconds for four parties. Our first protocol TANGO, fixing the SAMBA protocol using fully homomorphic encryption, only requires a single-key homomorphic encryption scheme by splitting the controller $\mathcal{C}$ (*i.e.,* the federation server) into two non-colluding federation servers called respectively the proxy node $\mathcal{P}$ and the reflector node $\mathcal{R}$. This approach allows TANGO to rely on more efficient fully homomorphic encryption schemes and to avoid a complex decryption procedure.

Positioning w.r.t. Secure Multi-Party Computations. Identifying the best arm can be transposed to the so-called *Millionaire problem*, introduced by Yao [Yao82]: In its original two-party definition, the problem consists for two millionaires to define which of the two millionaires is the richest one, without revealing the value of their respective fortunes. Stated as above, this problem does not directly fit our needs since we are solving the best arm identification for at least three data owners. Indeed, in case when K equals two, a data owner can easily infer private data on the other data owner. For instance, a data owner being pulled, let's say 4 times with a budget N of 6, infer the number of times the other arm has been pulled by subtracting N with its local number of pulls, resulting to 2. While the Yao protocol [Yao82] considers of two millionaires, we are addressing the more general case with K higher than two bandits. Recently, Tassa and Yanai [TY24] has introduced a cryptographic solution to address the millionaire problem for an arbitrary number of millionaires based on Arithmetic Black-Box (ABB) modelled as an ideal functionality, able to evaluate any n-fan gates while being privacy-preserving. At this point it becomes essential to understand the meaning of the comparison. When applied on two values, let say u and v, the two-fan gate comparison denoted $u \leq v$ is expected to return a bit at 1 if v is higher or equal than u, 0 otherwise. A problem occurs when generalising the comparison operator to n values, particularly in cases where two or more values are equal. Indeed, following the definition of [TY24], a comparison between n values results for the i-th millionaire

to obtain a bit b_i at 1 if the i-th millionaire is the highest fortune. In other words, by denoting the fortune of the n millionaires by $v_1, \ldots, v_n$ and v_M being the highest fortune, a multi-millionaires protocol responds positively for *every* millionaire i when v_i equals v_M. Unfortunately, by definition, the equality case results for two or more millionaires to obtain a bit at 1. This behaviour is absolutely not desired in our case because of a user can spent only one coin to a bandit at a time. During a multi-armed bandits algorithm execution, especially at the beginning, two or more bandits have a non-negligible probability to obtain the same score. This is particularly true for the ϵ-greedy algorithm whose score *i.e.,* the performance of the arm, is computed using the empirical mean defined as $v_i = \frac{s_i}{n_i}$ where s_i corresponds to the cumulative rewards and the n_i is the number of times the arm has been pulled (note that both s_i and n_i lives in $\mathbb{N}$ and $s_i \leq n_i$). In this setting when n_i equals 1, the score of each arm is either 0 or 1. In a scenario with at least three bandits, we have necessarily a collision, and the multi-millionaires protocol fails in case where all scores v_i equal 0. Because of this restriction to select exactly *one* bandit at a time, protocols solving the multi-millionaires problem are not directly suitable for multi-armed bandits.

In the same spirit of [TY24], Demmler, Schneider and Zohner have introduced the ABY framework [DSZ15], later improved in [PSSY21], supporting a large variety of computations, able for instance to solve the multi-millionaires problem. Because of its versatility, the ABY framework represents a great candidate to solve the secure federated multi-armed bandits problem. Since the secure federated multi-armed bandits has been focused by only few papers in the literature, to the best of our knowledge, currently no work has put forward a secure federated multi-armed bandits protocol based on secure multi-party computation. This is exactly the motivation of our second protocol Salsa, relying intensively on secure *two*-party computations, and more precisely on (a part of) the ABY framework.

Chapter Organisation

In Section 3.1, we introduce the basic notions on bandit algorithms. In Section 3.2, we introduce a formal security model, clearly defining the expected security of a secure federated multi-armed bandits. In Section 3.3, we present our first protocol Tango, prove its correctness with respect to the discretized multi-armed bandits, its security but also an empirical study of Tango. Finally, in Section 3.4, we present our second protocol Salsa, including a correctness, security and empirical performance analysis.

3.1 Multi-Armed Bandits Algorithms

The historical motivation [Tho33] 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. Maximising the sum of observed rewards means maximising 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 [LCLS10] to game playing [KS06]. In this chapter, we consider the typical setting of stochastic multi-armed bandits with Bernoulli rewards. Next, we introduce the notation related to bandit algorithms.

Notations for Multi-Armed Bandits. 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 $pull(\cdot)$ 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 maximising the sum of rewards. We illustrate Tango using a selection of three textbook algorithms [SB18, KP14, RVK$^+$18], which represent a variety of strategies. To minimise redundancy when presenting the aforementioned collection of algorithms, we present what is common to all of them in Figure 3.4. 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 M for each time t, that we present in Figure 3.5. We stress that at each time t only one arm is pulled, thus only the corresponding variables s_M and n_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 Alg we mean, depending on Alg the argument of the argmax.

Generic Multi-Armed Bandits Model

To be suitable in our framework, a multi-armed bandits algorithm has first to fit a federated setting, excluding existing multi-armed bandits algorithm that cannot be instantiated in such a setting. To precisely define what we mean by "suitable in our framework", we have introduced a general multi-armed bandits model tailored for our federated learning setting, featuring *local arm scores* and *best arm selection* properties. Let us focus on the first property: While the basic Alg algorithm is executed by a single party, the federated version states that each arm is represented by a distinct party, namely a data owner $\mathcal{DO}_i$, having only access to its internal state including the cumulative rewards s_i and the number of pulls n_i, to produce a score. This observation excludes directly every multi-armed bandits algorithm whose score computation is not exclusively dependent of s_i and n_i, as done in the Reinforcement Comparison [SB18] algorithm. The second property is focused on the arm selection once the score is computed, and is inher-

```
    /* Initialization: pull each arm once & initialise variables */
1 : for i ∈ [[K]]
2 :   r ← pull(i)  /* Random reward for arm i */
3 :   s_i ← r  /* Sum of observed rewards for arm i */
4 :   n_i ← 1  /* Number of pulls of arm i */

    /* Exploration-exploitation: pull one arm at each time step t */
5 : for t ∈ [[N]]
6 :   Choose M according to algorithm Alg
7 :   r ← pull(M)
8 :   s_M ← s_M + r
9 :   n_M ← n_M + 1

10 : return s_1 + ... + s_K
```

Figure 3.4: Generic cumulative reward maximisation with bandit algorithm Alg.

ently dependent of our framework capabilities. In particular, our framework precisely focuses on argmax-based multi-armed bandits algorithms. This is a necessary condition to provide a complete and formal security analysis of our scheme. We stress that despite these constraints, three multi-armed bandits algorithms fit our model, denoted MAB. Note that an algorithm that can be instantiated in this generic model can be used in our framework directly. Our generic model is composed of five suggestive algorithms Init, ScoreArm, BestArm, SelectArm and PullArm, presented in Definition 19 that we now describe.

The Init algorithm, called only once at the beginning of the MAB execution, is used to initiate the state of an arm composed of only two variables: The cumulative rewards s_i and the number of pulls n_i. As depicted in Figure 3.4 between Line 1 and Line 4, each arm is pulled once to construct a preliminary (yet limited) knowledge on the rewards probabilities for all arms. Recall that the pulling of an arm is based on a hidden and unknown reward probability μ. In our model, we assume the existence of the *pull* algorithm which given no parameter outputs a reward (zero or one) depending on the unknown reward probability μ. This *pull* algorithm is made available inside the Init algorithm. The output of Init, namely the number of pulls n_i is assumed to equal 1, while the cumulative rewards s_i may equal either zero or one depending on the unknown reward probability μ_i, depending on the output of the *pull* algorithm.

The identification of the best arm is based on a score denoted v_i with i being the index of the data owner $\mathcal{DO}_i$. To fit our federated setting in which a data owner $\mathcal{DO}_i$ has only access to its cumulative rewards s_i and its number of pulls n_i, the score of a candidate multi-armed bandits must be computable from s_i and n_i, and nothing else. For this reason, we have introduced the ScoreArm algorithm expecting as an input for a data owner $\mathcal{DO}_i$ the number of the cumulative rewards s_i and the number of pulls n_i. The current time step t is also inputted to include multi-armed bandits algorithm such as UCB [SB18] whose score depends on the time step t.

Since we are focusing argmax-based multi-armed bandits algorithms, we have introduced the MAB.BestArm which given K scores outputs the index M of highest score. While this algorithm can be omitted and replaced by the arg max function, as done in our framework TANGO, the MAB.BestArm is necessary for the completeness of our model. To fit in our framework, the index of the best arm M is not sufficient. Indeed, in our framework, we expect each arm to receive a *selection bit* notifying if the arm has been chosen. This transformation from best arm index M to a list of K selection bits is performed by the SelectArm algorithm, expected as an input the best arm index M, the current time step t, and outputting K selection bits $b_1, \ldots, b_K$.

A chosen arm represented by a data owner $\mathcal{DO}_i$ has to pull a reward r, used to update its cumulative rewards s_i and its number of pulls n_i. For this purpose our model includes the PullArm algorithm used specifically to obtain a reward using *pull* algorithm, to increment the number of pulls n_i by one and, in the eventuality that the obtained reward r equals one, to increment the cumulative rewards s_i by one as well.

Definition 19 (Generic Multi-Armed Bandits)**.** A multi-armed bandits fits our generic model if it can be instantiated using the following algorithms:

- MAB.Init() $\rightarrow (s_i, n_i)$: Given no parameter, this algorithm outputs n_i the number of times that the arm has been pulled along the cumulative rewards s_i. Since each arm is supposed to be pulled once at the beginning, we expect n_i to equal 1.
- MAB.ScoreArm$(s_i, n_i, t) \rightarrow v_{i,t}$: Given the cumulative rewards s_i, the number of pulls n_i as well as the current time step t, it outputs the score $v_{i,t}$ of the arm i.
- MAB.BestArm$(v_1, \ldots, v_K) \rightarrow M$: Given K scores v_i for i in $[\![K]\!]$, it outputs the index M of the best arm. In other words, M equals $\arg\max(v_1, \ldots, v_K)$.
- MAB.SelectArm$(M, t) \rightarrow b_1, \ldots, b_K$: Given the index $M \in [\![K]\!]$ corresponding to the best arm and current time step $t \in [\![N]\!]$, outputs n bits where every b_i equals 0, except b_M which equals 1, corresponding to the chosen bandit.
- MAB.PullArm$(t, b_i, s_i, n_i; \mu_i) \rightarrow (s'_i, n'_i)$: Given the current time $t \in [\![N]\!]$, a selection bit b_i, the cumulative rewards s_i and the number of pulls n_i, outputs the updated cumulative rewards and the updated number of pulls. We stress that when b_i equals 0, the both s'_i and n'_i equal respectively s_i and n_i. In the other case where b_i equals 1, n'_i equals $n_i + 1$ and s'_i equals $s_i + r_{i,t}$ for some value $r_{i,t} \in \{0, 1\}$ randomly obtained depending on the (hidden and unknown) reward probability μ_i. This reward probability μ_i is considered most of the time unknown. Later, this probability will be provided by the adversary hence it is interesting to model this probability.

By construction, a multi-armed bandits algorithm fitting in our MAB model needs to feature two properties. The first property, that we call the *arm score locality* property, stating that the score v_i for a data owner $\mathcal{DO}_i$ can be computed only from the cumulative rewards s_i and the number of arms n_i, as well as the current time step t. It also suggests that the arm selection should be performed only based on the highest score, called the *best score selection* property. This second property is necessary since our framework is specifically designed to select an arm based on the highest score. We show that three multi-armed bandits algorithms are indeed instantiable in our generic MAB model and

Algorithm Alg	*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** $\arg\max_{i \in [\![K]\!]} \theta_i$

Figure 3.5: Instantiation of three cumulative reward maximisation algorithms. Recall that for each arm $i \in [\![K]\!]$, 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.

hence in our framework, namely ε**-greedy**, **UCB** and **Thompson Sampling**, explained in [KP14] and depicted in Figure 3.5.

Instantiable Algorithms. We first focus on the two **UCB** and **Thompson Sampling** multi-armed bandits algorithms. In these two algorithms, the arm selection strategy always consist of selecting the higher score using the arg max function. Hence, for both of these algorithms and given the index M of the highest score, the MAB.SelectArm algorithm is programmed to return K selection bits $b_1, \ldots, b_K$ where b_M equals one. The only difference between these two algorithms is the computation of the score $v_{i,t}$. In **UCB**, the score is computed as $\hat{\mu}_i + \sqrt{\frac{2\ln(t)}{n_i}}$ where $\hat{\mu}_i = \frac{s_i}{n_i}$ is the empirical mean of the obtained rewards estimating the unknown reward probability μ_i. In contrast, the **Thompson Sampling** algorithm computes its score by sampling a random value from the (α, β)-Beta distribution, where α equals $s_i + 1$ and where β equals $n_i - s_i + 1$. Hence, only the MAB.ScoreArm algorithm differs in order to compute the appropriate score according to the desired strategy to use.

The ϵ**-greedy** algorithm requires more attention because of its dual arm selection strategy. The score of each arm is computed as the empirical mean of the obtained rewards $\hat{\mu}_i = \frac{s_i}{n_i}$, estimating the unknown reward probability μ_i. During an exploit phase, the arm selection intuitively selects the arm having the higher reward probability μ_i and hence the highest estimation $\hat{\mu}_i$. The exploration, in contrast, randomly choose an arm, hence ignoring scores. The decision to either explore or exploit at each time step $t \in [\![N]\!]$ is randomly chosen based on the probability ϵ. The ϵ probability has to be carefully chosen. Indeed, in case where ϵ is fixed and small then the arm selection strategy will exploit most of the time. Conversely, a fixed and high probability ϵ will lead to explore most of the time. An intermediate strategy used to set the probability ϵ is to decrease the value of ϵ over the time. To be instantiated in our MAB model, we have to deal with this dual arm selection strategy, namely the exploitation and the exploration strategies, with respect to the current probability ϵ. Observe that this probability is completely independent of the cumulative rewards s_i and the number of pulls n_i and hence can be made public without loss of security. To instantiate ϵ-greedy in our model, we only have to modify the behaviour of the MAB.ScoreArm algorithm: During the exploit, each arm computes and returns the empirical mean $\hat{\mu}_i$, whereas

during the exploration each data owner randomly chooses a score from an arbitrary but fixed distribution, say in $[\![K]\!]$. Indeed, since we are performing an arg max on the scores, when the scores are random then the best arm is chosen at random as well.

Non-instantiable Algorithm. For the sake of illustration, we focus on the Reinforcement Comparison [SB18] algorithm that cannot be instantiated in our MAB model. Briefly, this algorithm 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 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 our model.

3.2 Security Model for Secure Federated Multi-Armed Bandits

A Secure Federated Multi-Armed Bandits (SFMAB) protocol consists of $K+2$ parties: K data owners where each data owner maintains its own cumulative rewards s_i and number of pulls n_i as well as the reward function $pull_i$ (associated to the reward probability μ_i), a user $\mathcal{U}$ providing the budget N and expecting the total cumulative rewards s, and the *Controller* $\mathcal{C}$ in charge of the arm selection.

The protocol starts with a setup stage in which every entity obtains its secret and public keys, used later in the protocol. This setup phase is programmed as three distinct and independent key generation algorithms, one for each involved party: A key generation algorithm denoted $\mathcal{DO}$KeyGen for all the data owners, a key generation for the controller denoted $\mathcal{C}$KeyGen and a key generation algorithm for the user $\mathcal{U}$KeyGen. We sometimes refer to the key generation algorithms being executed by all the parties as the *setup phase*. To authenticate the public keys, we assume a Public-Key Infrastructure (PKI), where every public keys are implicitly stored.

Once the setup stage is executed, the arm selection protocol, called SelectArm, is executed for each time step $t \in [\![N]\!]$ where N is the budget between the K data owners and the controller $\mathcal{C}$. The SelectArm protocol starts with every data owner inputting its score $v_{i,t}$ obtained via the MAB.ScoreArm algorithm. The execution of the SelectArm ends with every data owner having a selection bit. Observe that from a security point-of-view, the fact that a *single* data owner (representing an arm) is selected at time step t is considered yet not explicit, since this property is more related to the correctness definition of a multi-armed. The uniqueness of the selected bandit is formally defined in our generic multi-armed bandits model in Definition 19, itself included in our security definition. Let us denote the selection bit obtained from the data owner $\mathcal{DO}_i$ at the time step t by $b_{i,t}$. Independently of the value of $b_{i,t}$, the data owner $\mathcal{DO}_i$ inputs the MAB.PullArm algorithm, along the cumulative rewards s_i and number of pulls n_i. The

output of the MAB.PullArm algorithm, consisting of the updated cumulative rewards s'_i and number of pulls n'_i, is modified if and only if the provided selection bit $b_{i,t}$ equals one. Remark that SelectArm is modelled as protocol to cover a large set of protocols.

Once all the budget has been spent, each data owner owns a cumulative rewards s_i. The last step of the SFMAB protocol execution consists of sending of the sum of all cumulative rewards to the final user $\mathcal{U}$, via the rewards sending protocol denoted SendRewards. In this protocol, all data owners are involved, in addition to the final user receiving the rewards, and the controller $\mathcal{C}$. SendRewards is also modelled as a protocol to cover a large set of secure federated multi-armed protocols.

Definition 20 (Secure Federated Multi-Armed Bandits, SFMAB)**.** Let Π be a secure federated multi-armed bandits protocol defined by the triplet of polynomial-time algorithm and protocols ($\mathcal{DO}$KeyGen, $\mathcal{C}$KeyGen, $\mathcal{U}$KeyGen, SelectArm, SendRewards) where:

- $\underline{\mathcal{DO}\mathsf{KeyGen}(1^\lambda) \rightarrow (sk_i, pk_i)}$: Given the unary representation of the security parameter, outputs the secret key sk_i and the public key pk_i of the data owner $\mathcal{DO}_i$.
- $\underline{\mathcal{C}\mathsf{KeyGen}(1^\lambda) \rightarrow (sk_\mathcal{C}, pk_\mathcal{C})}$: Given the unary representation of the security parameter, outputs the secret key $sk_\mathcal{C}$ and the public key $pk_\mathcal{C}$ of the controller $\mathcal{C}$.
- $\underline{\mathcal{U}\mathsf{KeyGen}(1^\lambda) \rightarrow (sk_\mathcal{U}, pk_\mathcal{U})}$: Given the unary representation of the security parameter, outputs the secret key $sk_\mathcal{U}$ and the public key $pk_\mathcal{U}$ of the user $\mathcal{U}$.
- $\underline{\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s_1, n_1, v_1), ..., \mathcal{DO}_K(sk_K, s_K, n_K, v_K), \mathcal{C}(sk_\mathcal{C})\rangle \rightarrow \mathcal{DO}_1(b_1), ..., \mathcal{DO}_K(b_K)}$: Given each data owner $\mathcal{DO}_i$ having the secret key sk_i, cumulative rewards s_i, number of pulls n_i and score v_i, given the controller $\mathcal{C}$ having the secret key $sk_\mathcal{C}$, the SelectArm protocol ends with each $\mathcal{DO}_i$ having the selection bit b_i. We stress that the returned selection bits should satisfy the two following constraints: First, $b_1 + \cdots + b_K$ equals 1, meaning that there is exactly one selection bit b_M at one, every other selection bits being at zero. Second, we must have $b_1, \ldots, b_K \leftarrow \mathsf{MAB.SelectArm}(\mathsf{MAB.BestArm}(v_1, \ldots, v_K), t)$ to ensure correctness of the multi-armed bandits algorithm.
- $\underline{\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s_1), ..., \mathcal{DO}_K(sk_K, s_K), \mathcal{C}(sk_\mathcal{C}), \mathcal{U}(sk_\mathcal{U})\rangle \rightarrow \mathcal{U}(s)}$: Given every $\mathcal{DO}_i$ having the cumulative rewards s_i, given the controller $\mathcal{C}$ having the secret key $sk_\mathcal{C}$ and the user $\mathcal{U}$ having the secret key $sk_\mathcal{U}$, the SendRewards protocol ends with the user $\mathcal{U}$ having the total cumulative rewards s which equals $s_1 + ... + s_n$.

Observe that SelectArm and SendRewards obtain as an input some data computed outside of the SFMAB specification. For instance the SelectArm protocol expects as an input the score v_i computed from the MAB.ScoreArm algorithm, as well as the cumulative rewards s_i and the number of pulls n_i maintained by the data owner. This behavior is desired since a SFMAB is only interesting when communications between the parties are involved *i.e.,* during the arm selection and the sending of the total cumulative rewards.

3.2.1 Correctness Definition

The correctness of a SFMAB scheme Π consists of returning the same total cumulative rewards s for the user $\mathcal{U}$, with respect to a multi-armed bandits algorithm MAB, assuming the same random coins for the arm selection and rewards. For clarity, we

MAB-Rewards(MAB, N, K)

1 : $(s_1, n_1), ..., (s_K, n_K) \leftarrow \mathsf{MAB.Init}(K)$
2 : **for** $t \in [\![N]\!]$
3 : $\quad \forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(s_i, n_i, t)$
4 : $\quad (b_1, \ldots, b_K) \leftarrow \mathsf{MAB.SelectArm}(\mathsf{MAB.BestArm}(v_{1,t}, \ldots, v_{K,t}), t)$
5 : $\quad \forall i \in [\![K]\!], s_i, n_i \leftarrow \mathsf{MAB.PullArm}(t, b_i, s_i, n_i)$
6 : **return** $s_1 + \cdots + s_K$

SFMAB-Rewards($\mathsf{MAB}, \Pi, N, K, \lambda$)

1 : $\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
2 : $(sk_1, pk_1), \ldots, (sk_K, pk_K) \leftarrow \Pi.\mathcal{DO}\mathsf{KeyGen}(1^\lambda)$
3 : $(sk_\mathcal{C}, pk_\mathcal{C}), (sk_\mathcal{U}, pk_\mathcal{U}) \leftarrow \Pi.\mathcal{C}\mathsf{KeyGen}/\Pi.\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
4 : **for** $t \in [\![N]\!]$
5 : $\quad \forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(t, s_i, n_i)$ ** Local score evaluation */*
6 : $\quad \Pi.\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s_1, n_1, v_{1,t}), ..., \mathcal{DO}_K(sk_K, s_K, v_K, v_{K,t}), \mathcal{C}(sk_\mathcal{C})\rangle$
$\qquad \rightarrow \mathcal{DO}_1(b_{1,t}), ..., \mathcal{DO}_K(b_{K,t})$
7 : $\quad \forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.PullArm}(t, b_{i,t}, s_i, n_i)$ ** Local arm pulling */*
8 : $\Pi.\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s_1), ..., \mathcal{DO}_K(sk_K, s_K), \mathcal{C}(sk_\mathcal{C}), \mathcal{U}(sk_\mathcal{U})\rangle \rightarrow \mathcal{U}(s)$
9 : **return** s

Figure 3.6: Execution of the standard multi-armed bandits algorithm (above) and its SFMAB version (below).

have depicted in Figure 3.6 the execution of a standard multi-armed bandits algorithm denoted MAB and its SFMAB version, both returning the total cumulative rewards s.

Definition 21 (p-correctness for SFMAB)**.** Let K be the number of arms and N the budget, and let $\mu_1, \ldots, \mu_K$ be the reward probabilities for all arms. Then, assuming a multi-armed bandits algorithm denoted MAB, and its SFMAB version denoted Π, then Π is said p-correct for some probability p if for every security parameter $\lambda \in \mathbb{N}$, the following probability holds:

$$\Pr\left[\mathsf{MAB\text{-}Rewards}(\mathsf{MAB}, N, K) = \mathsf{SFMAB\text{-}Rewards}(\mathsf{MAB}, \Pi, N, K, \lambda)\right] = p$$

3.2.2 Security Definition

As explained in the introduction of this chapter, we assume that all entities involved in the SFMAB protocol are *honest-but-curious*, in the sense that every party acts as expected, but tries to learn some information based on the received inputs. More importantly, we consider that each party does not collude with other parties. If a SFMAB requires to have a trusted setup (which is not the case with our construction) one may consider an honest trusted setup authority, acting only to generate and distribute keys. Since multi-armed bandits are mostly used in a setting where the server and the final user are publicly identified, we assume the two followings hypotheses: First, there is a Public-Key Infrastructure (PKI) authenticating public keys of the server and the final user. Second, we assume authenticated and confidential channels between every party.

$\mathsf{Exp}^{\mathsf{RealC}}_{\mathcal{A}}\ (\lambda, N, K)$

$\mu_1, \ldots, \mu_K \leftarrow \mathcal{A}(N, K)$
$\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
$(sk_1, pk_1), \ldots, (sk_K, pk_K), (sk_{\mathcal{C}}, pk_{\mathcal{C}}), (sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{DO}\mathsf{KeyGen}/\mathcal{C}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
for $t \in [\![N]\!]$
  $\forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(t, s_i, n_i)$
  $\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s_1, n_1, v_{1,t}), ..., \mathcal{DO}_K(sk_K, s_K, v_K, v_{K,t}), \mathcal{C}(sk_{\mathcal{C}})\rangle$
    $\rightarrow \mathcal{DO}_1(b_{1,t}), ..., \mathcal{DO}_K(b_{K,t})$
  $\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.PullArm}(t, b_{i,t}, s_i, n_i; \mu_i)$
$\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s_1), ..., \mathcal{DO}_K(sk_K, s_K), \mathcal{C}(sk_{\mathcal{C}}), \mathcal{U}(sk_{\mathcal{U}})\rangle \rightarrow \mathcal{U}(s)$
return $\mathcal{A}(sk_{\mathcal{C}}, pk_{\mathcal{U}}, pk_{\mathcal{C}}, pk_1, ..., pk_K, \mathsf{View}(\mathcal{C}))$

$\mathsf{Exp}^{\mathsf{IdealC}}_{\mathcal{A},\mathcal{S}}\ (\lambda, N, K)$ with a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$

$\mu_1, \ldots, \mu_K \leftarrow \mathcal{A}(N, K)$
$\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
$(sk_1, pk_1), \ldots, (sk_K, pk_K), (sk_{\mathcal{C}}, pk_{\mathcal{C}}), (sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{DO}\mathsf{KeyGen}/\mathcal{C}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
for $t \in [\![N]\!]$
  $(s'_1, n'_1, v'_{1,t}), ..., (s'_K, n'_K, v'_{K,t}) \leftarrow \mathcal{S}_0(t, K)$
  $\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s'_1, n'_1, v'_{1,t}), ..., \mathcal{DO}_K(sk_K, s'_K, n'_K, v'_{K,t}), \mathcal{C}(sk_{\mathcal{C}})\rangle$
    $\rightarrow \mathcal{DO}_1(b_{1,t}), ..., \mathcal{DO}_K(b_{K,t})$
$s'_1, ..., s'_K \leftarrow \mathcal{S}_1(K)$
$\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s'_1), ..., \mathcal{DO}_K(sk_K, s'_K), \mathcal{C}(sk_{\mathcal{C}}), \mathcal{U}(sk_{\mathcal{U}})\rangle$
  $\rightarrow \mathcal{U}(s'_1 + ... + s'_K)$
return $\mathcal{A}(sk_{\mathcal{C}}, pk_{\mathcal{U}}, pk_{\mathcal{C}}, pk_1, ..., pk_K, \mathsf{View}(\mathcal{C}))$

Figure 3.7: Security games for a real execution of the protocol and the simulation of the protocol denoted respectively $\mathsf{Exp}^{\mathsf{RealC}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealC}}_{\mathcal{A}}$ for the corruption case of $\mathcal{C}$.

Our second hypothesis remains practical since the server is publicly identified, allowing one to establish secure and authenticated channel using, for instance, TLS [Res18].

Security Properties. Based on the aforementioned hypotheses, we present the expected security properties of a SFMAB scheme. In this work, we have chosen to follow a *simulation*-based security formalism, in which an adversary is asked to distinguish between two security games called respectively the *real* world and the *ideal* world: The real world corresponds to a real execution of the protocol, whereas the ideal world corresponds to a protocol execution in which a simulator denoted $\mathcal{S}$ simulates parts of the protocol, using limited information. This approach is more commonly known as the *real-ideal* paradigm. In this work, our security definition is divided into three distinct security experiments following the real-ideal paradigm, one for each party involved in the protocol execution.

The simulation security for the controller $\mathcal{C}$, following the real-ideal paradigm, considers the real experiment RealC consisting of the real protocol execution, and the ideal experiment IdealC consisting of the ideal protocol execution, both depicted in Figure 3.7. The real world modelled in the $\mathsf{Exp}^{\mathsf{RealD}}_{\mathcal{A}}$ experiment starts by generating the arm states as well as generating the key pair for every party in the protocol. Then, the arm selection

protocol $\mathsf{SelectArm}$ is executed N times. Once the budget has been spent, the experiment executes the rewards sending protocol $\mathsf{SendRewards}$ ending with the final user $\mathcal{U}$ having the total cumulative rewards s. At the end of the experiment, the adversary is inputted with the public key of every party, as well as the secret key of the controller $\mathcal{C}$ along his view, denoted $\mathsf{View}(\mathcal{C})$. Note that the exact information contained in the view of $\mathcal{C}$ depends on the concrete protocol and hence cannot be inferred for the moment. The ideal world depicted in the $\mathsf{Exp}^{\mathsf{IdealC}}_{\mathcal{A}}$ experiment, in contrast to the RealC experiment, is equipped of the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$. Compared to the real experiment, the arm scores generation is omitted at each time step $t \in [\![N]\!]$, the real scores $v_{i,t}$ for $i \in [\![K]\!]$ being replaced with values obtained by running our first simulator $\mathcal{S}_0$, inputted with the number of arms K and the current time step t. Indeed, the simulator $\mathcal{S}_0$ is asked to output a list of tuples $(s'_i, n'_i, v'_{i,t})$ where s'_i is a random cumulative rewards, n'_i is a random number of pulls and $v'_{i,t}$ is a random score. The design of our simulator is motivated since it is asked to produce inputs for each data owner having only access to the *number of arms* K and the current time step $t \in [\![N]\!]$. If the adversary $\mathcal{A}$, corrupting the controller $\mathcal{C}$ in this experiment, cannot distinguish between the real world and the ideal world, then we can conclude that $\mathcal{C}$ learns nothing about the scores, otherwise is able to distinguish by comparing the scores distribution. The selection bits outputted by the data owners are honestly computed over the provided random scores, trivially leading to a random arm selection. A protocol exposing the selection bits to the adversary would trivially break the simulation by observing a different distribution of the selected arm. Our second simulator $\mathcal{S}_1$ taking place in our ideal world during the sending of the rewards with the user $\mathcal{U}$, is inputted with the number of arms K. It outputs K random cumulative rewards s'_i that is inputted to the data owner $\mathcal{DO}_i$ instead of the real cumulative rewards s_i. Again, if the adversary cannot distinguish, then we expect that the adversary $\mathcal{A}$ corrupting $\mathcal{C}$ does not learn any information about both each cumulative rewards but also the total cumulative rewards s.

Definition 22 (Simulation security against $\mathcal{C}$)**.** A SFMAB protocol Π defined by the tuple $(\mathcal{DO}\mathsf{KeyGen}, \mathcal{C}\mathsf{KeyGen}, \mathcal{U}\mathsf{KeyGen}, \mathsf{SelectArm}, \mathsf{SendRewards})$ is said SimC-secure with respect to the budget N and the number of arms K if for every security parameter λ and every adversary $\mathcal{A}$, there exist a simulator $\mathcal{S}$ such that:

$$\mathsf{Adv}^{\mathsf{SimC}}_{\mathcal{A},\Pi} = \left|\Pr\left[\mathsf{Exp}^{\mathsf{RealC}}_{\mathcal{A},\Pi}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{IdealC}}_{\mathcal{A},\mathcal{S},\Pi}(1^\lambda, N, K) \to 1\right]\right| \leq \mathsf{negl}(\lambda)$$

The simulation security for the user $\mathcal{U}$ follows the real-ideal paradigm with the ideal experiment IdealU and the real experiment RealU. Since the real experiment remains unchanged compared to the previous definition, we omit the description. During the arm selection protocol, namely $\mathsf{SelectArm}$, the user $\mathcal{U}$ is not involved. However, it is involved during the rewards sending protocol, namely $\mathsf{SendRewards}$. At the end of the protocol, $\mathcal{U}$ is expected to obtain the total cumulative rewards s computed as the sum of the cumulative rewards s_i for every $i \in [\![K]\!]$. To complete a successful simulation, we have to care about the honest execution of the multi-armed bandits algorithm. Indeed, the random sampling of a total cumulative rewards s' instead of the total cumulative

rewards s necessarily modifies the view of the adversary, allowing it to distinguish. This stands in contrast with the simulation for the controller $\mathcal{C}$ whose honest execution of the multi-armed bandits algorithm is not mandatory. From a security standpoint, we expect the user $\mathcal{U}$ not to learn any information on the score of an arm, on the cumulative rewards s_i of a given data owner $\mathcal{DO}_i$, but also on the selected arm, which can be inferred via the selection bits.

Once again, our ideal experiment IdealU is equipped of the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$. Our first simulator $\mathcal{S}_0$ is focused on the simulation of values inputted to the $\mathsf{SelectArm}$ protocol. Our simulator is inputted with a very limited information, namely, the current time step t and the number of arms K. The output of $\mathcal{S}_0$ consists of K tuples of the form $(s'_i, n'_i, v'_{i,t})$ where s'_i is a random cumulative rewards, n'_i is a random number of pulls and $v'_{i,t}$ is a random score. Note that the correctness of the multi-armed bandits algorithms is ensured by the challenger running the experiment. The arm selection protocol is inputted with completely random values, leading to a random arm selection. Similarly to the IdealC experiment, if no adversary can distinguish between the arm selection inputted with valid or random values, based on the $\mathcal{U}$'s view, then it ensures that $\mathcal{U}$ does not learn any information on the cumulative rewards, on the number of pulls but also on the selected arm. The second simulator $\mathcal{S}_1$ is inputted only with the number of arms and the total cumulative rewards s computed as the sum of the cumulative rewards, as well as the number of arms K. The simulator $\mathcal{S}_1$ is expected to return K cumulative rewards s'_i whose sum equals the total cumulative rewards s. Clearly, if the cumulative rewards s_i can be replaced by random ones without affecting the view of $\mathcal{U}$, then we can conclude that every s_i is not in the view of $\mathcal{U}$.

Definition 23 (Simulation security against $\mathcal{U}$)**.** A SFMAB protocol Π defined by the tuple $(\mathcal{DO}\mathsf{KeyGen}, \mathcal{C}\mathsf{KeyGen}, \mathcal{U}\mathsf{KeyGen}, \mathsf{SelectArm}, \mathsf{SendRewards})$ is said SimU-secure with respect to the budget N and the number of arms K if for every security parameter λ and every adversary $\mathcal{A}$, there exist a simulator $\mathcal{S}$ such that:

$$\mathsf{Adv}^{\mathsf{SimU}}_{\mathcal{A},\Pi} = \left|\Pr\left[\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A},\Pi}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S},\Pi}(1^\lambda, N, K) \to 1\right]\right| \leq \mathsf{negl}(\lambda)$$

The simulation security for a (single) data owner $\mathcal{DO}_i$ brings a different security. Indeed, a data owner $\mathcal{DO}_i$ has access to the cumulative rewards s_i, the number of pulls n_i but also the selection bit $b_{i,t}$ at each time step. Hence, we have to ensure the correctness of the arm selection at any time, otherwise is able to distinguish between a correct execution of the multi-armed bandits and an altered one. To correctly do the simulation, the security experiment starts with the adversary providing the corrupted data owner index j. In contrast to all other security definitions, we input the simulator $\mathcal{S}_0$ with the current time step t, the number of arms K but also two additional information: The score $v_{j,t}$ of the corrupted data owner $\mathcal{DO}_j$ and a bit identifying if the best arm index M, evaluated on the real scores, equals the corrupted data owner index j. In other words, the simulator $\mathcal{S}_0$ knows if the corrupted data owner $\mathcal{DO}_j$ owns the best score or at least if it has to be chosen. Indeed, in case where the corrupted data owner has to be chosen by the arm selection protocol, then the simulator $\mathcal{S}_0$ has to output

$\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}(\lambda, N, K)$

$\mu_1, \ldots, \mu_K \leftarrow \mathcal{A}(N, K)$ /* *The reward probabilities are implicitly used.* */
$\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
$(sk_1, pk_1), \ldots, (sk_K, pk_K), (sk_{\mathcal{C}}, pk_{\mathcal{C}}), (sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{DO}\mathsf{KeyGen}/\mathcal{C}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
for $t \in [\![N]\!]$
  $\forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(t, s_i, n_i)$
  $\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s_1, n_1, v_{1,t}), ..., \mathcal{DO}_K(sk_K, s_K, v_K, v_{K,t}), \mathcal{C}(sk_{\mathcal{C}})\rangle$
    $\rightarrow \mathcal{DO}_1(b_{1,t}), ..., \mathcal{DO}_K(b_{K,t})$
  $\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.PullArm}(t, b_{i,t}, s_i, n_i; \mu_i)$
$\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s_1), ..., \mathcal{DO}_K(sk_K, s_K), \mathcal{C}(sk_{\mathcal{C}}), \mathcal{U}(sk_{\mathcal{U}})\rangle \rightarrow \mathcal{U}(s)$
return $\mathcal{A}(sk_{\mathcal{U}}, pk_{\mathcal{U}}, pk_{\mathcal{C}}, pk_1, ..., pk_K, \mathsf{View}(\mathcal{U}))$

$\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}(\lambda, N, K)$ with a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$

$\mu_1, \ldots, \mu_K \leftarrow \mathcal{A}(N, K)$
$\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
$(sk_1, pk_1), \ldots, (sk_K, pk_K), (sk_{\mathcal{C}}, pk_{\mathcal{C}}), (sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{DO}\mathsf{KeyGen}/\mathcal{C}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
for $t \in [\![N]\!]$
  $\forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(t, s_i, n_i)$
  $(s'_1, n'_1, v'_{1,t}), ..., (s'_K, n'_K, v'_{K,t}) \leftarrow \mathcal{S}_0(t, K)$
  $\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s'_1, n'_1, v'_{1,t}; \mu_1), ..., \mathcal{DO}_K(sk_K, s'_K, n'_K, v'_{K,t}; \mu_K), \mathcal{C}(sk_{\mathcal{C}})\rangle$
  $M \leftarrow \arg\max(v_{1,t}, ..., v_{K,t})$
  $(s_M, n_M) \leftarrow \mathsf{MAB.PullArm}(t, 1, s_M, n_M; \mu_M)$
$s'_1, ..., s'_K \leftarrow \mathcal{S}_1(s_1 + ... + s_K, K)$
$\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s'_1), ..., \mathcal{DO}_K(sk_K, s'_K), \mathcal{C}(sk_{\mathcal{C}}), \mathcal{U}(sk_{\mathcal{U}})\rangle \rightarrow \mathcal{U}(s)$
return $\mathcal{A}(sk_{\mathcal{U}}, pk_{\mathcal{U}}, pk_{\mathcal{C}}, pk_1, ..., pk_K, \mathsf{View}(\mathcal{U}))$

Figure 3.8: Security games for a real execution of the protocol and the simulation of the protocol denoted respectively $\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A}}$ against $\mathcal{U}$.

a list of scores such that $\mathcal{DO}_j$ receives a positive selection bit $b_{j,t}$. Otherwise, $\mathcal{DO}_j$ has to receive a negative selection bit. Note that in this case, the simulator $\mathcal{S}_0$ does not have any information on the arm selection that has to be chosen. Hence, the arm selection is inputted with random scores (for all the data owners except for the corrupted one) leading to a random arm selection. We stress that the selection bits obtained by uncorrupted data owners are ignored, only the M-th data owner is pulled thanks to the challenger. Without having access to the real total cumulative rewards and individual cumulative rewards s_i, the second simulator $\mathcal{S}_1$ is asked to return the cumulative rewards s'_i for every honest data owner (*i.e.,* for all data owners except the corrupted data owner denoted $\mathcal{DO}_j$).

Definition 24 (Simulation security against $\mathcal{DO}_i$)**.** A SFMAB protocol Π defined by the tuple $(\mathcal{DO}\mathsf{KeyGen}, \mathcal{C}\mathsf{KeyGen}, \mathcal{U}\mathsf{KeyGen}, \mathsf{SelectArm}, \mathsf{SendRewards})$ is said to be SimDO-secure with respect to the budget N and the number of arms K if for every security

$\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}(\lambda, N, K)$

$\mu_1, \ldots, \mu_K \leftarrow \mathcal{A}(N, K)$ /* *The reward probabilities are implicitly used.* */
$\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
$(sk_1, pk_1), \ldots, (sk_K, pk_K), (sk_{\mathcal{C}}, pk_{\mathcal{C}}), (sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{DO}\mathsf{KeyGen}/\mathcal{C}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
$j \leftarrow \mathcal{A}(pk_{\mathcal{U}}, pk_{\mathcal{C}}, pk_1, ..., pk_K)$
for $t \in [\![N]\!]$
  $\forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(t, s_i, n_i)$
  $\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, s_1, n_1, v_{1,t}), ..., \mathcal{DO}_K(sk_K, s_K, v_K, v_{K,t}), \mathcal{C}(sk_{\mathcal{C}})\rangle$
    $\rightarrow \mathcal{DO}_1(b_{1,t}), ..., \mathcal{DO}_K(b_{K,t})$
  $\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.PullArm}(t, b_{i,t}, s_i, n_i)$
$\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s_1), ..., \mathcal{DO}_K(sk_K, s_K), \mathcal{C}(sk_{\mathcal{C}}), \mathcal{U}(sk_{\mathcal{U}})\rangle \rightarrow \mathcal{U}(s)$
return $\mathcal{A}(sk_j, \mathsf{View}(\mathcal{DO}_j))$

Security game $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}(\lambda, N, K)$ with a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$

$\mu_1, \ldots, \mu_K \leftarrow \mathcal{A}(N, K)$ /* *The reward probabilities are implicitly used.* */
$\forall i \in [\![K]\!], (s_i, n_i) \leftarrow \mathsf{MAB.Init}()$
$(sk_1, pk_1), \ldots, (sk_K, pk_K), (sk_{\mathcal{C}}, pk_{\mathcal{C}}), (sk_{\mathcal{U}}, pk_{\mathcal{U}}) \leftarrow \mathcal{DO}\mathsf{KeyGen}/\mathcal{C}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda)$
$j \leftarrow \mathcal{A}(pk_{\mathcal{U}}, pk_{\mathcal{D}}, pk_{\mathcal{P}}, pk_1, ..., pk_K)$
for $t \in [\![N]\!]$
  $\forall i \in [\![K]\!], v_{i,t} \leftarrow \mathsf{MAB.ScoreArm}(t, s_i, n_i)$
  $M \leftarrow \arg\max(v_{1,t}, ..., v_{K,t})$
  $\{(s'_i, n'_i, v'_{i,t})\}_{i \in [\![K]\!], i \neq j} \leftarrow \mathcal{S}_0(t, K, M \stackrel{?}{=} j, v_{j,t})$
  $\forall i \in [\![K]\!], i \neq j; V'_{i,t} \leftarrow (s'_i, n'_i, v'_{i,t})$
  $V_{j,t} \leftarrow (s_j, n_j, v_{j,t})$
  $\mathsf{SelectArm}\langle \mathcal{DO}_1(sk_1, V'_{1,t}), ..., \mathcal{DO}_j(sk_j, V_{j,t}), ..., \mathcal{DO}_K(sk_K, V'_{K,t}), \mathcal{C}(sk_{\mathcal{C}})\rangle$
    $\rightarrow \mathcal{DO}_1(b_{1,t}), ..., \mathcal{DO}_1(b_{j,t}), ..., \mathcal{DO}_K(b_{K,t})$
  $(s_M, n_M) \leftarrow \mathsf{MAB.PullArm}(t, 1, s_M, n_M; \mu_M)$
$s'_1, ..., s'_{j-1}, s'_{j+1}, ..., s'_K \leftarrow \mathcal{S}_1(K, j)$
$\mathsf{SendRewards}\langle \mathcal{DO}_1(sk_1, s'_1), ..., \mathcal{DO}_j(sk_j, s_j), ..., \mathcal{DO}_K(sk_K, s'_K), \mathcal{C}(sk_{\mathcal{C}}), \mathcal{U}(sk_{\mathcal{U}})\rangle$
  $\rightarrow \mathcal{U}(s'_1 + ... + s_j + ... + s'_K)$
return $\mathcal{A}(sk_j, \mathsf{View}(\mathcal{DO}_j))$

Figure 3.9: Security games for a real execution of the protocol and the simulation of the protocol denoted respectively $\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A}}$ against an individual data owner.

parameter λ and every adversary $\mathcal{A}$, there exist a simulator $\mathcal{S}$ such that:

$$\mathsf{Adv}^{\mathsf{SimDO}}_{\mathcal{A},\Pi} = \left| \Pr\left[\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A},\Pi}(1^\lambda, N, K) \rightarrow 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S},\Pi}(1^\lambda, N, K) \rightarrow 1\right] \right| \leq \mathsf{negl}(\lambda)$$

3.3 Tango: A Secure Federated Bandits Protocol

We propose Tango, our construction of SFMAB scheme relying on $K+3$ parties. First, we have K data owners in charge of maintaining their cumulative rewards s_i and number of pulls n_i. We have also the user $\mathcal{U}$, involved at the beginning to provide the budget N

and at the end to obtain the total cumulative rewards s. Finally, we have the controller $\mathcal{C}$, that is splitted here into *two* distinct parties, namely the proxy node $\mathcal{P}$ and the reflector node $\mathcal{R}$. Before introducing our construction, since we are splitting $\mathcal{C}$ into two distinct parties, some clarifications are needed about the security model we consider. As already assumed during the security definition statement (see Section 3.2.2), we assume that $\mathcal{P}$ and $\mathcal{R}$ are honest-but-curious. Another assumption we do, $\mathcal{P}$ and $\mathcal{R}$ should not collude. We stress that this assumption is a standard and common hypothesis in multi-party computations. For the sake of comprehension, let us first introduce an overview of our construction, whose design follows the blueprint of Samba [CLMS22].

Overview of Tango. The best arm identification consists of computing the index of the arm having the highest score. This is achieved in Tango using the key mechanism that we call the *homomorphic best arm identification*. In a nutshell, the best arm identification is performed directly over encrypted approximate scores, in the encrypted domain thanks to the fully homomorphic encryption (FHE) scheme. Note that the scores inputting the arg max function follow the order of the data owner index, which preserves the equality of the arg max when two or more scores are equal. Once the best arm index has been obtained, say M, it is still encrypted under the FHE scheme and hence unknown to the proxy node. Given this encrypted index M, the proxy node $\mathcal{P}$ computes every selection bit b_i for each data owner $\mathcal{DO}_i$, by testing the equality function between the current data owner index i and the best arm index M. As a result, the proxy node $\mathcal{P}$ obtains K encrypted selection bits, where only the encrypted bit b_M equals one (since only the M-th data owner the index M). These encrypted selection bits, along with K secret-keys of a secret-key encryption (SKE) scheme encrypted under a public-key encryption (PKE) scheme, received at each time step from every data owner, are permuted thanks to a permutation p_t generated by $\mathcal{P}$ at each time step $t \in [\![N]\!]$. These permuted encrypted selection bits and permuted encrypted secret-keys are shared with the reflector node $\mathcal{R}$, having the private decryption key for both the PKE and FHE schemes. After the decryption of all the received ciphertexts, the reflector node $\mathcal{R}$ obtains K permuted secret-keys and K permuted selection bits. It encrypts every i-th selection bit with i-th received secret-key of the SKE scheme. The K resulting ciphertexts, encrypting *permuted* selection bits, are sent back to the proxy node $\mathcal{P}$ which inverts the permutation before forwarding the encrypted selection bit to the appropriate data owner, having the secret-key to decrypt the selection bit.

We recall that during the protocol execution, a data owner $\mathcal{DO}_i$ maintains a cumulative rewards s_i that needs to be aggregated and sent to the final user during the SendRewards protocol. During this reward sending protocol, every $\mathcal{DO}_i$ encrypts its cumulative rewards s_i using an additively homomorphic encryption scheme AHE whose private decryption key is owned by the user $\mathcal{U}$. The resulting ciphertext that we denote as c_i^s, is then shared with the proxy node $\mathcal{P}$, that does the homomorphic sum resulting into the encryption of the total cumulative rewards s, later sent to the user $\mathcal{U}$ recovering s. To support the failure of a data owner $\mathcal{DO}_i$, at every time step t, $\mathcal{DO}_i$ encrypts its cumulative rewards s_i using the additively homomorphic encryption scheme and shares the resulting ciphertext c_i^s to the proxy node $\mathcal{P}$ which registers the pair (i, c_i^s), overwriting the entry $(i, \cdot)$ if such an entry exists.

$\mathcal{DO}\mathsf{KeyGen}\ (1^\lambda)$

1 : $sk_i \leftarrow \emptyset$
2 : $pk_i \leftarrow \emptyset$
3 : **return** (sk_i, pk_i)

$\mathcal{P}\mathsf{KeyGen}\ (1^\lambda)$

1 : $sk_\mathcal{P} \leftarrow \emptyset$
2 : $pk_\mathcal{P} \leftarrow \emptyset$
3 : **return** $(sk_\mathcal{P}, pk_\mathcal{P})$

$\mathcal{R}\mathsf{KeyGen}\ (1^\lambda)$

1 : $(sk_\mathcal{R}^{\mathsf{FHE}}, pk_\mathcal{R}^{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(\lambda)$
2 : $(sk_\mathcal{R}^{\mathsf{PKE}}, pk_\mathcal{R}^{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(\lambda)$
3 : $sk_\mathcal{R} \leftarrow (sk_\mathcal{R}^{\mathsf{PKE}}, sk_\mathcal{R}^{\mathsf{FHE}})$
4 : $pk_\mathcal{R} \leftarrow (pk_\mathcal{R}^{\mathsf{PKE}}, pk_\mathcal{R}^{\mathsf{FHE}})$
5 : **return** $(sk_\mathcal{R}, pk_\mathcal{R})$

$\mathcal{U}\mathsf{KeyGen}\ (1^\lambda)$

1 : $(sk_\mathcal{U}^{\mathsf{AHE}}, pk_\mathcal{U}^{\mathsf{AHE}}) \leftarrow \mathsf{AHE.KeyGen}(\lambda)$
2 : $sk_\mathcal{U} \leftarrow sk_\mathcal{U}^{\mathsf{AHE}}$
3 : $pk_\mathcal{U} \leftarrow pk_\mathcal{U}^{\mathsf{AHE}}$
4 : **return** $(sk_\mathcal{U}, pk_\mathcal{U})$

Figure 3.10: Presentation of the setup of Tango

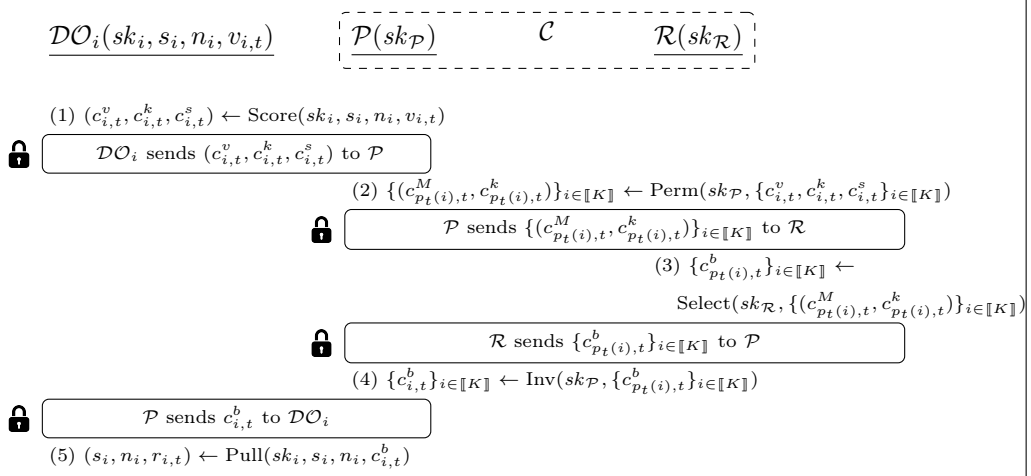
Setup of Tango (Figure 3.10)

In Tango the setup, being depicted in Figure 3.10, is represented by four independent key generation algorithms, one for each party involved in the protocol. In Tango, a data owner $\mathcal{DO}_i$ as well as the proxy node $\mathcal{P}$ are keyless. Hence, the key generation for each data owner $\mathcal{DO}_i$ as well as the proxy node $\mathcal{P}$, returning an empty private and public keys. During the protocol execution, the reflector node $\mathcal{R}$ and the user $\mathcal{U}$ have to decrypt ciphertexts, hence are required to generate one or more encryption key pairs. On a hand, the reflector node $\mathcal{R}$ needs to decrypt the encrypted permuted selection bits $b_{p_t(i),t}$, that are computed homomorphically using a fully homomorphic encryption scheme. On another hand, it has to decrypt a permuted list of encrypted secret-keys, encrypted by the data owners using the encryption public-key of $\mathcal{R}$. Hence, during the setup, the reflector node $\mathcal{R}$ generates two encryption key pairs, one for the fully homomorphic encryption scheme FHE and another one for the public-key encryption scheme PKE. On its side, the user $\mathcal{U}$ receives at the end of the SendRewards protocol a ciphertext encrypting the total cumulative rewards s, computed as the sum of encrypted cumulative rewards s_i, thanks to the additively homomorphic encryption scheme AHE. Hence, the user $\mathcal{U}$ generates a key pair for the AHE scheme. All public keys are assumed to be registered in the PKI.

Arm Selection Protocol of Tango (Figure 3.11)

The arm selection protocol, called SelectArm, starts with every data owner $\mathcal{DO}_i$ being inputted at the time step t with their respective scores $v_{i,t}$ as well as the cumulative rewards s_i and the number of pulls n_i. To allow the proxy node $\mathcal{P}$ to homomorphically evaluate the arg max function, each $\mathcal{DO}_i$ encrypts its score $v_{i,t}$ using the $\mathcal{R}$'s encryption key of the FHE scheme. $\mathcal{DO}_i$ also generates a secret-key $k_{i,t}$ that is encrypted using the $\mathcal{R}$'s encryption public key of the PKE scheme. Finally, to handle the case of failure, each $\mathcal{DO}_i$ encrypts its cumulative rewards s_i using the $\mathcal{U}$'s encryption public key of the AHE scheme. These three resulting ciphertexts respectively denoted $c_{i,t}^v$, $c_{i,t}^k$ and $c_{i,t}^s$ are sent to the proxy node $\mathcal{P}$.

Description of the SelectArm protocol at time $t \in [\![N]\!]$:



$\mathcal{DO}_i$ Score$(sk_i, s_i, n_i, v_{i,t})$

$k_{i,t} \leftarrow \mathsf{SKE.KeyGen}(\lambda)$
$c^v_{i,t} \leftarrow \mathsf{FHE.Enc}(pk^{\mathsf{FHE}}_\mathcal{R}, v_{i,t})$
$c^k_{i,t} \leftarrow \mathsf{PKE.Enc}(pk^{\mathsf{PKE}}_\mathcal{R}, k_{i,t})$
$c^s_{i,t} \leftarrow \mathsf{AHE.Enc}(pk^{\mathsf{AHE}}_\mathcal{U}, s_i)$
return $(c^v_{i,t}, c^k_{i,t}, c^s_{i,t})$

$\mathcal{P}$ Perm$(sk_\mathcal{P}, \{(c^v_{i,t}, c^k_{i,t}, c^s_{i,t})\}_{i\in[\![K]\!]})$

$\forall i \in [\![K]\!]$, Store $(i, c^s_{i,t})$
$p_t \leftarrow \mathsf{PRP.Gen}(K)$
$c^M_t \leftarrow \mathsf{FHE.Eval}(\arg\max, c^v_{1,t}, ..., c^v_{K,t})$
$\forall i \in [\![K]\!], c^M_{i,t} \leftarrow \mathsf{FHE.Eval}(\mathsf{Equal}, c^M_t, i)$ */* Ciphertext-Plaintext equality */*
$\{c^M_{p_t(i),t}\}_{i\in[\![K]\!]} \leftarrow \mathsf{PRP.Perm}(\{c^M_{i,t}\}_{i\in[\![K]\!]})$
$\{c^k_{p_t(i),t}\}_{i\in[\![K]\!]} \leftarrow \mathsf{PRP.Perm}(\{c^k_{i,t}\}_{i\in[\![K]\!]})$
return $\{(c^M_{p_t(i),t}, c^k_{p_t(i),t})\}_{i\in[\![K]\!]}$

$\mathcal{R}$ Select$(sk_\mathcal{R}, \{(c^M_{p_t(i),t}, c^k_{p_t(i),t})\}_{i\in[\![K]\!]})$

$\forall i \in [\![K]\!], k_{p_t(i),t} \leftarrow \mathsf{PKE.Dec}(sk^{\mathsf{PKE}}_\mathcal{R}, c^k_{p_t(i),t})$
$\forall i \in [\![K]\!], b_{p_t(i),t} \leftarrow \mathsf{FHE.Dec}(sk^{\mathsf{FHE}}_\mathcal{R}, c^M_{p_t(i),t})$
$\forall i \in [\![K]\!]; c^b_{p_t(i),t} \leftarrow \mathsf{SKE.Enc}(k_{p_t(i),t}, b_{p_t(i)})$
return $\{c^b_{p_t(i),t}\}_{i\in[\![K]\!]}$

$\mathcal{P}$ Inv$(sk_\mathcal{P}, \{c^b_{p_t(i),t}\}_{i\in[\![K]\!]})$

$\{c^b_{i,t}\}_{i\in[\![K]\!]} \leftarrow \mathsf{PRP.Inv}(\{c^b_{p_t(i),t}\}_{i\in[\![K]\!]})$
return $c^b_{A,t}, \ldots, c^b_{K,t}$

$\mathcal{DO}_i$ Pull$(sk_i, s_i, n_i, c^b_{i,t})$

$b_{i,t} \leftarrow \mathsf{SKE.Dec}(k_{i,t}, c^b_{i,t})$
return $b_{i,t}$

Figure 3.11: Presentation of the SelectArm protocol for TANGO. The padlock 🔓 means that an authenticated secure channel is used to communicate.

Once all the ciphertexts are received, thanks to the homomorphic property of the FHE scheme, $\mathcal{P}$ evaluates the arg max function over the encrypted scores, resulting into the index M, still encrypted under the $\mathcal{R}$'s encryption public key of the FHE scheme. The proxy node $\mathcal{P}$ converts this best arm index into K selection bits $b_{i,t}$, obtained via the homomorphic equality testing between the index i of a data owner $\mathcal{DO}_i$ and the encrypted best arm index M. For clarity, let us denote the encrypted selection bit for $\mathcal{DO}_i$ at a time step $t \in [\![N]\!]$ by $c^M_{i,t}$ and its encrypted secret-key by $c^k_{i,t}$. At this point, $\mathcal{P}$ generates a pseudo-random permutation p_t using the PRP scheme and permutes the encrypted selection bits and the encrypted secret-keys, resulting into the ciphertexts $c^M_{p_t(i),t}$ and $c^k_{p_t(i),t}$ for every $i \in [\![K]\!]$ at some time step $t \in [\![N]\!]$. These permuted ciphertexts are then shared with the reflector node $\mathcal{R}$ having the private decryption key for both the PKE and FHE schemes. After the decryption of these ciphertexts, $\mathcal{R}$ obtains K secret-keys $k_{p_t(i),t}$ of the SKE scheme and K selection bits $b_{p_t(i),t}$ whose only $p_t(M)$ equals one. Each selection bit $b_{p_t(i),t}$ is encrypted using the secret-key $k_{p_t(i),t}$ resulting into the SKE ciphertext denoted $c^b_{p_t(i),t}$.

Observe that $\mathcal{R}$ has now K ciphertexts, each one encrypting a selection bit. These encrypted selection bits, this time under the SKE scheme, are sent back to the proxy node $\mathcal{P}$, which does not have any key to decrypt the selection bits. Note that the secret-key being permuted by the proxy node are encrypted using the public-key encryption scheme PKE and hence prevents the proxy node to learn any information from these secret-keys, thanks to the IND-CPA security of the PKE scheme. Knowing the permutation p_t, the proxy node $\mathcal{P}$ can invert the permutation of the permuted list of encrypted selection bits $c^b_{p_t(1),t}, \ldots, c^b_{p_t(K),t}$ using the permutation inverse p_t^{-1}, resulting into the list $c^b_{1,t}, \ldots, c^b_{K,t}$. Hence, each encrypted selection bit $c^b_{i,t}$ is sent to i-th data owner $\mathcal{DO}_i$, having the secret-key $k_{i,t}$ of the SKE scheme to decrypt the ciphertext. Recall that the arm selection protocol ends with every data owner $\mathcal{DO}_i$ having a selection bit $b_{i,t}$, later used to pull itself using the MAB.PullArm algorithm. Hence, SelectArm ends with the return of the selection bit $b_{i,t}$.

We have omitted the role of the ciphertext $c^s_{i,t}$ encrypting the cumulative rewards s_i, transmitted by every data owner $\mathcal{DO}_i$ at each time step t to the proxy node $\mathcal{P}$. This ciphertext is not relevant for the arm selection but is useful to provide resistance against failure of a data owner: Suppose that a data owner $\mathcal{DO}_i$ becomes suddenly offline at some time $t \in [\![N]\!]$. Since a data owner does not hold any secret decryption key (except the one of the SKE scheme that is generated at each time step), the protocol is able to continue the arm selection without the offline data owner. However, if this data owner has been pulled, it is interesting to remember how much rewards it has produced. This is exactly for this reason that the encryption of s_i is sent at each time step to the proxy node $\mathcal{P}$, storing the pair $(i, c^s_{i,t})$. Later during the rewards sending protocol, if the data owner becomes offline, the saved cumulative rewards s_i. This prevents the loss of all the cumulative rewards collected by the data owner, even if the data owner remains offline until the end of the secure federated multi-armed bandits protocol execution.

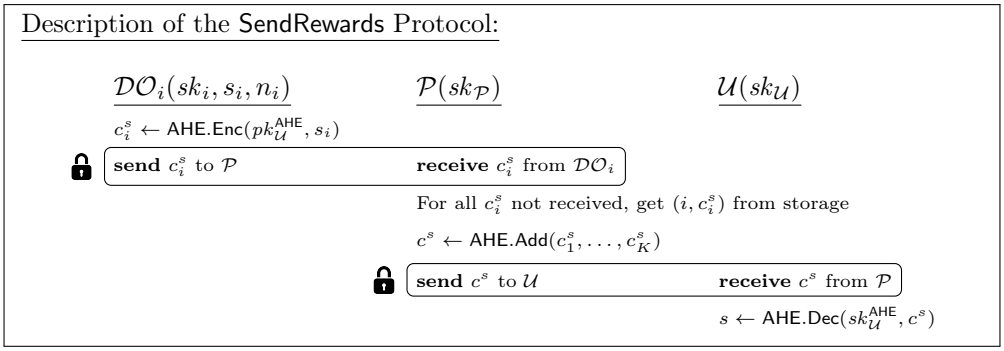


Figure 3.12: Presentation of the SendRewards protocol for TANGO. The padlock means that an authenticated secure channel is used to communicate.

Total Cumulative Rewards Sending Protocol of TANGO (Figure 3.12)

At the end of the protocol, the user $\mathcal{U}$ expects to obtain the total cumulative rewards s corresponding to the sum of the cumulative rewards s_i, owned by the data owners. To perform the secure aggregation of the cumulative rewards, we rely on the additively homomorphic property of the AHE scheme. The SendRewards protocol starts by each data owner inputted with its cumulative rewards s_i, encrypting s_i using the $\mathcal{U}$'s encryption public-key of the AHE scheme. The resulting ciphertext denoted c_i^s is sent to the proxy node $\mathcal{P}$, performing the secure aggregation using the additively homomorphic property of the AHE scheme. The encrypted total cumulative rewards s, denoted as the ciphertext c^s is shared with the user $\mathcal{U}$, owning the private decryption key and hence recovers the total cumulative rewards s.

3.3.1 Correctness of TANGO

To prove the correctness of TANGO, we have to show that all the cryptographic operations do not affect the arm selection as well as the computation of the total cumulative rewards. To prove this statement, we will proceed to a sequence of modifications starting from TANGO, in which all the cryptographic operations are removed one-by-one. To model potential errors due to cryptography, we explicitly denote by ϵ^{FHE} the probability of error of the fully homomorphic encryption scheme FHE during the arg max evaluation, and ϵ^{AHE} the probability of error of the additively homomorphic encryption scheme AHE during the secure aggregation. Note that these probabilities of error refer to the evaluation correctness definition (see Definition 8 and Definition 11) and not the decryption correctness, the second one ensuring correctness of decryption only for fresh ciphertexts. While in general secret-key and public-key encryption schemes without homomorphic property provides perfect correctness, for the sake of formalism, we denote by ϵ^{SKE} and ϵ^{PKE} the probability of decryption error respectively for the secret-key encryption scheme SKE and for the public-key encryption scheme PKE.

Removal of the Permutation. The permutation p_t generated at each time t is used to hide the identity of the arm chosen M. As a response, the proxy node obtains K selection bits $c^b_{p_t(i),t}$ encrypted under the SKE scheme, that are permuted with respect

to the permutation p_t randomly chosen by $\mathcal{P}$. Since $\mathcal{P}$ knows the permutation, it inverts the permutation with p_t^{-1} leading the ordered encrypted selection bit $c_{i,t}^b$. Hence, the selection bits are correctly assigned. Thus the correctness is ensured even in the presence of the permutation, so removing the permutation does not affect the arm selection.

Removal of the Homomorphic Arm Selection. The homomorphic evaluation of the arg max function using the FHE scheme results in the index M. The correctness of the multi-armed bandits algorithm, modelled under the MAB model, heavily holds under the hypothesis that M is indeed correctly evaluated using the FHE scheme. Due to the probability of error during the homomorphic arg max evaluation followed by the K equality tests, the returned total cumulative rewards s can differ with and without the FHE scheme. Assuming the error probability ϵ^{FHE} that an error occurs during the decryption procedure after the homomorphic evaluation of the arg max evaluation and the K equality tests, since the reflector node $\mathcal{R}$ decrypts K FHE ciphertexts, at each time step the probability that no decryption error occurs equals $(1-\epsilon^{\mathsf{FHE}})^K$. Then, this probability for all the time steps equals $(1-\epsilon^{\mathsf{FHE}})^{NK}$.

Removal of the Secret-Key and Public-Key Encryptions. At this point, the permutation p_t as well as the FHE scheme are removed. This let us with a hybrid version of Tango, where only the secret-keys of the SKE scheme, the selection bits and the cumulative rewards are encrypted. Following the same approach as above, for each time step, we remove the encryption of the SKE and the PKE schemes, transmitting all the data in clear instead. This transition does not affect the total cumulative rewards s as long as that the decryption of the K secret-keys and the K selection bits are correctly decrypted respectively by the reflector node $\mathcal{R}$ and by every data owner. The probability that no error occurs during the decryption for all the time steps equals $(1-\epsilon^{\mathsf{SKE}})^{NK} \cdot (1-\epsilon^{\mathsf{PKE}})^{NK}$.

Removal of the Homomorphic Aggregation. The remaining cryptographic operations occur in the $\mathsf{SendRewards}$ protocol, in which each cumulative rewards s_i is encrypted with the $\mathcal{U}$'s encryption public key of the AHE scheme. After the homomorphic addition performed by $\mathcal{P}$, the resulting ciphertext encrypting s is sent to the user $\mathcal{U}$. The resulting total cumulative rewards s obtained by $\mathcal{U}$ is valid if and only if the homomorphic addition using the AHE scheme is correct, which happens with probability $1-\epsilon^{\mathsf{AHE}}$. Note that this probability is the same even in case of the failure of a data owner $\mathcal{DO}_i$ using our reward saving mechanism in the sense that it does not involve an additional encryption, addition or decryption.

The resulting protocol, that we denote FedMab for clarity, no more contains any cryptographic operation and hence corresponds to the $\mathsf{MAB\text{-}Rewards}$ algorithm depicted in Figure 3.6. In FedMab, the correctness of the multi-armed bandits algorithm is clear: The arm selection is computed via the arg max function evaluated over unencrypted scores. As we have observed during this sequence, the returned total cumulative rewards s between FedMab and Tango are identical, at condition that all cryptographic decryptions act correctly. By aggregating all the aforementioned probabilities, we can infer the probability p that no error occurs due to cryptography (hence returning the same total

	KeyGen				SelectArm, N times				SendRewards			
	$\mathcal{DO}_i$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{U}$	$\mathcal{DO}_i$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{U}$	$\mathcal{DO}_i$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{U}$
SKE.KeyGen					1							
Enc							K					
Dec					1							
PKE.KeyGen			1									
Enc					1							
Dec							K					
FHE.KeyGen			1									
Enc					1							
Eval						$\mathcal{O}(K)$						
Dec							K					
AHE.KeyGen				1								
Enc					1				1			
Add										K-1		
Dec												1

Figure 3.13: Cryptographic operations performed in the protocol.

cumulative rewards s), defined as $p = (1-\epsilon^{\mathsf{FHE}})^{NK} \cdot (1-\epsilon^{\mathsf{SKE}})^{NK} \cdot (1-\epsilon^{\mathsf{PKE}})^{NK} \cdot (1-\epsilon^{\mathsf{AHE}})$. Then, we state TANGO ensures p-correctness.

3.3.2 Complexity of TANGO

During our protocol, five cryptographic primitives are necessary, namely the fully homomorphic encryption scheme FHE, the secret-key and public key encryption schemes SKE and PKE, the additive homomorphic encryption scheme AHE as well as the pseudo-random permutation PRP. We omit the pseudo-random permutation from our analysis since it is a fairly straightforward and efficient primitive, which leave us with four cryptographic primitives. For the sake of clarity, we have depicted in Figure 3.13 a summary of the usage of each primitive during the key generation and the two protocols SelectArm and SendRewards.

During the setup, the user $\mathcal{U}$ as well as the reflector node $\mathcal{R}$ generate key pairs for the public-key encryption schemes including the public-key encryption scheme PKE, the additively homomorphic encryption scheme AHE and the fully homomorphic encryption scheme FHE.

During the arm selection protocol SelectArm, every data owner $\mathcal{DO}_i$ encrypts its own score, its own cumulative rewards as well as its own freshly generated secret-key, using three distinct schemes, namely the FHE, AHE and PKE schemes. At the end of the protocol, $\mathcal{DO}_i$ receives a ciphertext encrypting its selection bit that has to be decrypted using the SKE scheme. Since the SelectArm is executed for each time step $t \in [\![N]\!]$, the asymptotic complexity of $\mathcal{DO}_i$ for all the executions of the SelectArm protocol is $\mathcal{O}(N \cdot (\mathcal{O}(\mathsf{FHE.Enc}) + \mathcal{O}(\mathsf{AHE.Enc}) + \mathcal{O}(\mathsf{PKE.Enc}) + \mathcal{O}(\mathsf{SKE.Dec})))$. From a cryptographic standpoint, the proxy node $\mathcal{P}$ performs the most time-consuming operation consisting of the homomorphic evaluation of the arg max function (applied on K scores) followed by K equality testings. For simplicity, we hide all these operations under the FHE.Eval algortihm. Since this step is performed N times, the asymptotic complexity of $\mathcal{P}$ for all the executions of SelectArm is $\mathcal{O}(NK \cdot \mathsf{FHE.Eval})$. The reflector node $\mathcal{R}$, on its side,

decrypts the encrypted selection bits, encrypted under the FHE scheme as well as the secret-keys encrypted under the PKE scheme. It also encrypts all the selection bits using the SKE scheme. Since we have K encrypted secret-keys to decrypt with PKE, K selection bits to decrypt with PKE and K selection bits to encrypt with SKE, and since the SelectArm protocol is executed N times, then for all the executions of SelectArm, $\mathcal{R}$ has an asymptotic complexity of $\mathcal{O}(NK \cdot (\mathcal{O}(\mathsf{PKE.Dec}) + \mathcal{O}(\mathsf{SKE.Enc}) + \mathcal{O}(\mathsf{FHE.Dec})))$.

During the SendRewards protocol, every data owner $\mathcal{DO}_i$ encrypts its own cumulative rewards s_i using the AHE scheme. Hence, the asymptotic complexity of $\mathcal{DO}_i$ during the SendRewards protocol is $\mathcal{O}(\mathsf{AHE.Enc})$. The proxy node $\mathcal{P}$ aggregates the cumulative rewards s_i to obtain the encryption of the total cumulative rewards s using the homomorphic property of the AHE scheme. Hence, the asymptotic complexity of the proxy node during the SendRewards protocol is $\mathcal{O}(K \cdot \mathcal{O}(\mathsf{AHE.Add}))$. Finally, the user $\mathcal{U}$ receives and decrypts the encryption of s, still using the AHE scheme, leading to the complexity $\mathcal{O}(\mathsf{AHE.Dec})$.

The final asymptotic complexities for all the parties involved in the protocol are now easy to deduce by summing all the asymptotic complexities. In particular, a data owner $\mathcal{DO}_i$ has complexity $\mathcal{O}(N \cdot (\mathcal{O}(\mathsf{FHE.Enc}) + \mathcal{O}(\mathsf{AHE.Enc}) + \mathcal{O}(\mathsf{PKE.Enc}) + \mathcal{O}(\mathsf{SKE.Dec})) + \mathsf{AHE.Enc})$. The proxy node $\mathcal{P}$ has complexity $\mathcal{O}(NK \cdot (\mathcal{O}(\mathsf{FHE.Eval}) + K \cdot \mathsf{AHE.Add})$ having the NK factor, which is similar to the asymptotic complexity of the reflector $\mathcal{R}$ of $\mathcal{O}(NK \cdot (\mathcal{O}(\mathsf{PKE.Dec}) + \mathcal{O}(\mathsf{FHE.Dec}) + \mathcal{O}(\mathsf{SKE.Dec})))$. Finally, the user $\mathcal{U}$ has complexity $\mathcal{O}(\mathsf{AHE.Dec})$.

3.3.3 Security of Tango

To prove the security of Tango, we proceed for each party into a sequence of games, starting from the real protocol to the ideal protocol where the security is easy to prove. We start the security proof by the controller, implemented in Tango by the proxy and reflector nodes, then we prove the security for the user and finally for a data owner.

Security Proofs for Proxy and Reflector

We first focus on the case of the controller $\mathcal{C}$, being splitted into two distinct non-colluding nodes, namely the reflector node $\mathcal{R}$ and the proxy node $\mathcal{P}$. To prove that the controller $\mathcal{C}$ does not learn any information on any score $v_{i,t}$, on any cumulative rewards n_i, on any number of pulls for a data owner $\mathcal{DO}_i$ at some time step t, or on the total cumulative rewards s, we proceed with two lemmas. The first lemma considers the case of the (honest-but-curious) corruption of the reflector node $\mathcal{R}$, whereas the second lemma considers the case of a the (honest-but-curious) corruption of the proxy node $\mathcal{P}$. For clarity, we denote by RealR (respectively RealP) the real experiment being the same of the real experiment RealC, except that the adversary corrupts $\mathcal{R}$ (respectively $\mathcal{P}$). Similarly, we denote by IdealR (respectively IdealP) the ideal experiment being the ideal experiment IdealC, except that the adversary corrupts only $\mathcal{R}$ (respectively $\mathcal{P}$). To prove the simulation security SimC of Tango, it suffices to prove these two lemmas.

Lemma 1. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, for every security parameter $\lambda \in \mathbb{N}$, every polynomial-time adversary $\mathcal{A}$, there exists a

simulator $\mathcal{S}$ such that TANGO is statistically SimR-secure:

$$\Pr\left[\mathsf{G}^{\mathsf{RealR}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \overset{\mathrm{s}}{\approx} \Pr\left[\mathsf{G}^{\mathsf{IdealR}}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Proof. Recall that our security definition requires the definition of the view of $\mathcal{R}$, denoted $\mathsf{View}(\mathcal{R})$, containing every piece of data collected during the SelectArm protocol, including $\{k_{p_t(i),t}, b_{p_t(i),t}\}_{i\in[\![K]\!],t\in[\![N]\!]}$. In order to prove the above theorem, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ used in the game $\mathsf{Exp}^{\mathsf{IdealR}}_{\mathcal{A},\mathcal{S}}$:

- *Simulator* $\mathcal{S}_0$. Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$ and the number of arms K. At the end of the execution, for some time step $t \in [\![K]\!]$, the simulator $\mathcal{S}_0$ is required to output a K-sized list of tuples $(s'_i, n'_i, v'_{i,t})$ for $i \in [\![K]\!]$ where s'_i is a random cumulative rewards, n'_i is a random number of pulls and $v'_{i,t}$ is a random score.
  The simulator $\mathcal{S}_0$ starts by choosing randomly K scores $v'_{1,t}, \ldots, v'_{K,t}$, K random cumulative rewards $s'_1, \ldots, s'_K$ and K random number of pulls $n'_1, \ldots, n'_K$. Finally, it outputs the list of tuples $(s'_i, n'_i, v'_{i,t})$ for every $i \in [\![K]\!]$.
- *Simulator* $\mathcal{S}_1$. Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K, and produces the simulated cumulative rewards s'_i randomly chosen from an arbitrary space, say $\mathbb{N}$, for each arm $i \in [\![K]\!]$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the real experiment $\mathsf{Exp}^{\mathsf{RealR}}_{\mathcal{A}}$, and the ideal world corresponding to the experiment $\mathsf{Exp}^{\mathsf{IdealR}}_{\mathcal{A},\mathcal{S}}$.

Game G^0. This game corresponds to our real experiment $\mathsf{Exp}^{\mathsf{RealR}}_{\mathcal{A}}$, hence:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \overset{\mathrm{p}}{=} \Pr\left[\mathsf{Exp}^{\mathsf{RealR}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right]$$

Game G^1. This game is the same as G^0, except that we introduce our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ in the game. Our first simulator $\mathcal{S}_0$ is inputted at each time $t \in [\![N]\!]$ with the appropriate inputs *i.e.,* the current time t and the number of arms K. Our second simulator $\mathcal{S}_1$ is also added to the game, inputted with the number of arms K. Note that the outputs of our simulator, including the list of tuples $(s'_i, n'_i, v'_{i,t})$ for each $i \in [\![K]\!]$ and for each time step $t \in [\![N]\!]$ is not used as well as the outputted random cumulative rewards s'_i at the end of the protocol execution. Thus it does *not* impact the view of $\mathcal{R}$, providing no advantage for the adversary to distinguish:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \overset{\mathrm{p}}{=} \Pr\left[\mathsf{G}^1_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^2. This game is the same as G^1 except that we focus on the cumulative rewards s_i inputted to the SendRewards protocol, allowing the user $\mathcal{U}$ to obtain the total cumulative rewards s. Remark that no information about the total cumulative rewards s or on any cumulative rewards s_i is contained on the view of $\mathcal{R}$. Therefore, in this game, we are allowed to replace the real cumulative rewards by the random cumulative rewards computed by our simulator $\mathcal{S}_1$, inputted with the number of arms K, and outputting K random cumulative rewards $s'_1, ..., s'_K$. Observe that the view of $\mathcal{R}$ is completely independent of this modification, leading to a perfect indistinguishability between the

game G^1 and the game G^2:

$$\Pr\left[\mathsf{G}^1_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^3. This game is the same as G^2 except that we focus on the cumulative rewards s_i and the number of pulls n_i inputted to the $\mathsf{SelectArm}$ protocol. In TANGO, each cumulative rewards s_i is sent encrypted under the AHE scheme to the proxy node $\mathcal{P}$ in order to continue the protocol execution even in case of a data owner shutdown. Observe that all encrypted cumulative rewards s_i remain on the proxy node $\mathcal{P}$ and thus are not in the view of the reflector node $\mathcal{R}$. Hence, as done previously, we are allowed to replace the real cumulative rewards s_i by the random cumulative rewards s'_i computed by our simulator $\mathcal{S}_0$, while having a perfect indistinguishability between G^2 and G^3:

$$\Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^4. This game is the same as G^3, except that at each time $t \in [\![N]\!]$, each score $v_{i,t}$ for $i \in [\![K]\!]$ is replaced by the random score $v'_{i,t}$.

Let us explain why this game is statistically indistinguishable from G^3: Recall that the view of $\mathcal{R}$ is composed at each time step $t \in [\![N]\!]$ of K permuted selection bits $b_{p_t(1),t}, \ldots, b_{p_t(K),t}$ computed via the homomorphic arg max evaluation over the provided scores, where p_t is the pseudo-random permutation chosen by $\mathcal{P}$. The view of $\mathcal{R}$ also contains the secret-key $k_{p_t(i),t}$ obtained after the decryption. Observe that by replacing each real score $v_{i,t}$ with a completely random score $v'_{i,t}$, the best arm index M will necessary be altered. Let denote M' the best arm index computed over random scores. Clearly, without the permutation p_t the reflector node $\mathcal{R}$ would notice this modification. This fact is strengthened by the fact that the adversary provides the reward probabilities for all the arms. Thanks to the permutation p_t, $\mathcal{R}$ has to distinguish between the case where the only one selection bit equaling one is $b_{p_t(M),t}$ and the case where the only selection bit equaling one is $b_{p_t(M'),t}$. Furthermore, note that the permutation p_t is different and randomly chosen at each time step $t \in [\![N]\!]$, hence the probability to have $b_{p_t(M'),t}$ at one equals the probability to receive $b_{p'_t(M),t}$ for some permutation p'_t. In other words, $\mathcal{R}$ cannot distinguish between the case where the arg max has been performed over real or random scores, given only the permuted index $p_t(M)$ where $b_{p_t(M)}$ equals one. This argument holds if the generated pseudo-random permutation p_t is indeed random, otherwise an adversary would be able to distinguish using a statistical attack. As a result, the view of $\mathcal{R}$ is statistically indistinguishable between G^3 and G^4:

$$\Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \overset{\text{s}}{\approx} \Pr\left[\mathsf{G}^4_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Observe that the game G^4 is exactly our ideal world, corresponding to the IdealR experiment, leading to the following equality:

$$\Pr\left[\mathsf{G}^4_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{IdealR}}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Hence, we have shown that:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealR}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \overset{\mathrm{s}}{\approx} \Pr\left[\mathsf{Exp}^{\mathsf{IdealR}}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

□

Lemma 2. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, assuming an IND-CPA-secure secret-key encryption scheme SKE, an IND-CPA-secure public-key encryption scheme PKE, an IND-CPA-secure fully homomorphic encryption scheme FHE and an IND-CPA-secure additively homomorphic encryption scheme AHE for every security parameter $\lambda \in \mathbb{N}$, then polynomial-time adversary $\mathcal{A}$, there exists a polynomial-time simulator $\mathcal{S}$ such that Tango is SimP-secure with the following bound:

$$\begin{aligned}&\left|\Pr\left[\mathsf{Exp}^{\mathsf{RealP}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{IdealP}}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \\ &\leq 2NK \cdot (\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{PKE}} + \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{SKE}} + \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{FHE}} + \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{AHE}}) + 2K \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{AHE}}\end{aligned}$$

Proof. Recall that our security definition requires the definition of the view of $\mathcal{P}$, denoted $\mathsf{View}(\mathcal{P})$, containing the encrypted total cumulative rewards c^s, all the ciphertexts exchanged during the arm selection protocol (including $\{c^s_{i,t}, c^v_{i,t}, c^k_{i,t}, c^b_{i,t}, p_t\}_{i\in[\![K]\!],t\in[\![N]\!]}$) and all the ciphertexts exchanged during the sending of the total cumulative rewards (including $\{s_i, n_i, c^s_i\}_{i\in[\![K]\!]}$). In order to prove the above theorem, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ used in the game $\mathsf{Exp}^{\mathsf{IdealP}}_{\mathcal{A},\mathcal{S}}$:

- *Simulator $\mathcal{S}_0$.* Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$ and the number of arms K. At the end of the execution, the simulator $\mathcal{S}_0$ is required to output two lists: A K-sized list of tuples $(s'_i, n'_i, v'_{i,t})$ for $i \in [\![K]\!]$ where s'_i is a random cumulative rewards, n'_i is a random number of pulls and $v'_{i,t}$ is a random score.

  Observe that in the $\mathsf{Exp}^{\mathsf{IdealP}}_{\mathcal{A},\mathcal{S}}$ experiment, the simulator $\mathcal{S}_0$ is not required to ensure correctness of the multi-armed bandits execution, since the view of $\mathcal{P}$ is only composed of ciphertexts. Hence, our simulator independently samples each random score $v'_{i,t}$ from an arbitrary space, say $[\![N]\!]$.

- *Simulator $\mathcal{S}_1$.* Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K, and produces the random rewards s'_i randomly chosen from an arbitrary space, say $\mathbb{N}$ for each arm $i \in [\![K]\!]$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the experiment $\mathsf{Exp}^{\mathsf{RealR}}_{\mathcal{A}}$, and the ideal world corresponding to $\mathsf{Exp}^{\mathsf{IdealR}}_{\mathcal{A},\mathcal{S}}$.
Game G^0. This initial game corresponds to the $\mathsf{Exp}^{\mathsf{RealP}}_{\mathcal{A}}$ experiment, hence:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \overset{\mathrm{p}}{=} \Pr\left[\mathsf{Exp}^{\mathsf{RealP}}_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^1. This game is the same as G^0, except that we introduce our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ as follows: The first simulator $\mathcal{S}_0$ is inputted at each time $t \in [\![N]\!]$ with the appropriate inputs *i.e.,* the number of arms K. The output of $\mathcal{S}_0$ is composed of K

tuples where each tuple contains a random cumulative rewards s'_i, a random number of pulls n'_i and a random score $v'_{i,t}$. Note that at this point, this output is not used. The second simulator $\mathcal{S}_1$ is inputted with the number of arms K. The output of the simulator $\mathcal{S}_1$ consists of K random rewards s'_i for each $i \in [\![K]\!]$, and is also not used for the moment. Since the output of the simulator $\mathcal{S}$ is not used, the view of the adversary remains unchanged. Then, G^0 and G^1 are perfectly indistinguishable:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \stackrel{\text{p}}{=} \Pr\left[\mathsf{G}^1_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^k for $k \in [\![2, K+1]\!]$. This game is the same as G^{k-1}, except that we replace the encryption of cumulative rewards s_i inputted to the $\mathsf{SendRewards}$ protocol, i being the $(k-1)$-th data owner, with an encryption of the random cumulative rewards s'_i chosen by the simulator $\mathcal{S}_1$. Since the view of $\mathcal{P}$ contains the encryption of s'_i, then its view is modified. Now we prove that $\mathcal{P}$ cannot distinguish between this game G^k and the previous game G^{k-1} thanks to computational indistinguishability of the AHE scheme.

For the sake of contradiction, suppose a polynomial-time adversary $\mathcal{A}$ having a non-negligible advantage $\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k-1},\mathsf{G}^k}$ to distinguish between the distribution of G^{k-1} and G^k, following the view of $\mathcal{P}$. We construct an adversary $\mathcal{B}$ against the IND-CPA security of the AHE scheme, working as follows: After generating the key pair $(sk^{\mathsf{AHE}}, pk^{\mathsf{AHE}})$, the challenger for the $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{B}}$ experiment sends the public encryption key pk^{AHE} to the adversary $\mathcal{B}$, simulating the $\mathsf{G}^k_{\mathcal{A},\mathcal{S}}$ experiment against $\mathcal{A}$. The adversary $\mathcal{B}$ considers pk^{AHE} as $pk^{\mathsf{AHE}}_{\mathcal{U}}$. Since $\mathcal{B}$ has access to the public key $pk^{\mathsf{AHE}}_{\mathcal{U}}$, every encryption can be directly computed by $\mathcal{B}$ in the simulated experiment. Before to encrypt the cumulative rewards s_i, given the random cumulative rewards s'_i outputted by our second simulator $\mathcal{S}_1$, the adversary $\mathcal{B}$ sends $m_0 = s_i$ and $m_1 = s'_i$ to the challenger of the IND-CPA experiment, and expects as a response the challenge ciphertext encrypting m_b for a random bit b chosen by the IND-CPA challenger. The returned ciphertext is considered as c^s_i. Then, $\mathcal{B}$ continues the protocol execution, sending c^s_i to $\mathcal{P}$ following the protocol specification, and hence in the view of $\mathcal{A}$. At the end of the experiment, $\mathcal{A}$ responds to $\mathcal{B}$ with a prediction bit b', forwarded to the challenger. Clearly, the advantage of $\mathcal{B}$ against the IND-CPA security of AHE highly depends on the advantage of $\mathcal{A}$:

$$\begin{aligned}
\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{AHE}} &= \left|\Pr[b = b'] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2} \cdot \Pr[b' = 0 | b = 0] + \frac{1}{2} \cdot \Pr[b' = 1 | b = 1] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2} \cdot \Pr[b' = 0 | b = 0] + \frac{1}{2} \cdot (1 - \Pr[b' = 1 | b = 0]) - \frac{1}{2}\right| \\
&= \left|\frac{1}{2} \cdot (\Pr[b' = 0 | b = 0] - \Pr[b' = 1 | b = 0])\right| \\
&= \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k-1},\mathsf{G}^k}
\end{aligned}$$

Therefore, the advantage for the adversary to distinguish between G^{k-1} and the game G^{k} is bounded by the indistinguishability of the AHE scheme:

$$\left|\Pr\left[\mathsf{G}^{k-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{k}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{AHE}}$$

Hence, we conclude that the game G^2 is computationally indistinguishable from the game G^{K+1} with the following bound:

$$\left|\Pr\left[\mathsf{G}^{2}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{K+1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2K \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{AHE}}$$

We now proceed to N sequences of $4K$ games. Intuitively, we will replace part of the $\mathsf{SelectArm}$ protocol including the encrypted cumulative rewards s'_i, the encrypted secret-key $k_{i,t}$, the encrypted selection bit $b'_{i,t}$ and the encrypted score $v_{i,t}$ by random values. For the sake of clarity, we denote by $\mathsf{G}^{k,l}$ the l-th game of the k-th sequence of games, with k ranging from 1 to N and l ranging from 1 to $4K$. The game $\mathsf{G}^{k,0}$ for some k refers to the game $\mathsf{G}^{k-1,4K}$, the game $\mathsf{G}^{0,4K}$ being the game G^{K+1}. For clarity, we subdivide the k-th sequence of $4K$ games in four sequences of K games where l is ranging respectively from 1 to K in the first sequence, from $K+1$ to $2K$ in the second sequence, from $2K+1$ to $3K$ in the third sequence and from $3K+1$ to $4K$ in the fourth and last sequence.

Game $\mathsf{G}^{k,l}$ for $k \in [\![1, N]\!]$ and for $l \in [\![1, K]\!]$. In this game indexed by k, we focus on the ciphertext $c^s_{i,t}$ sent by each data owner $\mathcal{DO}_i$ with i at l during the $\mathsf{SelectArm}$ protocol. Note that we are modifying the $\mathsf{SelectArm}$ protocol from each $\mathcal{DO}_i$ for a given time step t, from the last time step N to the first time step where t equals 1 in this order. Hence, t equals $N-k-1$. In this game, we replace the cumulative rewards s_i by the random cumulative rewards s'_i generated by the first simulator $\mathcal{S}_0$ at time t. The view of $\mathcal{P}$ contains the ciphertext $c^s_{i,t}$ and hence the view of $\mathcal{P}$ is modified. We now show that this modification is indistinguishable, thanks to the indistinguishability of the AHE scheme.

For the sake of contradiction, suppose a polynomial-time adversary $\mathcal{A}$ having a non-negligible advantage $\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$ to distinguish between the distribution of $\mathsf{G}^{k,l-1}$ and $\mathsf{G}^{k,l}$, following the view of $\mathcal{P}$. We construct an adversary $\mathcal{B}$ against the IND-CPA security of the AHE scheme, working as follows: After generating the key pair $(sk^{\mathsf{AHE}}, pk^{\mathsf{AHE}})$, the challenger of the IND-CPA experiment for the AHE scheme sends the public encryption key pk^{AHE} to the adversary $\mathcal{B}$, simulating the experiment $\mathsf{G}^{k,l}_{\mathcal{A},\mathcal{S}}$ against $\mathcal{A}$, considering pk^{AHE} as $pk^{\mathsf{AHE}}_{\mathcal{R}}$. Since $\mathcal{B}$ has access to the public key $pk^{\mathsf{AHE}}_{\mathcal{R}}$, every encryption can be directly computed by $\mathcal{B}$ in the simulated experiment. During the time t, before to encrypt the cumulative rewards s_i, the adversary $\mathcal{B}$ defines the two message m_0 and m_1 such that $m_0 = s_i$ and $m_1 = s'_i$ to the challenger. It expects as a response the challenge ciphertext encrypting m_b for a random bit b chosen by the challenger. The returned ciphertext is considered as $c^s_{i,t}$. Then, $\mathcal{B}$ continues the protocol execution by sending $c^s_{i,t}$ to $\mathcal{P}$ and hence in the view of $\mathcal{A}$. At the end of the experiment, $\mathcal{A}$ responds to $\mathcal{B}$ with a prediction bit b', forwarded to the challenger. Clearly, the advantage of $\mathcal{B}$ against the IND-CPA security of AHE highly depends on the advantage of $\mathcal{A}$, defined as follows:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{AHE}} = \left|\Pr[b = b'] - \frac{1}{2}\right| = \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$$

Therefore, the advantage for the adversary to distinguish between $\mathsf{G}^{k,l-1}$ and the game $\mathsf{G}^{k,l}$ is bounded by the computational indistinguishability of the AHE scheme:

$$\left|\Pr\left[\mathsf{G}^{k,l-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{k,l}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{AHE}}$$

By induction, we conclude that the game $\mathsf{G}^{k,1}$ is computationally indistinguishable from the game $\mathsf{G}^{k,K}$ with the following bound:

$$\left|\Pr\left[\mathsf{G}^{k,1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{k,K}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2 \cdot K \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{AHE}}$$

Game $\mathsf{G}^{k,l}$ for $k \in [\![1, N]\!]$ and for $l \in [\![K+1, 2K]\!]$. In this game indexed by k, we focus on the ciphertext $c^k_{i,t}$ sent by the data owner $\mathcal{DO}_i$ with i at $l - K$ during the $\mathsf{SelectArm}$ protocol. Similarly to our previous sequence of games, we are modifying the $\mathsf{SelectArm}$ protocol from the last time step N to the first time step where t equals 1. Hence t equals $N - k - 1$. In this game, we replace the freshly generated key $k_{i,t}$ by a completely random key $k'_{i,t}$. Since $k_{i,t}$ is used once during all the protocol execution and transmitted encrypted once to $\mathcal{P}$, the only difference between $\mathsf{G}^{k,l-1}$ and $\mathsf{G}^{k,l}$ in the view of $\mathcal{P}$ is the ciphertext $c^k_{i,t}$. We have now to prove the view of $\mathcal{P}$ between these two games are computationally indistinguishable, thanks to the indistinguishability of the public encryption scheme PKE.

For the sake of contradiction, suppose a polynomial-time adversary $\mathcal{A}$ having a non-negligible advantage $\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$ to distinguish between the distribution of $\mathsf{G}^{k,l-1}$ and $\mathsf{G}^{k,l}$, following the view of $\mathcal{P}$. We construct an adversary $\mathcal{B}$ against the IND-CPA security of the PKE scheme, working as follows: After generating the key pair $(sk^{\mathsf{PKE}}, pk^{\mathsf{PKE}})$, the challenger sends the public encryption key pk^{PKE} to the adversary $\mathcal{B}$, simulating the experiment $\mathsf{G}^{k,l}_{\mathcal{A},\mathcal{S}}$ against $\mathcal{A}$, considering pk^{PKE} as $pk^{\mathsf{PKE}}_{\mathcal{R}}$. Since $\mathcal{B}$ has access to the public key $pk^{\mathsf{PKE}}_{\mathcal{R}}$, every encryption can be directly computed by $\mathcal{B}$ in the simulated experiment. During the time t, before to encrypt the generated key $k_{i,t}$, the adversary $\mathcal{B}$ randomly chooses a random value $k'_{i,t}$. After sending $m_0 = k_{i,t}$ and $m_1 = k'_{i,t}$ to the challenger, $\mathcal{B}$ expects as a response the challenge ciphertext encrypting m_b for a random bit b chosen by the challenger. The returned ciphertext is considered as $c^k_{i,t}$. Then, $\mathcal{B}$ continues the protocol execution, sending $c^k_{i,t}$ to $\mathcal{P}$ and hence in the view of $\mathcal{A}$. At the end of the experiment, $\mathcal{A}$ responds to $\mathcal{B}$ with a prediction bit b', forwarded to the challenger. Clearly, the advantage of $\mathcal{B}$ against the IND-CPA security of PKE highly depends on the advantage of $\mathcal{A}$, defined as follows:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{PKE}} = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right| = \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$$

Therefore, the advantage for the adversary to distinguish between $\mathsf{G}^{k,l-1}$ and the game $\mathsf{G}^{k,l}$ is bounded by the computational indistinguishability of the PKE scheme:

$$\left|\Pr\left[\mathsf{G}^{k,l-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{k,l}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{PKE}}$$

By induction, we conclude that the game $\mathsf{G}^{k,K+1}$ is computationally indistinguishable from the game $\mathsf{G}^{k,2K}$ with the following bound:

$$\left|\Pr\left[\mathsf{G}^{k,K+1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{k,2K}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2K \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{PKE}}$$

Game $\mathsf{G}^{k,l}$ for $k \in [\![1, N]\!]$ and for $l \in [\![2K+1, 3K]\!]$. In this game, we are focused on the selection bit $b_{i,t}$ sent by the data owner $\mathcal{DO}_i$ with i at $l - 2K$ during the $\mathsf{SelectArm}$ protocol. In this game proceed to the replacement of the selection bit $b_{i,t}$ for the time step t at $N - k$. In this game, we replace the selection bit $b_{i,t}$ by a randomly chosen selection bit $b'_{i,t}$ from the binary space $\{0, 1\}$.

For the sake of contradiction, suppose a polynomial-time adversary $\mathcal{A}$ having a non-negligible advantage $\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$ to distinguish between the distribution of $\mathsf{G}^{k,l-1}$ and $\mathsf{G}^{k,l}$, following the view of $\mathcal{P}$. We construct an adversary $\mathcal{B}$ against the IND-CPA security of the SKE scheme, working as follows: Since the secret-key is kept by the challenger, the adversary $\mathcal{B}$ simulating the experiment $\mathsf{G}^{k,l}_{\mathcal{A},\mathcal{S}}$ against $\mathcal{A}$ has now access to an encryption oracle to compute SKE's ciphertexts. After sending to the IND-CPA challenger the two messages m_0 corresponding to the real selection bit $b_{i,t}$, and m_1 corresponding to a random selection bit $b'_{i,t}$, $\mathcal{B}$ expects as a response the challenge ciphertext encrypting m_b for a random bit b chosen by the challenger. The returned ciphertext is considered as $c^b_{i,t}$. Then, $\mathcal{B}$ continues the protocol execution, sending $c^b_{i,t}$ to $\mathcal{P}$ and hence in the view of $\mathcal{A}$. At the end of the experiment, $\mathcal{A}$ responds to $\mathcal{B}$ with a prediction bit b', forwarded to the challenger. We obtain the advantage of $\mathcal{B}$ as follows:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{SKE}} = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right| = \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$$

Therefore, we obtain the following bound:

$$\left|\Pr\left[\mathsf{G}^{k,l-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{l,k}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{SKE}}$$

By induction, we conclude that the game $\mathsf{G}^{k,l-1}$ is computationally indistinguishable from the game $\mathsf{G}^{k,l}$ with the following bound:

$$\left|\Pr\left[\mathsf{G}^{k,2K+1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] - \Pr\left[\mathsf{G}^{k,3K}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]\right| \leq 2 \cdot K \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B},\mathsf{SKE}}$$

Game $\mathsf{G}^{k,l}$ for $k \in [\![1, N]\!]$ and for $l \in [\![3K+1, 4K]\!]$. In this game indexed by k, we focus on the data owner $\mathcal{DO}_i$ with i at $l - 3K$, sending during $(N - k)$-th execution of the $\mathsf{SelectArm}$ protocol the score $v_{i,t}$. In this game, we replace $c^v_{i,t}$ encrypting the score $v_{i,t}$ by an encryption of the random score $v'_{i,t}$ chosen by the simulator $\mathcal{S}_0$.

For the sake of contradiction, suppose a polynomial-time adversary $\mathcal{A}$ having a non-negligible advantage $\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{A},\mathsf{G}^{k,l-1},\mathsf{G}^{k,l}}$ to distinguish between the distribution of $\mathsf{G}^{k,l-1}$ and $\mathsf{G}^{k,l}$, following the view of $\mathcal{P}$. We construct an adversary $\mathcal{B}$ against the IND-CPA security of the FHE scheme, working as follows: The adversary $\mathcal{B}$ simulating the experiment $\mathsf{G}^{k,l}_{\mathcal{A},\mathcal{S}}$ against $\mathcal{A}$ have access to the public encryption key $pk^{\mathsf{FHE}}_{\mathcal{R}}$, useful to send encrypted scores. During the time t, before to encrypt the score $v_{i,t}$ the adversary $\mathcal{B}$ sends $m_0 = v_{i,t}$ corresponding to the real score and $m_1 = v'_{i,t}$ the random score chosen by the simulator $\mathcal{S}_0$, to the challenger. As a response, $\mathcal{B}$ obtains the challenge ciphertext encrypting m_b

for a random bit b chosen by the challenger. The returned ciphertext is considered as $c_{i,t}^{v}$. Then, $\mathcal{B}$ continues the protocol execution, sending $c_{i,t}^{v}$ to $\mathcal{P}$ and hence in the view of $\mathcal{A}$. At the end of the experiment, $\mathcal{A}$ responds to $\mathcal{B}$ with a prediction bit b', forwarded to the challenger. We obtain the advantage of $\mathcal{B}$ as follows:

$$\mathsf{Adv}_{\mathcal{B},\mathsf{FHE}}^{\text{IND-CPA}} = \left|\Pr\left[b = b'\right] - \frac{1}{2}\right| = \frac{1}{2} \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Dist}}$$

Therefore, we obtain the following bound:

$$\left|\Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{k,l-1}(1^{\lambda}, N, K) \to 1\right] - \Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{k,l}(1^{\lambda}, N, K) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}_{\mathcal{B},\mathsf{FHE}}^{\text{IND-CPA}}$$

By induction, we conclude that the game $\mathsf{G}^{k,3K+1}$ is computationally indistinguishable from the game $\mathsf{G}^{k,4K}$ with the following bound:

$$\left|\Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{k,3K+1}(1^{\lambda}, N, K) \to 1\right] - \Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{k,4K}(1^{\lambda}, N, K) \to 1\right]\right| \leq 2 \cdot K \cdot \mathsf{Adv}_{\mathcal{B},\mathsf{FHE}}^{\text{IND-CPA}}$$

Game $\mathsf{G}_{\mathcal{A},\mathcal{S}}^{N,4K}$. Observe that this game, obtained at the end of our previous sequence of games, corresponds to the ideal experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}$: Our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is integrated and acts as specified by the ideal world depicted in the $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}$. Therefore:

$$\Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{N,4K}(1^{\lambda}, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}(\lambda, N, K) \to 1\right]$$

Since the previous sequence of $4K$ games is executed N times, one for each time step, we deduce the bound obtained during this proof to distinguish between our real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealP}}$ and our ideal experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}$, described below:

$$\begin{aligned}
&\left|\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealP}}(1^{\lambda}, N, K) \to 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}(1^{\lambda}, N, K) \to 1\right]\right| \\
&\leq N \cdot (2K \cdot \mathsf{Adv}_{\mathcal{A},\mathsf{PKE}}^{\text{IND-CPA}} + 2K \cdot \mathsf{Adv}_{\mathcal{A},\mathsf{SKE}}^{\text{IND-CPA}} + 2K \cdot \mathsf{Adv}_{\mathcal{A},\mathsf{FHE}}^{\text{IND-CPA}} + 2K \cdot \mathsf{Adv}_{\mathcal{A},\mathsf{AHE}}^{\text{IND-CPA}}) \\
&+ 2K \cdot \mathsf{Adv}_{\mathcal{A},\mathsf{AHE}}^{\text{IND-CPA}} \\
&\leq 2NK \cdot (\mathsf{Adv}_{\mathcal{A},\mathsf{PKE}}^{\text{IND-CPA}} + \mathsf{Adv}_{\mathcal{A},\mathsf{SKE}}^{\text{IND-CPA}} + \mathsf{Adv}_{\mathcal{A},\mathsf{FHE}}^{\text{IND-CPA}} + \mathsf{Adv}_{\mathcal{A},\mathsf{AHE}}^{\text{IND-CPA}}) + 2K \cdot \mathsf{Adv}_{\mathcal{A},\mathsf{AHE}}^{\text{IND-CPA}}
\end{aligned}$$

And since we have assumed the IND-CPA security for all of our encryption schemes, all advantages to distinguish are negligible, hence we state that our real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealP}}$ and our ideal experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}$ are computationally indistinguishable, concluding our proof. □

By Lemma 1 and Lemma 2, we know that the honest-but-curious corruption of the reflector node $\mathcal{R}$ or the proxy node $\mathcal{P}$ does not allow the adversary to learn any information on the scores, on the cumulative rewards, on the total cumulative rewards but also on the pulled armed. And since the controller $\mathcal{C}$ is composed of the two nodes $\mathcal{R}$ and $\mathcal{P}$, we achieve the simulation security for the controller $\mathcal{C}$.

Security Proof for User

Theorem 1. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, for every security parameter $\lambda \in \mathbb{N}$, every polynomial-time adversary $\mathcal{A}$, there exists a

simulator $\mathcal{S}$ such that TANGO is perfectly SimU-secure:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \stackrel{\text{p}}{=} \Pr\left[\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Proof. The view of $\mathcal{U}$, denoted $\mathsf{View}(\mathcal{U})$, consists of the single ciphertext c^s encrypting the total cumulative rewards s. In order to prove the above theorem, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ used in the game $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}$:

- *Simulator* $\mathcal{S}_0$. Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$ and the number of arms K. At the end of its execution, the simulator $\mathcal{S}_0$ is required to output a list of tuples scores $(s'_i, n'_i, v'_{i,t})$ whose first element is a random cumulative rewards, the second element is a random number of pulls and the third and last element is a random score.
  The simulator $\mathcal{S}_0$ generates and returns K random scores $v'_{i,t}$, K random number of pulls n'_i and K cumulative rewards s'_i independently.
- *Simulator* $\mathcal{S}_1$. Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K as well as the total cumulative rewards s, and produces the simulated cumulative rewards s'_i randomly chosen from $\mathbb{N}$ for each arm $i \in [\![K]\!]$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the real experiment $\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}$, and the ideal world corresponding to the experiment $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}$.

Game G^0. This initial game corresponds to the $\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}$ experiment. Therefore, we have a perfect indistinguishability:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \stackrel{\text{p}}{=} \Pr\left[\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^1. This game works exactly as the game G^0 except that we replace the execution of the $\mathsf{MAB.PullArm}$ algorithm executed for all arms, by a single execution of $\mathsf{MAB.PullArm}$ for the data owner indexed by M, the index of the data owner having the highest score. As a result, the selection bits outputted by the arm selection protocol are now ignored. By correctness of the arm selection protocol $\mathsf{SelectArm}$, this modification does not affect the selected arm at each time step and hence does not affect the total cumulative rewards returned to the user $\mathcal{U}$. Hence, we have a perfect indistinguishability between the game G^0 and the game G^1:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \stackrel{\text{p}}{=} \Pr\left[\mathsf{G}^1_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^2. In this game, we focus on the integration of our first simulator $\mathcal{S}_0$: The simulator $\mathcal{S}_0$ is inputted at each time $t \in [\![N]\!]$ with t and the number of arms K. Recall that the output of the simulator $\mathcal{S}_0$ is composed of a K-sized list of tuples $(s'_i, n'_i, v'_{i,t})$. During the execution of the $\mathsf{SelectArm}$ protocol, at each time $t \in [\![N]\!]$ each data owner $\mathcal{DO}_i$ is inputted with its secret key sk_i and a random cumulative rewards s'_i, a random number of pulls n'_i and a random score $v'_{i,t}$. The obtained selection bits at the end of the arm selection protocol are ignored.

The view of the user $\mathcal{U}$ is limited to the total cumulative rewards s. The arm chosen by the $\mathsf{SelectArm}$ protocol is random since the arm selection protocol is inputted

with completely random values. However, thanks to the challenger executing the correct $\mathsf{MAB.PullArm}$ algorithm for the arm having the best score, the correctness of the multi-armed bandits algorithm is guaranteed. Therefore, the returned total cumulative rewards is perfect indistinguishability between this game and the previous game G^1:

$$\Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}_0}(\lambda, N, K) \to 1\right]$$

Game G^3. In this game, we focus on the integration of our second simulator $\mathcal{S}_1$, inputted at the end of the protocol with the number of arms K and the total cumulative rewards s, corresponding to the sum of all cumulative rewards (*i.e.,* $\sum_{i=1}^{K} s_i$). The simulator $\mathcal{S}_1$ is asked to output a K-sized list of random scores $s'_1, \ldots, s'_K$, where each s'_i is inputted to $\mathcal{DO}_i$ at the beginning of the $\mathsf{SendRewards}$ protocol execution. From the observation that both $s_1 + \cdots + s_K$ (computed by the real multi-armed bandits execution) and $s'_1 + \cdots + s'_K$ (randomly sampled from the simulator $\mathcal{S}_0$) equals s, so we conclude that the view of $\mathcal{U}$ between the game G^3 and the game G^2 is perfectly indistinguishable:

$$\Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}_0}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(\lambda, N, K) \to 1\right]$$

And since G^3 corresponds to the ideal experiment $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}$, we observe a perfect indistinguishability between $\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}$:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}(\lambda, N, K) \to 1\right]$$

□

Security Proof for a Data Owner

Theorem 2. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, for every security parameter $\lambda \in \mathbb{N}$, every polynomial-time adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Proof. The view of the corrupted data owner $\mathcal{DO}_j$, denoted $\mathsf{View}(\mathcal{DO}_j)$, contains the cumulative rewards s_i as well as the number of pulls n_i. It also contains every score $v_{j,t}$ and every selection bit $b_{j,t}$ for each time step $t \in [\![N]\!]$. In order to prove the above theorem, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$:

- *Simulator* $\mathcal{S}_0$. Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$, the number of arms K, the index j of the corrupted data owner $\mathcal{DO}_j$ chosen by the adversary $\mathcal{A}$, its real score $v_{j,t}$ and a bit associated to the value $M \stackrel{?}{=} j$, where the index M corresponds to the arm index having its best score. At the end of the execution, the simulator $\mathcal{S}_0$ outputs a $(K-1)$-sized list of tuples of the form $(s'_i, n'_i, v'_{i,t})$ whose first element is a random cumulative rewards, the second element is a random number of pulls and the third element is a random score.
  Our simulator $\mathcal{S}_0$ acts differently depending on the value of $M \stackrel{?}{=} j$. For the sake of clarity, let denote the value of $M \stackrel{?}{=} j$ by the bit b_{eq}. If the bit b_{eq} equals one, then the corrupted data owner $\mathcal{DO}_j$ expects to receive a positive selection bit. Hence,

given its score $v_{j,t}$, the simulator generates $K-1$ random scores $v'_{i,t}$ with the condition that every $v'_{i,t}$ is strictly lower than $v_{j,t}$. This ensures that the selected data owner will be the corrupted one. At the opposite, when b_{eq} is a negative bit, then the simulator generates $K-1$ random scores with the condition that at least one score $v'_{i,t}$ is strictly higher than $v_{j,t}$. Note that the selected arm here is chosen at random since the simulator does not have any information on the index M. This models an idealized situation in which a data owner that is not pulled has at most $\frac{1}{K-1}$ chance to correctly identify the chosen arm. The cumulative rewards and the number of pulls returned by the simulator are chosen at random.

- *Simulator* $\mathcal{S}_1$. Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K as well as the index j of the corrupted data owner, and produces $K-1$ simulated cumulative rewards $s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_K$ randomly chosen from an arbitrary space, say $\mathbb{N}$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the real experiment $\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}$, and the ideal world corresponding to the experiment $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}$.

Game G^0. This initial game corresponds to the $\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}$ experiment. Therefore, we have a perfect indistinguishability:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^1. This game works exactly as the game G^0 except that we replace the execution of the $\mathsf{MAB.PullArm}$ algorithm executed for all arms, by a single execution of $\mathsf{MAB.PullArm}$ for the data owner indexed by M, the index of the data owner having the highest score. As a result, the selection bits outputted by the arm selection protocol are now ignored. By correctness of the arm selection protocol $\mathsf{SelectArm}$, this modification does not affect the selected arm at each time step and hence does not affect the selection bits received by the corrupted data owner $\mathcal{DO}_j$ Hence, we have a perfect indistinguishability between G^0 and G^1:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^1_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^2. This game is the same as G^1, except that we introduce our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$. The simulator $\mathcal{S}_0$ is inputted at each time step $t \in [\![N]\!]$ with the current time t, the number of arms K, the index j of the corrupted data owner $\mathcal{DO}_j$, its real score $v_{j,t}$, as well as the bit b_{eq} associated to the value $M \stackrel{?}{=} j$. The simulator $\mathcal{S}_1$ is inputted with the number of arms K and the corrupted index j. For the moment, the output of $\mathcal{S}$ is not used. Therefore, the view of $\mathcal{DO}_j$ remains unchanged leading to a perfect indistinguishability between G^1 and G^2:

$$\Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^3. This game is the same as G^2 except that at each time $t \in [\![N]\!]$, given the index j of the corrupted data owner $\mathcal{DO}_j$, the real score $v_{i,t}$ for i different of j is replaced by the random score $v'_{i,t}$ computed by the simulator $\mathcal{S}_0$. We do the same replacement

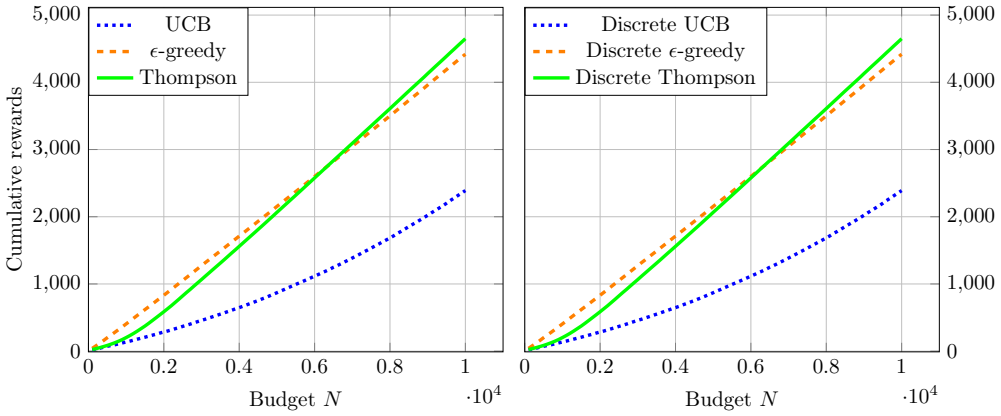


Figure 3.14: Obtained total cumulative rewards with the standard multi-armed bandits algorithm (left figure) and its discrete version (right figure), using UCB, ϵ-greedy and Thompson Sampling. We have set ϵ at 0.1. The two plots show identical results.

for the cumulative rewards and the number of pulls. When the chosen data owner is the corrupted data owner $\mathcal{DO}_j$, then the simulator $\mathcal{S}_0$ receives a positive selection bit. Otherwise, the arm selection selects randomly an arm among all honest data owner leading to a random arm selection. By construction of the experiment, the correctness is preserved thanks to the challenger running the appropriate arm pulling. As a result, the execution of the multi-armed bandits is still valid (even if random scores are provided), leading to a perfect indistinguishability between G^2 and G^3:

$$\Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^4. This game is the same as G^3 except that during the $\mathsf{SendRewards}$ protocol execution, the real cumulative rewards $s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_j$ are replaced by the random cumulative rewards $s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_j$ obtained at the output of the simulator $\mathcal{S}_1$. Observe that this modification does not affect the view of the corrupted data owner $\mathcal{DO}_j$ since it does contain any cumulative rewards s_i for i different of j. As a result, we obtain a perfect indistinguishability between G^3 and G^4:

$$\Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^4_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

And since G^4 corresponds to the ideal experiment $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}$, we observe a perfect indistinguishability between $\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}$:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}(\lambda, N, K) \to 1\right]$$

□

3.3.4 Empirical Study of Tango

Comparison Between Standard and Discrete Bandits

Our framework Tango works with *discrete* multi-armed bandits. However, we have focused our attention on the most standard multi-armed bandits such as UCB, working

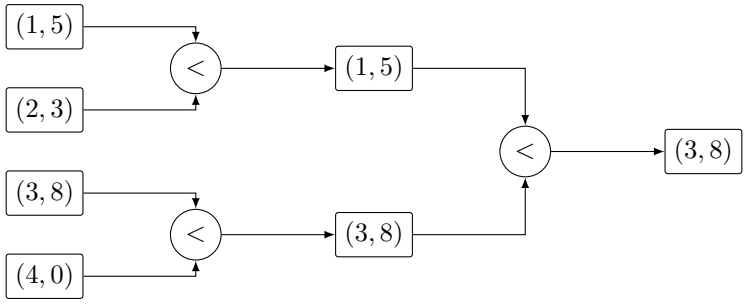


Figure 3.15: Graphical representation of the binary-tree comparison of four couples (i, v_i) where v_i is the score of the i-th party.

with floating-point values. To fill this gap, our approach consists of following the score computation of the standard multi-armed bandits returning a floating-point score, say v, by obtaining an approximation of v: This score is multiplied by a scale factor Δ, carefully chosen to fit the plaintext space of the FHE scheme. The scaled score Δv is rounded, resulting into the its integer version v', which equals $\lfloor \Delta v \rfloor$. In our implementation, we have set Δ as 10^{10} allowing us to consider 10 decimals. One may expect a decrease of the returned total cumulative rewards due to the approximation. To motivate the interest of our approach, we have studied the impact of the approximation on the returned total cumulative rewards: We have implemented the three multi-armed bandits algorithms instantiated in Tango, namely UCB, ϵ-greedy and Thompson Sampling, in their standard, floating-point scores version, but also in their discretized version. The resulting total cumulative rewards are presented in Figure 3.14. The used dataset, MovieLens [HK16], is composed of 100 arms and the highest reward probability is approximatively 0.53. Surprisingly, the returned total cumulative rewards between the standard multi-armed bandits algorithm and its approximated (*i.e.,* discrete) version are the *same*. While this fact strengthened our contribution, we stress however that this does not hold in general. Before to apply a multi-armed bandits algorithm in Tango, it is interesting to first analyze the impact of the approximation on the outputted total cumulative rewards. The reward maximization performance of Tango should be compared with respect to its *discretized* version of the instantiated multi-armed bandits algorithm, rather that its standard version that may diverge due to the approximation.

Execution Time of Tango

We propose an empirical study of the required execution time to run Tango. More precisely, we have implemented Tango in Rust. For the encryption schemes, we have used AES-256-GCM [AES07] as our secret-key encryption scheme, ElGamal [Elg85a] as our public-key encryption scheme, Paillier [Pai99] for our additively homomorphic encryption scheme and TFHE [CGGI19] as our fully-homomorphic encryption scheme. While the three first encryption schemes seem a natural choice, TFHE is more specifically designed for boolean arithmetic. Within the Concrete [Zam20] library implementing TFHE, an integer (up to 128 bits) is represented by its binary representation, all standard operations being performed over boolean gates. While this approach involves more gates, it has the advantage to be perfectly correct and particularly flexible, enjoying

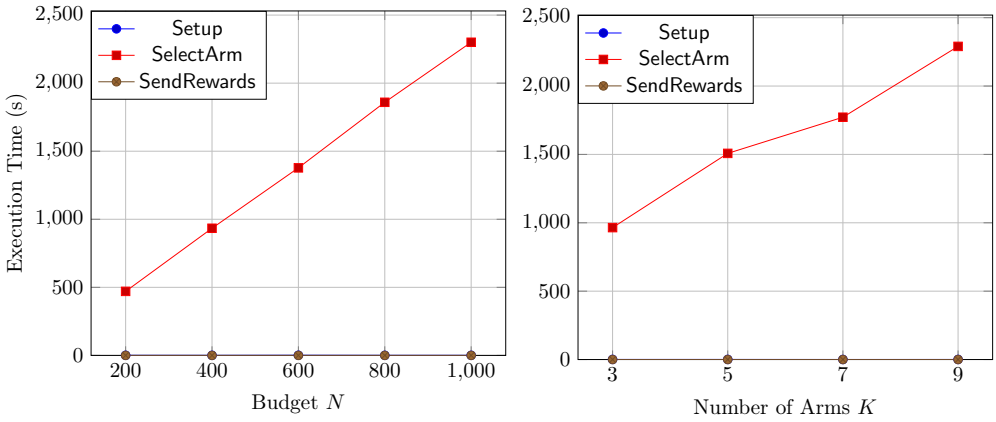


Figure 3.16: Execution time of TANGO. The presented execution times are the mean of 10 iterations. When varying the budget N, the number of arms K is set to 9. When varying the number of arms K, the budget N is set to 1000.

all the circuit optimization literature brings by the processor architecture. By nature, the boolean computation has the advantage to be correct, potential errors coming from the homomorphic encryption scheme only. It stands in contrast with other arithmetic fully homomorphic encryption schemes such as BGV [BGV12], supporting polynomial evaluation (involving addition, multiplication and constant exponentiation with coefficients over $\mathbb{Z}_p$ for some prime p). Indeed, to model comparison, it is usually common to rely on the sign function, which given a number x returns 1 if $0 \leq x$ and -1 otherwise. Badly, this function cannot be directly computed using only arithmetics. Instead, we approximate the sign function using a polynomial approximation [CKK20]. This polynomial approximation may produce invalid result, a problem that we do not have with the binary computation, even if it implies a larger amount of transmitted data (we have now one ciphertext for one bit).

We have implemented the homomorphic arg max function evaluation in order to optimize at the same time the execution time, and the noise consumption due to the homomorphic computation. More precisely, we have chosen to implement the arg max function following a binary tree structure, depicted in Figure 3.15. To return the index of maximal value in the list, we do the binary-tree comparison on the couples (i, v_i), where i corresponds to the index the data owner $\mathcal{DO}_i$ having sent v_i. Given two couples $(1, v_1)$ and $(2, v_2)$, the comparison results into the couple (M, v_M) where M equals 1 if v_1 is higher than v_2 and 2 otherwise. The score v_M equals the highest score. This binary-tree comparison allows at the same time to compare in parallel using multi-threading, but also to limit the noise consumption involved by the homomorphic computation, by comparing a score with another only $\log_2(K)$ times instead of $K-1$ times. The binary-tree comparison results into the couple (M, v_M). The next step consists of performing the equality test between the resulting encrypted best arm index M with every possible index i in $[\![K]\!]$. Remark that since the index i is known by the server, the equality test can be optimized using the plaintext-ciphertext equality supported by the Concrete library, which can be parallelized as well.

The resulting execution times of TANGO, depicted in Figure 3.16, has been performed on a server embedding an Intel Xeon Gold 6136 processor equipped of 16 cores

and cadenced at 3.00GHz, and having 50Gb of memory. The plotted execution times, being the mean of 10 iterations, show the execution time respectively of SelectArm and SendRewards. For completeness, we have studied the execution time when varying the budget N (between 200 and 1000 with a step of 200) and the number of arms K (between 3 and 9 with a step of 2). When the number of arms K does not vary, we observe a linear execution time with respect to the budget N, ranging from 500 seconds for a budget of 200 to 2300 seconds for a budget 1000. This linearity is explained by the fact that we are executing the arm selection protocol N times. The execution time for the setup and the rewards sending being independent of the budget N and negligible compared to the arm selection protocol, there are not relevant in the overall protocol execution time. When the budget N does not vary and when the number of arms K varies, we study the impact of adding an arm on the execution time. As shown in Figure 3.16, the execution time for the arm selection protocol when varying K is linear, ranging from 1000 seconds for 3 arms to 2300 seconds for 9 arms.

The execution of the arm selection involves homomorphic computation, largely impacting the overall performance of the arm selection protocol. For instance, assuming 9 arms and a budget N of 1000, more than 2300 seconds are required to spend all the budget. It wipes out any hope to apply Tango on real-time constrained applications. Although, Tango is still interesting for applications in which the result of the arm selection can be precomputed and later delivered. It occurs for instance with movie recommendation: Once the user has watched the suggested movie, the user provides a rating, which will be used to suggest new movies. Between two screenings, the federation server executes the arm selection and then obtains a new suggestion. Assuming 9 categories, since the arm selection protocol is executed once between two screenings, the user waits at most 2.3 seconds corresponding to a *single* execution of the SelectArm protocol obtained via our implementation.

Obtained Rewards using Tango

By construction of Tango, several multi-armed bandits strategies can be plugged and executed, enjoying the security offered by Tango. Using the same server presented above, we have executed the three discretized multi-armed bandits UCB, ϵ-greedy and Thompson Sampling algorithms in Tango and observed the obtained total cumulative rewards over the time. As seen previously, Tango is more suitable for a small numbers of arms, hence we have run Tango over 9 arms extracted from the MovieLens dataset, the highest reward probability being approximatively 0.34. In Figure 3.17, we observe that all the plugged algorithms produce rewards following closely the theoritical maximum (obtained by always pulling the best arm). For a budget N of 1000, our best strategy ϵ-greedy returns 301 rewards, followed by Thompson Sampling returning approximatively 288 rewards, followed by our last strategy, UCB, returning 220 rewards. These rewards performance are consistent with [CLMS22], where UCB needs a very large budget (around 10^5) to be more efficient than ϵ-greedy.

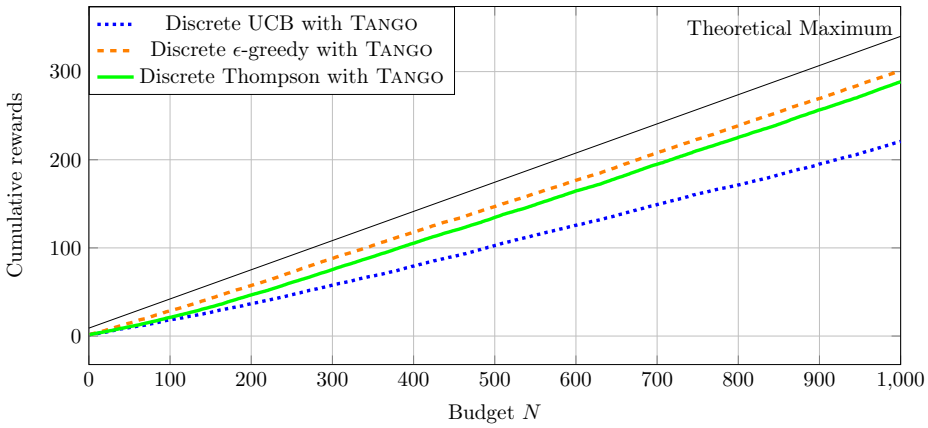


Figure 3.17: Obtained total cumulative rewards using 9 arms extracted from the MovieLens dataset over a budget N of 1000. We set ϵ at 0.1. The theoritical maximum corresponds to the situation in which the best arm is always chosen, the highest reward probability in this experiment is 0.34. Hence, for a budget N of 1000, the theoritical maximum rewards is $1000 \cdot 0.34 = 340$.

Obtained Rewards on Failure

Our protocol TANGO is said to be resistant to the scenario where one or more data owners are suddenly offline. This property is particularly interesting in federated learning and is empirically confirmed in TANGO. This property is achieved by sending at each time step $t \in [\![N]\!]$ the encryption of the cumulative rewards s_i of every (online) data owner $\mathcal{DO}_i$. As a result, if a data owner $\mathcal{DO}_i$ becomes offline at some point during the protocol, the cumulative rewards s_i remains known by the proxy node $\mathcal{P}$, rather than being lost since it is only known by $\mathcal{DO}_i$.

To empirically confirm the failure resistance of TANGO, several parameters have to be considered, including the used data set, the number of data owners K, the budget N, the multi-armed bandits algorithm, the frequency of failures and the data owners to disconnect. Due to the potentially large number of cases, we have focus our attention on the most impactful: We have chosen to apply the UCB multi-armed bandits algorithm, chosen for its deterministic score computation, on the MovieLens [HK16] data set equipped of 100 arms with a budget N of 1000. To emphasize the impact of the data owners being offline, we voluntary disconnect respectively 0%, 25%, 50%, 75% and 100% of the data owners having the highest reward probability, at time step 500 *i.e.,* at the middle of the multi-armed bandits execution. Hence, with 0% of the data owners being disconnected, we expect no difference between the total cumulative rewards with and without the saving of cumulative rewards. Let's move on the case where half of the data owners, having the highest reward probabilities, are considered offline. Remember that we disconnect the arms in the reward probability, in the decreasing order. Observe that since half of the arms including all the best arms are offline, the remaining budget, used for the next 500 time steps, are expected to provide less rewards.

The obtained total cumulative rewards with and without the saving of cumulative rewards is presented in Figure 3.18. All the results are the mean of 200 successive iterations. Note that all the computations performed are in clear, allowing us to increase the

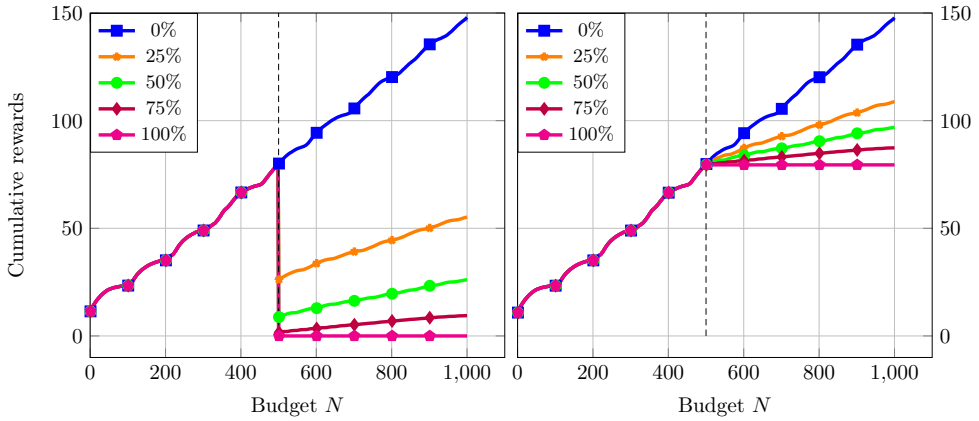


Figure 3.18: Obtained total cumulative rewards over the time using UCB when respectively 0%, 25%, 50%, 75% and all the data owners are going offline at the 500-th time step, without our rewards saving mechanism on the left and with our rewards saving mechanism on the right. The presented results are the mean of 200 iterations.

number of iterations considerably. The first figure, which depicts the multi-armed bandits execution without the saving of the cumulative rewards, shows a sudden decrease in the total cumulative rewards whereas the execution of the multi-armed bandits execution with the saving of the cumulative rewards does not. The sudden decrease reflects the way in which we have counted the total cumulative rewards. At each time step $t \in \mathbb{N}$, for the scenario without the saving of the cumulative rewards, we add every cumulative rewards s_i of all online data owners $\mathcal{DO}_i$. That is, an offline data owner $\mathcal{DO}_i$, more precisely its cumulative rewards s_i is no more counted in the current total cumulative rewards s after that $\mathcal{DO}_i$ lefts the protocol at the 500-th time step, explaining the sudden decrease of the total cumulative rewards s at the time step where the data owners disconnect. It correctly models the real scenario where the locally stored cumulative rewards s_i of an offline data owner $\mathcal{DO}_i$ being not accessible anymore. In contrast, with the saving of the cumulative rewards put in practice in Tango, the cumulative rewards s_i of the offline data owner $\mathcal{DO}_i$ is preserved in the memory of $\mathcal{P}$.

Observe that when all the data owners are offline, without the saving of the cumulative rewards, the returned total cumulative rewards is zero, whereas with the saving of the cumulative rewards, the returned total cumulative rewards corresponds to the cumulative rewards that have been received at the time step 499 (the 500-th time step being the step where all data owners are assumed to disconnect). After this time step t, since all data owners are offline, the UCB algorithm is not able to obtain new rewards since there is no more arm to pull, explaining the flat curve.

3.4 Salsa: A Scalable Secure Federated Bandits

As we have observed, Tango offers correctness and security but suffers from a high latency during the arm selection due to the homomorphic operations. Hence, Tango is not suitable for real-time applications whose delay to obtain the selection bits is reduced. For these applications, we propose a new *scalable* secure multi-armed bandits protocol, this time based on secure three-party computations.

Overview of Salsa. Compared to Tango respecting the architecture and the philosophy of Samba, Salsa drastically moves away to focus on secure multi-party computations: Rather to rely on two servers, one acting as a proxy for the other one, Salsa relies on two servers denoted $\mathcal{C}_0$ and $\mathcal{C}_1$ to perform computations over *shared* scores, in the vein of the ABY framework [DSZ15]. In a nutshell, at the beginning of the arm selection protocol execution at time step $t \in [\![N]\!]$, every data owner $\mathcal{DO}_i$ computes the two shares of its score $v_{i,t}$, denoted respectively $\langle v_{i,t}\rangle^0$ and $\langle v_{i,t}\rangle^1$. The first share $\langle v_{i,t}\rangle^0$ is sent to $\mathcal{C}_0$ while the second share $\langle v_{i,t}\rangle^1$ is sent to $\mathcal{C}_1$. Using secure two-party computation techniques each server $\mathcal{C}_j$ ends with the share of every selection bit $\langle b_{i,t}\rangle^j$ for every data owner $\mathcal{DO}_i$. As we will see later in the detailed description of the arm selection protocol, the two servers are supported by an additional server acting as a trusted random provider, providing sharings of particular randoms, required by the servers $\mathcal{C}_0$ and $\mathcal{C}_1$ to evaluate the arm selection. Given the two shares $\langle b_{i,t}\rangle^0$ and $\langle b_{i,t}\rangle^1$, the data owner $\mathcal{DO}_i$ reconstructs $b_{i,t}$ efficiently before to execute the arm pulling function MAB.PullArm.

At the end of protocol, every data owner shares its cumulative rewards s_i with the servers, via the transmission of an additional share $\langle s_i\rangle^0$ for the server $\mathcal{C}_0$ and the share $\langle s_i\rangle^1$ for the server $\mathcal{C}_1$. Using additive property of secret sharing, from the shares $\langle s_1\rangle^j, \ldots, \langle s_K\rangle^j$ the server $\mathcal{C}_j$ is able to derive the share $\langle s\rangle^j$. Then, the shares $\langle s\rangle^0$ and $\langle s\rangle^1$ computed respectively by the server $\mathcal{C}_0$ and $\mathcal{C}_1$ are transmitted to the user $\mathcal{U}$, which given these two shares, can efficiently reconstruct the total cumulative rewards s.

Setup of Salsa

Compared to Tango which is built based on cryptographic primitives, Salsa does not necessitate keys, thanks to the secure multi-party computation paradigm. Instead, the privacy of the data is guaranteed using information-theoretic argument, in the spirit of the one-time pad security. Indeed, the arithmetic sharing [DSZ15] that we use in Salsa, consists for a value v if defining two shares $\langle v\rangle^0$ and $\langle v\rangle^1$ where $\langle v\rangle^0$ equals r and $\langle v\rangle^1$ equals $v - r$. Observe that given only one share, it is theoretically infeasible to recover v. Indeed, since v is independent of the first share $\langle v\rangle^0$ consisting of r, and since the second share $v - r$ consists of a one-time pad, it is theoretically infeasible to recover v. It's interesting to note that, in the same way as Tango, we assume here that all communications are authenticated and secure, a reasonable hypothesis since the identity of the servers is publicly known, which is sufficient to establish an authenticated secure channel. This hypothesis is used to prevent a party to observe a communication between two other parties.

Arm Selection of Salsa (Figure 3.19)

In Salsa, at time step $t \in [\![N]\!]$, the arm selection protocol SelectArm depicted in Figure 3.19 starts with every data owner being inputted with a score $v_{i,t}$, its cumulative rewards s_i and the number of pulls n_i. Recall that Salsa, as well as Tango, are supposed to work for *discretized* multi-armed bandits algorithm. For this reason, the score $v_{i,t}$ can be represented using its binary representation over l bits. Let us denote the binary representation of $v_{i,t}$ as $v_{i,t}[1], \ldots, v_{i,t}[l]$ where $v_{i,t}[p]$ denotes the p-th bit of $v_{i,t}$.

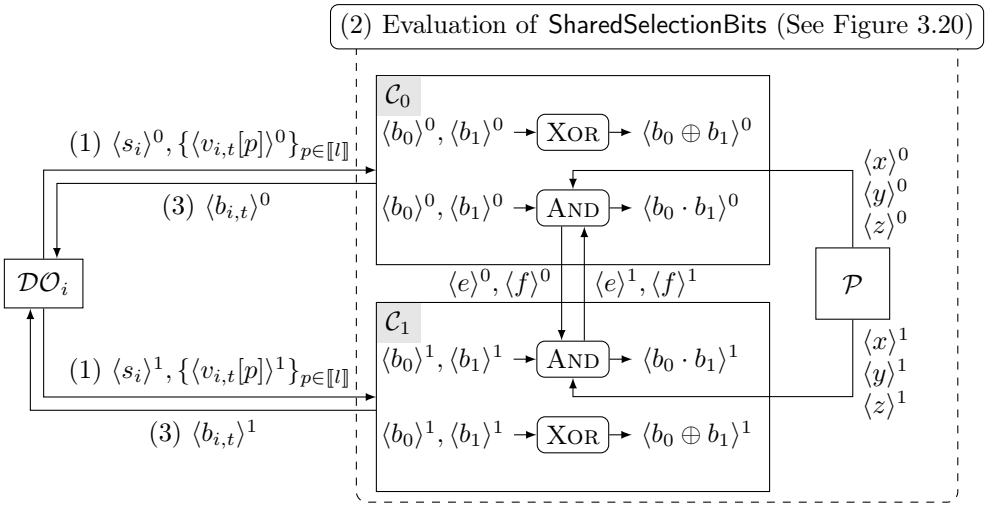


Figure 3.19: Representation of an arm selection protocol execution in Salsa. At step (1), every data owner distributes the share of its binary-decomposed score on which the servers $\mathcal{C}_0$ and $\mathcal{C}_1$ computes the shares of the selection bits, sent back at step (3). All communications between the parties are assumed to be authenticated and secure.

Each data owner $\mathcal{DO}_i$ starts the arm selection protocol by computing two shares of each bit $v_{i,t}[p]$, denoted respectively $\langle v_{i,t}[p]\rangle^0$ and $\langle v_{i,t}[p]\rangle^1$. These shares working on bits are called *binary shares* and follows the same spirit of arithmetic share: The two shares $\langle b\rangle^0$ and $\langle b\rangle^1$ of a bit b are computed respectively as r and $b \oplus r$ for a randomly chosen bit r. Following this sharing method, every bit $v_{i,t}[p]$ is shared as $\langle v_{i,t}[p]\rangle^0 = r$ and $\langle v_{i,t}[p]\rangle^1 = v_{i,t}[p] \oplus r$ for a random bit r. Then, the shares $\langle v_{i,t}[1]\rangle^j, \ldots, \langle v_{i,t}[l]\rangle^j$ are sent the server $\mathcal{C}_j$. Given all these binary shares, the servers $\mathcal{C}_0$ and $\mathcal{C}_1$ are able to evaluate the arg max circuit resulting into the best arm index M, then compared with every index $i \in [\![K]\!]$ resulting into K selection bits. This procedure described in Figure 3.20 has access to two functions SharedArgmax and SharedEq, depicted respectively in Figure 3.23 and Figure 3.21, executing respectively the arg max function and the equality test function using only Xor and And (from which Or and Not binary gates can be derived). As we will see in a moment, these functions are not difficult, the only particularity being that Xor and And binary gates are executed over shares instead of bits, following the computation sharing techniques from [DSZ15].

SharedSelectionBits$_{j,t}(\{\langle v_{1,t}[p]\rangle^j\}_{p\in[\![l]\!]}, \ldots, \{\langle v_{K,t}[p]\rangle^j\}_{p\in[\![l]\!]})$

Subroutines: SharedArgmax (See Figure 3.23), SharedEq (See Figure 3.21)

Constants: $\langle 1[1]\rangle^j, \ldots, \langle 1[l]\rangle^j, \ldots, \langle K[1]\rangle^j, \ldots, \langle K[l]\rangle^j$ */* Shared constant indexes */*

$\langle M[1]\rangle^j, \ldots, \langle M[l]\rangle^j \leftarrow$ SharedArgmax$_{j,t}(\{\langle v_{1,t}[p]\rangle^j\}_{p\in[\![l]\!]}, \ldots, \{\langle v_{K,t}[p]\rangle^j\}_{p\in[\![l]\!]})$

for $\langle i[1]\rangle^j, \ldots, \langle i[l]\rangle^j \in [\![K]\!]$

  $\langle b_{i,t}\rangle^j \leftarrow$ SharedEq$_{j,t}(\{\langle M[p]\rangle^j\}_{p\in[\![l]\!]}, \{\langle i[p]\rangle^j\}_{p\in[\![l]\!]})$

return $\langle b_{1,t}\rangle^j, \ldots, \langle b_{K,t}\rangle^j$

Figure 3.20: The selection bits generation function inputted with binary-decomposed shared scores.

Before introducing these algorithms, we first describe the following SharedIfThenElse and SharedEq subroutines depicted in Figure 3.21. The SharedIfThenElse simulates the behaviour of a conditional branching written as `if b then { x } else { y }` where b is a bit respresenting a condition and where x and y are l-bits values. Specialized to handle bit sharings, this function expects as an input $2l+1$ sharings: A bit sharing for b and $2l$ bit sharings for x and y. The function returns l binary sharings corresponding to the bits of x if b equals true, otherwise it returns l bit sharings corresponding to the bits of y. The SharedEq function expects $2l$ bit sharings x and y and returns a single bit sharing of 1 if x equals y, otherwise it returns a single bit sharing of 0.

$\mathsf{SharedIfThenElse}_{j,t}(\langle b\rangle^j, \langle x[1]\rangle^j, \ldots, \langle x[l]\rangle^j, \langle y[1]\rangle^j, \ldots, \langle y[l]\rangle^j)$

for $p \in [\![l]\!]$: $\langle r[p]\rangle^j \leftarrow \textsc{Or}(\textsc{And}(\langle b\rangle^j, \langle x[p]\rangle^j), \textsc{And}(\textsc{Not}(\langle b\rangle^j), \langle y[p]\rangle^j))$

return $\langle r[1]\rangle^j, \ldots, \langle r[l]\rangle^j$

$\mathsf{SharedEq}_{j,t}(\langle x[1]\rangle^j, \ldots, \langle x[l]\rangle^j, \langle y[1]\rangle^j, \ldots, \langle y[p]\rangle^j)$

return $\textsc{Not}(\textsc{Or}(\textsc{Xor}(\langle x[1]\rangle^j, \langle y[1]\rangle^j), \ldots, \textsc{Xor}(\langle x[p]\rangle^j, \langle y[p]\rangle^j)))$

Figure 3.21: Instantiation of the SharedIfThenElse and SharedEq functions.

We provide an additional function SharedLessThan described in Figure 3.22 which simulates the behaviour of `a < b` for two l-bit values. More precisely, this function takes as an input $2l$ binary sharings corresponding to the bits of x and y, and returns a single bit sharing of 1 if x is strictly less than y, otherwise it returns a single bit sharing of 0. Intuitively, the SharedLessThan function identifies a position $p \in [\![l]\!]$ for which $x[i]$ and $y[i]$ are different. In particular, if $y[i]$ equals one and no higher bits differs between x and y, then we directly conclude that y is stricly higher than x. A sharing $\langle f\rangle^j$ is used to stop the modification of the result sharing $\langle r\rangle^j$ when a difference has been detected. Observe that the two variables r and f are not modified only in case where a difference has been found or if there is no difference between x and y. This function assumes the sharing of the two constants 0 and 1, which is generated once and reuse over all the executions.

$\mathsf{SharedLessThan}_{j,t}(\langle x[1]\rangle^j, \ldots, \langle x[l]\rangle^j, \langle y[1]\rangle^j, \ldots, \langle y[l]\rangle^j)$

Constants: $\langle 0\rangle^j, \langle 1\rangle^j$

$\langle f\rangle^j \leftarrow \langle 1\rangle^j, \langle r\rangle^j \leftarrow \langle 0\rangle^j$

for $p \in [\![l, \ldots, 1]\!]$

- $\langle d\rangle^j \leftarrow \textsc{Xor}(\langle x[p]\rangle^j, \langle y[p]\rangle^j)$
- $\langle b\rangle^j \leftarrow \textsc{And}(\langle f\rangle^j, \langle d\rangle^j)$
- $\langle \neg b\rangle^j \leftarrow \textsc{Not}(\langle b\rangle^j)$
- $\langle r\rangle^j \leftarrow \textsc{Or}(\textsc{And}(\langle b\rangle^j, \textsc{And}(\langle d\rangle^j, \langle y[p]\rangle^j)), \textsc{And}(\langle \neg b\rangle^j, \langle r\rangle^j)$
- $\langle f\rangle^j \leftarrow \textsc{Or}(\textsc{And}(\langle b\rangle^j, \textsc{Not}(\langle f\rangle^j)), \textsc{And}(\langle \neg b\rangle^j, \langle f\rangle^j))$

return $\langle r\rangle^j$

Figure 3.22: Instantiation of the SharedLessThan function.

The SharedArgmax function depicted in Figure 3.23 takes as an input K scores where each score is decomposed of l bit sharings. Our instantiation directly relies on the previously introduced functions and follows the natural arg max definition.

$\mathsf{SharedArgmax}_{j,t}(\{\langle v_{1,t}[p]\rangle^j\}_{p\in[\![l]\!]}, \ldots, \{\langle v_{K,t}[p]\rangle^j\}_{p\in[\![l]\!]})$

Subroutines: SharedLessThan (See Figure 3.22), SharedIfThenElse (See Figure 3.21)

Constants: $\langle 1[1]\rangle^j, \ldots, \langle 1[l]\rangle^j, \ldots, \langle K[1]\rangle^j, \ldots, \langle K[l]\rangle^j$

$\langle v[1]\rangle^j, \ldots, \langle v[l]\rangle^j \leftarrow \langle v_{1,t}[1]\rangle^j, \ldots, \langle v_{1,t}[l]\rangle^j$

$\langle M[1]\rangle^j, \ldots, \langle M[l]\rangle^j \leftarrow \langle 1[1]\rangle^j, \ldots, \langle 1[l]\rangle^j$ */* Constant shares of index 1 */*

for $\langle i[1]\rangle^j, \ldots, \langle i[l]\rangle^j \in [\![2, K]\!]$ */* Constant shares of index i */*

  $\langle b\rangle^j \leftarrow \mathsf{SharedLessThan}(\langle v[1]\rangle^j, \ldots, \langle v[l]\rangle^j, \langle v_{i,t}[1]\rangle^j, \ldots, \langle v_{i,t}[l]\rangle^j)$

  $\langle v[1]\rangle^j, \ldots, \langle v[l]\rangle^j \leftarrow \mathsf{SharedIfThenElse}(\langle b\rangle^j, \langle v_{i,t}[1]\rangle^j, \ldots, \langle v_{i,t}[l]\rangle^j, \langle v[1]\rangle^j, \ldots, \langle v[l]\rangle^j)$

  $\langle M[1]\rangle^j, \ldots, \langle M[l]\rangle^j \leftarrow \mathsf{SharedIfThenElse}(\langle b\rangle^j, \langle i[1]\rangle^j, \ldots, \langle i[l]\rangle^j, \langle M[1]\rangle^j, \ldots, \langle M[l]\rangle^j)$

return $\langle M[1]\rangle^j, \ldots, \langle M[l]\rangle^j$

Figure 3.23: Instantiation of the SharedArgmax function.

An interesting fact about binary sharing is that a XOR gate can be computed *locally* without any interaction between the two servers. Suppose two bits b_0 and b_1 whose respective shares are defined as follows: Let denote the shares of b_0 as $\langle b_0\rangle^0 = r_0$, $\langle b_0\rangle^1 = b_0 \oplus r_0$ and let denote the shares of b_1 as $\langle b_1\rangle^1 = r_1$, $\langle b_1\rangle^1 = b_1 \oplus r_1$. To obtain the output of the XOR gate denoted $b_2 = b_0 \oplus b_1$, a server $\mathcal{C}_j$ simply evaluates $\langle b_2\rangle^j$ as $\langle b_0\rangle^j \oplus \langle b_1\rangle^j$, leading to the shares $\langle b_2\rangle^0 = r_0 \oplus r_1$ and $\langle b_2\rangle^1 = b_0 \oplus b_1 \oplus r_0 \oplus r_1$. Hence, if we rewrite $r_0 \oplus r_1$ by a random bit r', then we have $\langle b_2\rangle^0 = r'$ and $\langle b_2\rangle^1 = b_0 \oplus b_1 \oplus r'$ which constitutes a valid sharing of b_2, without implying any communication between the two servers.

This stands in contrast with the AND gate which requires an interaction between the two servers. For the sake of clarity, suppose the shares $\langle b_0\rangle^0 = r_0$, $\langle b_0\rangle^1 = b_0 \oplus r_0$ and $\langle b_1\rangle^1 = r_1$, $\langle b_1\rangle^1 = b_1 \oplus r_1$ of two bits b_0 and b_1. To evaluate the AND gate, the two servers are required to have the so-called *correlated randomness*[1]. This correlated randomness takes the form of a triplet (x, y, z) where x and y are two random bits and z equals $x \wedge y$. Each server is assumed to have a share of each of these bits. Hence, the server $\mathcal{C}_j$ is supposed to have the triplet of shares $(\langle x\rangle^j, \langle y\rangle^j, \langle z\rangle^j)$. Assuming this triplet, the server $\mathcal{C}_j$ computes $\langle e\rangle^j$ as $\langle b_0\rangle^j \oplus \langle x\rangle^j$ and the share $\langle f\rangle^j$ as $\langle b_1\rangle^j \oplus \langle y\rangle^j$. At this point, each server $\mathcal{C}_j$ communicates its shares $\langle e\rangle^j$ and $\langle f\rangle^j$ with the other server $\mathcal{C}_{1-j}$ and proceeds to the reconstruction of e and f. Then, each server $\mathcal{C}_j$ computes the resulting share $\langle b_2\rangle^j$ of the AND gate as $j \cdot e \cdot f \oplus f \cdot \langle y\rangle^j \oplus e \cdot \langle x\rangle^j \oplus \langle z\rangle^j$ [DSZ15]. As we have observed, the evaluation of the AND gate requires a bilateral communication between the two servers. Hence, the efficiency of the AND gate highly depends on the efficiency of the network latency between the two servers $\mathcal{C}_0$ and $\mathcal{C}_1$. Hopefully, in our architecture, the two servers are publicly identified and does not change over the time. Hence, it is realistic to assume a low-latency communication channel between these two servers. Furthermore, the evaluation of a single AND gate involves the communication of 4 bits, one bit for each share of e and f.

[1] In the literature, it is sometimes referred as a multiple triplet or a Beaver's triplet.

To evaluate an AND gate while ensuring the privacy of the computation, a correlated randomness triplet can be used once. However, in our best arm identification problem, the evaluation of an arg max circuit over ten 64-bits values followed by ten 64-bits equality test involves the evaluation of more than 9500 AND gates. Even more, a budget N of 1000 requires to execute the arm selection protocol 1000 times leading to the evaluation of more than 9 millions of AND gates. Therefore, the efficiency of th correlated randomness generation is crucial to limit as most as possible the computation overhead. Based on [DSZ15], it is possible to obtain a correlated randomness triplet via additive homomorphic encryption such as the Paillier cryptosystem [Pai99]. However, it is recommended to perform these computations during a setup phase when the input is unknown, to avoid the computation of these triplets when receiving the input. However, due to the potentially high numbers of triplets to be generated, this solution is not ideal. Instead, we have chosen to include in our architecture an additional server, referred as $\mathcal{P}$ (for Provider), acting as a trusted correlated randomness provider. The provider $\mathcal{P}$ generates two random bits x and y, and computes the sharing $\langle x\rangle^0$ and $\langle x\rangle^1$ and the sharings $\langle y\rangle^0$ and $\langle y\rangle^1$. In addition, it computes $z = x \oplus y$ as well as the associated shares $\langle z\rangle^0$ and $\langle z\rangle^1$. Finally, it provides the triplet $(\langle x\rangle^j, \langle y\rangle^j, \langle z\rangle^j)$ to the server $\mathcal{C}_j$. With our approach, we avoid additive homomorphic encryption while transmitting only 6 bits for a single AND gate, 3 bits by server. Observe that to evaluate 9 millions of AND gates, approximately 3.4 megabytes have to be transmitted between the trusted provider $\mathcal{P}$ and a server $\mathcal{C}_j$ using a low-latency communication network.

Given the XOR and AND gates, the servers $\mathcal{C}_0$ and $\mathcal{C}_1$ are able to evaluate the arg max function using shares on the binary decomposition of the scores, leading to the shared best arm index M, followed by an equality test between M and every shared index $i \in [\![K]\!]$. At the end of the evaluation, the server $\mathcal{C}_j$ obtains K shares $\langle b_{1,t}\rangle^j, \ldots, \langle b_{K,t}\rangle^j$. Each share $\langle b_{i,t}\rangle^j$ is sent back to the data owner $\mathcal{DO}_i$, leading for $\mathcal{DO}_i$ to have the two shares $\langle b_{i,t}\rangle^0$ and $\langle b_{i,t}\rangle^1$. Using the binary sharing reconstruction method, $\mathcal{DO}_i$ computes $\langle b_{i,t}\rangle^0 \oplus \langle b_{i,t}\rangle^1 = r \oplus b_{i,t} \oplus r$ leading to the reconstruction of $b_{i,t}$. Given its selection bit, every data owner $\mathcal{DO}_i$ pulls itself as specified by our generic multi-armed bandits model.

We do not have mentionned it for brevity, but along the shares of the score's bits, every data owner computes and sends the arithmetic share $\langle s_i\rangle^j$ to the server $\mathcal{C}_j$, where $\langle s_i\rangle^0$ equals r_i and $\langle s_i\rangle^1$ equals $s_i - r_i$ for some random r_i. This constitutes the heart of our rewards saving mechanism, allowing our protocol SALSA to be secure against data owners failures.

Total Cumulative Rewards Sending Protocol of SALSA (Figure 3.24)

At the end of the protocol execution, the user $\mathcal{U}$ expects to have the total cumulative rewards s corresponding to the sum of all the cumulative rewards s_i. For clarity, suppose that all data owners are online. The rewards sending protocol starts with every data owner computing and sending the arithmetic share $\langle s_i\rangle^j$ to the server $\mathcal{C}_j$, where $\langle s_i\rangle^0$ equals r_i and $\langle s_i\rangle^1$ equals $s_i - r_i$ for some random r_i. As a result, the first server $\mathcal{C}_0$ obtains the shares $\langle s_1\rangle^0, \ldots, \langle s_K\rangle^0$ corresponding to the randoms $r_1, \ldots, r_K$, whereas the second server $\mathcal{C}_1$ obtains the shares $\langle s_1\rangle^1, \ldots, \langle s_K\rangle^1$ correspond-

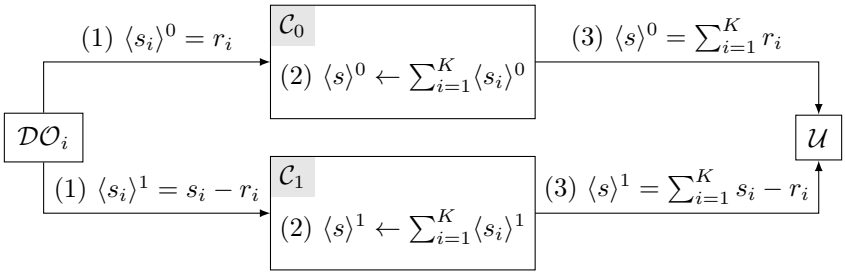


Figure 3.24: Representation of the total rewards sending protocol execution in Salsa. All communications are assumed to be authenticated and secure.

ing to $s_1 - r_1, \ldots, s_K - r_K$. The shares $\langle s \rangle^0$ and $\langle s \rangle^1$ are efficiently computable as $\langle s \rangle^0 = \sum_{i=1}^K r_i$ and $\langle s \rangle^1 = \sum_{i=1}^K s_i - \sum_{i=1}^K r_i$. These two resulting shares are then forwarded to the user $\mathcal{U}$ which recovers the total cumulative rewards s by computing $\langle s \rangle^0 + \langle s \rangle^1$. Thanks to the previously sent shares $\langle s \rangle^j$, the shares of a data owner having left the protocol before the end can be replaced by the shares receiving during the arm selection protocol.

3.4.1 Correctness of Salsa

By construction, the correctness of Salsa heavily holds on the correctness of the arithmetic and binary sharing techniques. We first focus on the correctness of the arithmetic sharing by supposing two values a and b and their shares $\langle a \rangle^0 = r_a, \langle a \rangle^1 = a - r_a$ of a and the shares $\langle b \rangle^0 = r_b, \langle b \rangle^1 = b - r_b$. The reconstruction of the sharings of a, denoted $\langle a \rangle^0 + \langle a \rangle^1$, is to recover a and hence validates the correctness of the reconstruction. The addition $c = a + b$ can be computed non-interactively with $\langle c \rangle^0 = \langle a \rangle^0 + \langle b \rangle^0 = r_a + r_b$ and $\langle c \rangle^1 = \langle a \rangle^1 + \langle b \rangle^1 = a + b - r_a - r_b$. The reconstruction of c performed via the addition of $\langle c \rangle^0$ and $\langle c \rangle^1$ results into $a + b$ which confirms the correctness of the arythmetic sharings. Since the total cumulative rewards sending protocol is only based on reconstruction and shared addition to share the total cumulative rewards s with the user $\mathcal{U}$, the correctness of this part of the protocol follows.

The arm selection protocol, in contrast, relies on binary shares. The shares $\langle b \rangle^0 = r$ and $\langle b \rangle^1 = b \oplus r$ can be efficiently reconstructed as $\langle b \rangle^0 \oplus \langle b \rangle^1$ in order to recover the bit b. Binary sharings support the evaluation of the Xor and And gates. Since Xor is by essence an addition over $\mathbb{F}_2$, the correctness of the Xor gate evaluation holds under the same argument for the addition using arithmetic sharings. The And gate requires more explanation since it is involving a correlated randomness. Suppose b_0 and b_1 two bits, as well as two random bits x and y. Let denote by z the evaluation of the And gate over x and y, written as $x \cdot y$. The And gate evaluation assumes that the server $\mathcal{C}_j$ has access to the correlated randomness $(\langle x \rangle^j, \langle y \rangle^j, \langle z \rangle^j)$, and involves the reconstruction of e and f defined respectively as $b_0 \oplus x$ and $b_1 \oplus y$ [DSZ15]. Assuming b_2 being the resulting shared bit obtain after evaluation of the And gate over the shared bits b_0 and b_1, the

reconstruction of the sharing $\langle b_2 \rangle^0$ and $\langle b_2 \rangle^1$ of b_2 can be obtained as follows:

$$\begin{aligned}
\langle b_2 \rangle^0 \oplus \langle b_2 \rangle^1 &= (f \cdot \langle x \rangle^0 \oplus e \cdot \langle y \rangle^0 \oplus \langle z \rangle^0) \oplus (e \cdot f \oplus f \cdot \langle x \rangle^1 \oplus e \cdot \langle y \rangle^1 \oplus \langle z \rangle^1) \\
&= f \cdot (\langle x \rangle^0 \oplus \langle x \rangle^1) \oplus e \cdot (\langle y \rangle^0 \oplus \langle y \rangle^1) \oplus e \cdot f \oplus \langle z \rangle^0 \oplus \langle z \rangle^1 \\
&= f \cdot x \oplus e \cdot y \oplus e \cdot f \oplus z \\
&= (b_1 \oplus y) \cdot x \oplus (b_0 \oplus x) \cdot y \oplus (b_0 \oplus x)(b_1 \oplus y) \oplus x \cdot y \\
&= (b_1 \oplus y) \cdot (x \oplus b_0 \oplus x) \oplus (b_0 \oplus x) \cdot y \oplus x \cdot y \\
&= (b_1 \oplus y) \cdot b_0 \oplus (b_0 \oplus x) \cdot y \oplus x \cdot y \\
&= b_0 \cdot b_1 \oplus y \cdot b_0 \oplus y \cdot b_0 \oplus x \cdot y \oplus x \cdot y \\
&= b_0 \cdot b_1
\end{aligned}$$

Since $b_0 \cdot b_1$ equals one if and only if both bits equal one, the reconstruction of the outputted sharings leads to a valid result. Furthemore, since all the arm selection computations are based on these XOR and AND gates, the correctness of binary computations and by extension our SharedSelectionBits algorithm, are ensured.

3.4.2 Security of SALSA

In contrast with TANGO whose security is based on computational hypothesis, SALSA relies on information theoretic arguments thanks to the secure two-party computations. By design, both binary and arithmetic shares $\langle v \rangle^0$ and $\langle v \rangle^1$ prevents one to learn v given only one share. Before providing the security proof of SALSA, we give the overall intuition on the privacy-preserving property of sharings.

Security of Arithmetic Shares

In SALSA, arithmetic shares are used during the SelectArm protocol to share the cumulative rewards s_i and during the SendRewards protocol in which addition over sharings is performed. Let focus on the SendRewards protocol being the most insightful, in which the first server $\mathcal{C}_0$ has access to first share $\langle s_i \rangle^0$ which equals r_i, and where the second server $\mathcal{C}_1$ has access to the second share $\langle s_i \rangle^1$ which equals $s_i - r_i$. Whereas it is clear that $\mathcal{C}_0$ has no particular advantage to recover s_i from r_i, since r_i is randomly chosen by $\mathcal{DO}_i$, by the one-time pad security, the second server $\mathcal{C}_1$ has no advantage to recover s_i from $s_i - r_i$. And since every computation performed by the servers $\mathcal{C}_0$ and $\mathcal{C}_1$ are limited to use the additive property of arithmetic shares, implying no communication, the server $\mathcal{C}_j$ has a statistical advantage to distinguish between a share $\langle s_i \rangle^j$ real cumulative rewards s_i and a share $\langle s'_i \rangle^j$ of a random cumulative rewards s'_i.

Security of Binary Shares

The binary shares are used in SALSA during the arm selection protocol to obtain the selection bits from the (shared) scores provided by the data owners. The selection bits generation circuit takes as an input the binary representation of the scores, where each bit of $v_{i,t}$ is shared among the two servers $\mathcal{C}_0$ and $\mathcal{C}_1$, using the binary sharing technique. Similarity to arithmetic shares, the first share $\langle b \rangle^0$ of a bit b is defined by a random bit

r chosen by the data owner $\mathcal{DO}_i$, and the second share $\langle b \rangle^1$ equals $b \oplus r$. As we have observed, the construction of the XOR gate does not involve an interaction between the two servers. However, due to the complexity of the circuit we are trying to evaluate to obtain the selection bits, we have also to evaluate the AND gate which requires an exchange of shares, implying the input bits and the intermediate data. Indeed, each server $\mathcal{C}_j$ is expected the shares $\langle e \rangle^{1-j}$ and $\langle f \rangle^{1-j}$ of the other server to reconstruct respectively e and f, which by definition equal respectively $b_0 \oplus x$ and $b_1 \oplus y$. The value of x and y are randomly and privately chosen in the protocol by the trusted provider $\mathcal{P}$ which is assumed to not collude with the other servers and with a data owner. These values e and f are then used to construct the resulting shares, which once combined, recover the output of the AND gate. Since the value of x and y are never revealed and randomly chosen by the trusted random provider $\mathcal{P}$ then under the one-time pad security, both $\mathcal{C}_0$ and $\mathcal{C}_1$ cannot distinguish between the p-th bit of a score and a random bit [DSZ15].

Security Proofs for the Server $\mathcal{C}_j$ and the Trusted Provider $\mathcal{P}$

We first focus on the case of the controller $\mathcal{C}$, being splitted into three distinct non-colluding servers: Two servers $\mathcal{C}_0$ and $\mathcal{C}_1$ performing the computations and a trusted random provider $\mathcal{P}$. To prove that the controller $\mathcal{C}$ does not learn any information on any score $v_{i,t}$, on any cumulative rewards n_i, on any number of pulls for a data owner $\mathcal{DO}_i$ at some time step t, or on the total cumulative rewards s, we proceed in two lemmas. The first lemma considers the case of the (honest-but-curious) corruption of $\mathcal{C}_j$ for j in $\{0, 1\}$, whereas the second lemma considers the case of a the (honest-but-curious) corruption of the trusted provider $\mathcal{P}$.

For clarity, we denote by RealC_j (respectively RealP) the real experiment being the same of the real experiment RealC_j, except that the adversary corrupts $\mathcal{C}_j$ (respectively $\mathcal{P}$). Similarly, we denote by IdealC_j (respectively IdealP) the ideal experiment being the ideal experiment IdealC_j, except that the adversary corrupts only $\mathcal{C}_j$ (respectively $\mathcal{P}$). To prove the simulation security SimC of SALSA, it suffices to prove these two lemmas.

Lemma 3. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, for every security parameter $\lambda \in \mathbb{N}$, every polynomial-time adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that SALSA is statistically SimC_j-secure:

$$\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealC}_j}(1^\lambda, N, K) \to 1\right] \overset{s}{\approx} \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealC}_j}(1^\lambda, N, K) \to 1\right]$$

Proof. The view of $\mathcal{C}_j$, denoted $\mathsf{View}(\mathcal{C}_j)$ is composed of the shares $\langle v_{i,t}[p] \rangle^j$ for every $p \in [\![l]\!]$, $\langle b_{i,t} \rangle^j$ and $\langle s_i \rangle^j$ for every $i \in [\![K]\!]$ and every time step $t \in [\![t]\!]$, as well as the share $\langle s \rangle^j$. It also includes all intermediate shares obtained during the $\mathsf{SelectArm}$ execution protocol. In order to prove the above lemma, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ used in the game $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealC}_j}$:

- *Simulator* $\mathcal{S}_0$. Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$ and the number of arms K. At the end of the execution, for some time step $t \in [\![K]\!]$, the simulator $\mathcal{S}_0$ is required to output a K-sized list of tuples $(s'_i, n'_i, v'_{i,t})$

for $i \in [\![K]\!]$ where s'_i is a random cumulative rewards, n'_i is a random number of pulls and $v'_{i,t}$ is a random score.
The simulator $\mathcal{S}_0$ starts by randomly chosen K scores $v'_{1,t}, \ldots, v'_{K,t}$, K random cumulative rewards $s'_1, \ldots, s'_K$ and K random number of pulls $n'_1, \ldots, n'_K$. Finally, it outputs the list of tuples $(s'_i, n'_i, v'_{i,t})$ for every $i \in [\![K]\!]$.

- *Simulator* $\mathcal{S}_1$. Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K, and produces the simulated cumulative rewards s'_i randomly chosen from an arbitrary space, say $\mathbb{N}$, for each arm $i \in [\![K]\!]$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealC}_j}$, and the ideal world corresponding to the experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealC}_j}$.

Game G^0. This game corresponds to our real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealC}_j}$, hence:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealC}_j}(1^\lambda, N, K) \to 1\right]$$

Game G^1 . This game is the same as G^0, except that we introduce our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ in the game. Our first simulator $\mathcal{S}_0$ is inputted at each time $t \in [\![N]\!]$ with the appropriate inputs *i.e.,* the current time t and the number of arms K. Our second simulator $\mathcal{S}_1$ is also added to the game, inputted with the number of arms K. Note that the outputs of our simulator, including the list of tuples $(s'_i, n'_i, v'_{i,t})$ for each $i \in [\![K]\!]$ and for each time step $t \in [\![N]\!]$ is not used as well as the outputted random cumulative rewards s'_i at the end of the protocol execution. Thus it does *not* impact the view of $\mathcal{C}_j$, providing no advantage for the adversary to distinguish, leading to a perfect indistinguishability:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^1_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^2. In this game, we replace the K executions of the arm pulling algorithm inputted with the selection bits, with a single execution of the arm pulling algorithm inputted with the best arm M computed by the challenger running the game. Based on the correctness of the $\mathsf{SelectArm}$ protocol, the adversary should not notice any modification and hence we obtain a perfect indistinguishability between this game G^2 and the previous game G^1:

$$\Pr\left[\mathsf{G}^1_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

In the following K games, we will replace the cumulative rewards in the rewards sending protocol. To improve clarity, we index these games as $\mathsf{G}^{3,k}$ for k ranging from 1 to K, in which we focus on k-th data owner $\mathcal{DO}_k$. The game $\mathsf{G}^{3,0}$ refers to G^2.

Game $\mathsf{G}^{3,k}$ for $k \in [\![1, K]\!]$. In this game, we replace the cumulative rewards s_k sent by the data owner $\mathcal{DO}_k$ during rewards sending protocol $\mathsf{SendRewards}$, with the randomly chosen cumulative rewards s'_k produced by the simulator $\mathcal{S}_1$. Hence, instead of receiving the share $\langle s_k \rangle^j$, the server $\mathcal{C}_j$ receives the share $\langle s'_k \rangle^j$ from which it cannot recover s_k or s'_k based on the security of the arithmetic share. Hence, we have a statistical

indistinguishability between $\mathsf{G}^{2,k-1}$ and $\mathsf{G}^{2,k}$:

$$\Pr\left[\mathsf{G}^{2,k-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \overset{\text{s}}{\approx} \Pr\left[\mathsf{G}^{2,k}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

In the following $N \cdot K$ games, we replace the cumulative rewards s_i whose shares are sent by the data owner $\mathcal{DO}_i$ at a time step $t \in [\![N]\!]$ during the arm selection protocol, used in our rewards saving mechanism. The following games are denoted $\mathsf{G}^{4,t,i}$ for $t \in [\![N]\!]$ and $i \in [\![K]\!]$, the game $\mathsf{G}^{4,t,0}$ referring to the game $\mathsf{G}^{4,t-1,K}$ and the game $\mathsf{G}^{4,0,K}$ being the game previous game $\mathsf{G}^{3,K}$.

Game $\mathsf{G}^{4,t,i}$ for $t \in [\![N]\!], i \in [\![K]\!]$. In this game, we replace the cumulative rewards s_i sent by the data owner $\mathcal{DO}_i$ with the randomly chosen cumulative rewards s'_i produced by the simulator $\mathcal{S}_0$. Hence, instead of receiving the share $\langle s_i \rangle^j$, the server $\mathcal{C}_j$ receives the share $\langle s'_i \rangle^j$. The indistinguishability between holds under the security of the arithmetic share. Hence, we have a statistical indistinguishability between $\mathsf{G}^{4,t,i-1}$ and $\mathsf{G}^{4,t,i}$:

$$\Pr\left[\mathsf{G}^{4,t,i-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \overset{\text{s}}{\approx} \Pr\left[\mathsf{G}^{4,t,i}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

In the following $N \cdot K \cdot l$ games, we are going to replace the p-th bit of the score $v_{i,t}$ produced by the data owner $\mathcal{DO}_i$ at time step $t \in [\![N]\!]$ with the p-th bit the random score $v'_{i,t}$ chosen by the simulator $\mathcal{S}_0$. For clarity, we denote the following games as $\mathsf{G}^{5,t,i,p}$ for $t \in [\![N]\!], i \in [\![K]\!]$ and $p \in [\![l]\!]$. The game $\mathsf{G}^{5,1,1,0}$ refers to the previous game $\mathsf{G}^{4,N,K}$ and the game $\mathsf{G}^{5,t,i,0}$ refers to the game $\mathsf{G}^{5,t,i-1,l}$.

Game $\mathsf{G}^{5,t,i,p}$ for $t \in [\![N]\!], i \in [\![K]\!], p \in [\![l]\!]$. In this game, we replace the p-th bit of the score $v_{i,t}$ by a p-th bit of the random score $v'_{i,t}$ chosen by the simulator $\mathcal{S}_0$. Hence, instead of receiving the share $\langle v_{i,t} \rangle^j$, the server $\mathcal{C}_j$ receives the share $\langle v'_{i,t} \rangle^j$. The indistinguishability of these two games is based on the security of the binary share. Note that the modification of a bit breaks the correctness of the $\mathsf{SelectArm}$ protocol. This does not bring any advantage for the server $\mathcal{C}_j$ since its view only contains shares and hence does not learn any plaintext information related on the protocol execution. This leads to a statistical indistinguishability between $\mathsf{G}^{5,t,i,p-1}$ and $\mathsf{G}^{5,t,i,p}$:

$$\Pr\left[\mathsf{G}^{5,t,i,p-1}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \overset{\text{s}}{\approx} \Pr\left[\mathsf{G}^{5,t,i,p}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

In the last game $\mathsf{G}^{5,N,K,l}$, the server $\mathcal{C}_j$ observes shares of random values chosen by the simulator, which corresponds to the ideal experiment $\mathsf{Exp}^{\mathsf{IdealC}_j}_{\mathcal{A},\mathcal{S}}$. As a result, we have:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealC}_j}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \overset{\text{s}}{\approx} \Pr\left[\mathsf{Exp}^{\mathsf{IdealC}_j}_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

□

Lemma 4. The view of the trusted randomness provider $\mathcal{P}$, denoted $\mathsf{View}(\mathcal{P})$ is limited to the all the randoms and associated shares, transmitted with the servers $\mathcal{C}_0$ and $\mathcal{C}_1$. The security proof of $\mathcal{P}$ is straightforward in the sense that view of $\mathcal{P}$ is fundamentally independent of the scores, selection bits and cumulative rewards. Indeed, the trusted randomness provider is not inputted with a sensitive data and never receive any information from another party, its single task being to generate and distribute a correlated randomness for the AND gates evaluation. Hence, any modification of the scores, the

selection bits or the cumulative rewards does not provide any advantage for $\mathcal{P}$. Therefore, it is clear that we obtain a perfect indistinguishability between the real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealP}}$ and the ideal experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{IdealP}}$:

$$\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealP}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealP}}(1^\lambda, N, K) \to 1\right]$$

By Lemma 3 and Lemma 4, we know that the honest-but-curious corruption of the servers $\mathcal{C}_j$ or the trusted provider $\mathcal{P}$ does not allow the adversary to learn any information on the scores, on the cumulative rewards, on the total cumulative rewards but also on the pulled armed. And since the controller $\mathcal{C}$ is composed of these nodes, we achieve the simulation security for the controller $\mathcal{C}$.

Security Proof for User

Theorem 3. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, for every security parameter $\lambda \in \mathbb{N}$, every polynomial-time adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that Salsa is perfectly SimU-secure:

$$\Pr\left[\mathsf{G}_{\mathcal{A}}^{\mathsf{RealU}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealU}}(1^\lambda, N, K) \to 1\right]$$

Proof. The view of $\mathcal{U}$, denoted $\mathsf{View}(\mathcal{U})$, consists of the single ciphertext c^s encrypting the total cumulative rewards s. In order to prove the above theorem, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ used in the game $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealU}}$:

- *Simulator* $\mathcal{S}_0$. Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$ and the number of arms K. At the end of its execution, the simulator $\mathcal{S}_0$ is required to output a list of tuples scores $(s'_i, n'_i, v'_{i,t})$ whose first element is a random cumulative rewards, the second element is a random number of pulls and the third and last element is a random score.
  The simulator $\mathcal{S}_0$ generates and returns K random scores $v'_{i,t}$, K random number of pulls n'_i and K cumulative rewards s'_i independently.
- *Simulator* $\mathcal{S}_1$. Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K as well as the total cumulative rewards s, and produces the simulated cumulative rewards s'_i randomly chosen from $\mathbb{N}$ for each arm $i \in [\![K]\!]$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealU}}$, and the ideal world corresponding to the experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealU}}$.

Game G^0. This initial game corresponds to the $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealU}}$ experiment. Therefore, we have a perfect indistinguishability:

$$\Pr\left[\mathsf{G}_{\mathcal{A}}^{0}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealU}}(\lambda, N, K) \to 1\right]$$

Game G^1. This game works exactly as the game G^0 except that we replace the execution of the $\mathsf{MAB.PullArm}$ algorithm executed for all arms, by a single execution of $\mathsf{MAB.PullArm}$ for the data owner indexed by M, the index of the data owner having the highest score. As a result, the selection bits outputted by the arm selection protocol are

now ignored. By correctness of the arm selection protocol $\mathsf{SelectArm}$, this modification does not affect the selected arm at each time step and hence does not affect the total cumulative rewards returned to the user $\mathcal{U}$. Hence, we have a perfect indistinguishability between G^0 and G^1:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^1_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^2. In this game, we focus on the integration of our first simulator $\mathcal{S}_0$: The simulator $\mathcal{S}_0$ is inputted at each time $t \in [\![N]\!]$ with t and the number of arms K. Recall that the output of the simulator $\mathcal{S}_0$ is composed of a K-sized list of tuples $(s'_i, n'_i, v'_{i,t})$. During the execution of the $\mathsf{SelectArm}$ protocol, at each time $t \in [\![N]\!]$ each data owner $\mathcal{DO}_i$ is inputted with its secret key sk_i and a random cumulative rewards s'_i, a random number of pulls n'_i and a random score $v'_{i,t}$. The obtained selection bits at the end of the arm selection protocol are ignored.

The view of the user $\mathcal{U}$ is limited to the total cumulative rewards s. The arm chosen by the $\mathsf{SelectArm}$ protocol is random since the arm selection protocol is inputted with completely random values. However, thanks to the challenger executing the correct $\mathsf{MAB.PullArm}$ algorithm for the arm having the best score, the correctness of the multi-armed bandits algorithm is guaranteed. Therefore, the returned total cumulative rewards is perfect indistinguishability between this game and the previous game G^1:

$$\Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}_0}(\lambda, N, K) \to 1\right]$$

Game G^3. In this game, we focus on the integration of our second simulator $\mathcal{S}_1$, inputted at the end of the protocol with the number of arms K and the total cumulative rewards s, corresponding to the sum of all cumulative rewards (*i.e.,* $\sum_{i=1}^{K} s_i$). The simulator $\mathcal{S}_1$ is asked to output a K-sized list of random scores $s'_1, \ldots, s'_K$, where each s'_i is inputted to $\mathcal{DO}_i$ at the beginning of the $\mathsf{SendRewards}$ protocol execution. From the observation that both $s_1 + \cdots + s_K$ (computed by the real multi-armed bandits execution) and $s'_1 + \cdots + s'_K$ (randomly sampled from the simulator $\mathcal{S}_0$) equals s, so we conclude that the view of $\mathcal{U}$ between the game G^3 and the game G^2 is perfectly indistinguishable:

$$\Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}_0}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(\lambda, N, K) \to 1\right]$$

And since G^3 corresponds to the ideal experiment $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}$, we observe a perfect indistinguishability between $\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}$:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealU}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{IdealU}}_{\mathcal{A},\mathcal{S}}(\lambda, N, K) \to 1\right]$$

□

Security Proof for a Data Owner

Theorem 4. Let $N \in \mathbb{N}$ be the budget and $K \in \mathbb{N}$ be the number of arms. Then, for every security parameter $\lambda \in \mathbb{N}$, every polynomial-time adversary $\mathcal{A}$, there exists a

simulator $\mathcal{S}$ such that:

$$\Pr\left[\mathsf{G}_{\mathcal{A}}^{\mathsf{RealDO}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealDO}}(1^\lambda, N, K) \to 1\right]$$

Proof. The view of the corrupted data owner $\mathcal{DO}_j$, denoted $\mathsf{View}(\mathcal{DO}_j)$, contains the cumulative rewards s_i as well as the number of pulls n_i. It also contains every score $v_{j,t}$ and every selection bit $b_{j,t}$ for each time step $t \in [\![N]\!]$. In order to prove the above theorem, we first construct the simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$:

- *Simulator* $\mathcal{S}_0$. Our first simulator $\mathcal{S}_0$ obtains as an input the current time $t \in [\![N]\!]$, the number of arms K, the index j of the corrupted data owner $\mathcal{DO}_j$ chosen by the adversary $\mathcal{A}$, its real score $v_{j,t}$ and a bit associated to the value $M \stackrel{?}{=} j$, where the index M corresponds to the arm index having its best score. At the end of the execution, the simulator $\mathcal{S}_0$ is required to output a $(K-1)$-sized list of tuples of the form $(s'_i, n'_i, v'_{i,t})$ whose first element is a random cumulative rewards, the second element is a random number of pulls and the third element is a random score.

  Our simulator $\mathcal{S}_0$ acts differently depending on the value of $M \stackrel{?}{=} j$. For the sake of clarity, let denote the value of $M \stackrel{?}{=} j$ by the bit b_{eq}. If the bit b_{eq} equals one, then the corrupted data owner $\mathcal{DO}_j$ expects to receive a positive selection bit. Hence, given its score $v_{j,t}$, the simulator generates $K-1$ random scores $v'_{i,t}$ with the condition that every $v'_{i,t}$ is strictly lower than $v_{j,t}$. This ensures that the selected data owner will be the corrupted one. At the opposite, when b_{eq} is a negative bit, then the simulator generates $K-1$ random scores with the condition that at least one score $v'_{i,t}$ is strictly higher than $v_{j,t}$. Note that the selected arm here is chosen at random since the simulator does not have any information on the index M. This models an idealised situation in which a data owner that is not pulled has at most $\frac{1}{K-1}$ chance to correctly identify the chosen arm. The cumulative rewards and the number of pulls returned by the simulator are chosen at random.
- *Simulator* $\mathcal{S}_1$. Our second simulator $\mathcal{S}_1$ obtains as an input the number of arms K as well as the index j of the corrupted data owner, and produces $K-1$ simulated cumulative rewards $s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_K$ randomly chosen from an arbitrary space, say $\mathbb{N}$.

By construction, our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ is polynomial-time and hence is a suitable simulator. We are ready to prove the indistinguishability of the real world corresponding to the real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealDO}}$, and the ideal world corresponding to the experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}^{\mathsf{IdealDO}}$.

Game G^0. This initial game corresponds to the $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealDO}}$ experiment. Therefore, we have a perfect indistinguishability:

$$\Pr\left[\mathsf{G}_{\mathcal{A}}^{0}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RealDO}}(\lambda, N, K) \to 1\right]$$

Game G^1. This game works exactly as the game G^0 except that we replace the execution of the $\mathsf{MAB.PullArm}$ algorithm executed for all arms, by a single execution of $\mathsf{MAB.PullArm}$ for the data owner indexed by M, the index of the data owner having the highest score. As a result, the selection bits outputted by the arm selection protocol

are now ignored. By correctness of the arm selection protocol $\mathsf{SelectArm}$, this modification does not affect the selected arm at each time step and hence does not affect the selection bits received by the corrupted data owner $\mathcal{DO}_j$ Hence, we have a perfect indistinguishability between G^0 and G^1:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^1_{\mathcal{A}}(\lambda, N, K) \to 1\right]$$

Game G^2. This game is the same as G^1, except that we introduce our simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$. The simulator $\mathcal{S}_0$ is inputted at each time step $t \in [\![N]\!]$ with the current time t, the number of arms K, the index j of the corrupted data owner $\mathcal{DO}_j$, its real score $v_{j,t}$, as well as the bit b_{eq} associated to the value $M \stackrel{?}{=} j$. The simulator $\mathcal{S}_1$ is inputted with the number of arms K and the corrupted index j. For the moment, the output of $\mathcal{S}$ is not used. Therefore, the view of $\mathcal{DO}_j$ remains unchanged leading to a perfect indistinguishability between G^1 and G^2:

$$\Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^3. This game is the same as G^2 except that at each time $t \in [\![N]\!]$, given the index j of the corrupted data owner $\mathcal{DO}_j$, the real score $v_{i,t}$ for i different of j is replaced by the random score $v'_{i,t}$ computed by the simulator $\mathcal{S}_0$. We do the same replacement for the cumulative rewards and the number of pulls. When the chosen data owner is the corrupted data owner $\mathcal{DO}_j$, then the simulator $\mathcal{S}_0$ receives a positive selection bit. Otherwise, the arm selection selects randomly an arm among all honest data owner leading to a random arm selection. By construction of the experiment, the correctness is preserved thanks to the challenger running the appropriate arm pulling. As a result, the execution of the multi-armed bandits is still valid (even if random scores are provided), leading to a perfect indistinguishability between G^2 and G^3:

$$\Pr\left[\mathsf{G}^2_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

Game G^4. This game is the same as G^3 except that during the $\mathsf{SendRewards}$ protocol execution, the real cumulative rewards $s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_j$ are replaced by the random cumulative rewards $s'_1, \ldots, s'_{j-1}, s'_{j+1}, \ldots, s'_j$ obtained at the output of the simulator $\mathcal{S}_1$. Observe that this modification does not affect the view of the corrupted data owner $\mathcal{DO}_j$ since it does contain any cumulative rewards s_i for i different of j. As a result, we obtain a perfect indistinguishability between G^3 and G^4:

$$\Pr\left[\mathsf{G}^3_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{G}^4_{\mathcal{A},\mathcal{S}}(1^\lambda, N, K) \to 1\right]$$

And since G^4 corresponds to the ideal experiment $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}$, we observe a perfect indistinguishability between $\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}$:

$$\Pr\left[\mathsf{Exp}^{\mathsf{RealDO}}_{\mathcal{A}}(1^\lambda, N, K) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{IdealDO}}_{\mathcal{A},\mathcal{S}}(\lambda, N, K) \to 1\right]$$

□

3.4.3 Empirical Study of Salsa

Compared to Tango, which is based on cryptographic primitives whose security holds under computational hypothesis, Salsa relies on secure two-party computations, more precisely on computation over shares, featuring an incredibly fast computation. For instance, the binary Xor gate requires the execution of a single $\oplus$ operation, hence without overhead. The arithmetic share enjoys the same computational efficiency. The And gate over shares, in constrast, requires a previously computed and shared correlated randomness whose computation is performed efficiently thanks to our randomness provider. The And gate evaluation performed by the servers requires few bit multiplications and few exclusive-or, which are efficient.

For all these reasons, Salsa is expected to be fast and hence scalable. This desired efficiency is confirmed by our implementation whose execution time (ignoring the communication times being highly dependent of the network configuration), following the same execution parameters of the benchmark of Tango, are presented in Figure 3.25. Our proof-of-concept achieves a thousand sequential executions of the SelectArm protocol in less than 150 milliseconds. Compared to Tango where a thousand sequential execution of Tango.SelectArm requires 2500 seconds. As a result, we obtain an approximated 16666 speedup factor. Based on these results, it is clear that Salsa is particularly fast and scalable. The presented execution times for the SendRewards protocol are negligible compared to the SelectArm protocol since it is executed only once, independently of the budget N. And since SendRewards requires only addition of shares, it is particularly efficient as well. Furthermore, Salsa does not require any setup, hence we have omitted its execution time.

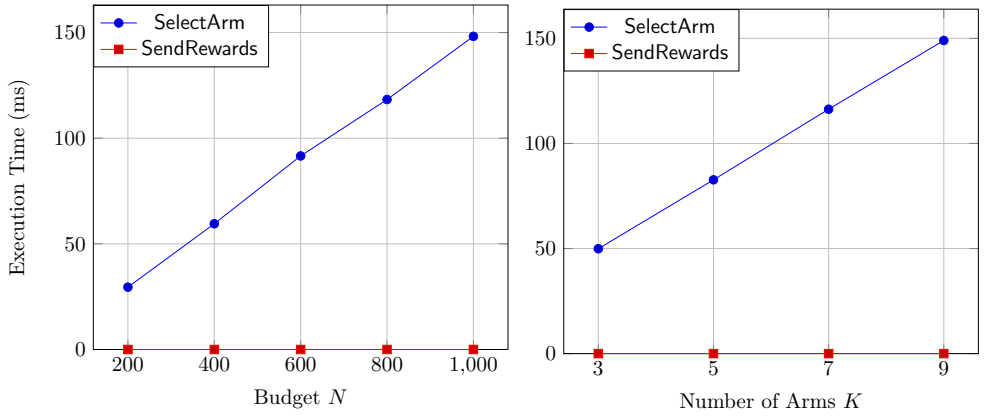


Figure 3.25: Execution time of Salsa. The presented execution times are the mean of 10 iterations. When varying the budget N, the number of arms K is set to 9. When varying the number of arms K, the budget N is set to 1000.

3.5 Conclusion & Discussion

The reward maximization problem in the federated multi-armed bandits setting, is a very interesting problem having several applications including recommendation systems, financial investment and medical treatment. In the previous works, this reward max-

imization problem was not studied under the prism of the federated learning, or with insufficient solutions such as our initial attempt Samba suffering from correctness and security issues.

In this chapter, we have first proposed a new formal security definition using the real-ideal paradigm, putting forward the desired security properties: Any party involved in the protocol should not learn any information on the score of a data owner (revealing sufficient information to recover reward probability), on the chosen arm, but also on the rewards generated by a data owner. Then, we have proposed two protocols fitting our security definition: Tango whose design aims to be closer than Samba, and Salsa based on a different paradigm, namely secure two-party computations.

Our first protocol Tango enjoys several features: First, it has been formally proved secure with respect to our security definition. Second, it is generic in the sense that any discrete multi-armed bandits following a precise and formal definition can be plugged in Tango, to enjoy the security features without affecting the returned total cumulative rewards. Third and last, Tango provides resistance against failures of one or more data owners. This useful property allows to return almost the same cumulative rewards which would be normally stored by a data owner being offline, even until the end of the protocol. We have provided an empirical evaluation of Tango confirming the advantage of our construction. First, despite the transposition of a standard multi-armed bandits algorithm into a discrete one, requiring the approximation of every score, we do not have observed any loss in the returned total cumulative rewards. Second, the returned total cumulative rewards using Tango is almost optimal, which confirms the interest of our approach to maximize the reward. Third and last, despite its simplicity, our failure resistance turns out to be an effective solution to limit the loss of rewards in case when some data owners go offline.

Despite all the mentioned advantages, it appears that Tango is not suitable for real-time applications due to the time-consuming homomorphic operations performed during the arm selection, requiring more than 2.5 seconds to pull the best among 9 arms. However, our construction remains interesting for applications where the arm selection is allowed to be performed in a longer delay. For instance, this is suitable for movie recommendation systems where a user pays to have recommendation for movies to watch. In this scenario, the delay between two movies seen by the user is long enough to evaluate the arm selection and later to recommend a new movie. The security properties enjoyed by Tango prevent the system to learn the recommanded movie.

To overcome the execution time bottleneck due to the homomorphic comparison in Tango, we have proposed a second protocol Salsa, offering all the same advantages offered by Tango, including correctness, genericity, security, and resistance against data owner failure, while featuring a fast arm selection, performed in less than 0.15 millisecond. Hence, Salsa is suitable for real-time applications.

Chapter 4

Anonymous, Transferable and Auditable Tickets

Contents

Electronic tickets (*E-Ticket*) have become the standard. The rationale behind the digitisation of the ticket industry is both practical and economical. More sales can be achieved by allowing users to purchase tickets from anywhere. The practical aspect of ticket digitisation entails significant drawbacks, resulting in a negative impact on the second-hand market and the protection of users' privacy. In a world where privacy is a central concern, it is essential to preserve the multiple facets of a paper ticket, such as the right to trade them, and ensure full confidence in their validity. However, the original property of non-replicable paper tickets is often lost with the development of E-Tickets. A formal security model for electronic ticketing is needed to ensure their security, and a system that combines the best properties of both worlds is necessary.

The research has focused on multiple cryptographic technologies with properties similar to their physical counterparts. The subject of *e-cash* [BCFK15, BFQ21, TH16], *e-coupon* [CES+05, LMY14, Ngu06] (e-coupon are similar to e-cash with items chosen when redeeming an e-coupon remains unknown to the service provider) or *n-times anonymous authentication* [TFS04, CHK+06] appears to be closely associated with our issue. Even though they have been studied in the literature, their design makes them incompatible with being used as a ticketing system. In numerous electronic cash systems [BCFK15, BFQ21, TH16], the practice of double-spending, wherein a coin is spent twice, is mitigated by a central bank that maintains a record of the spent coins. When receiving a coin from a merchant, the bank checks to ensure that it does not belong to the list of spent coins. Therefore, the bank is able to detect if a coin has been double-spent, and can compensate the scammed merchant. This mechanism, however, is not suitable to construct a desirable ticketing system, since a scam would be left undetected by a second merchant accepting the same coin until they both reach the bank. This fraud can

be generalised to transfer of coins in transferable e-cash. Putting it in different terms, cheaters could resell twice a ticket and this would be left undetected until they try to access the event. On the other hand, in *e-coupon* systems [CES+05, LMY14, Ngu06], the transfer may appear unfavourable or even atypical, as a coupon has been issued by a merchant to a specific customer with the intention of gaining their loyalty, whereas *n-times anonymous authentication* [TFS04], cannot be safely transferred. With e-tickets, it is imperative to guarantee the validity of a ticket at any time in order to avoid double-spent tickets. This primary concern renders every electronic cash system that prevents double-spending at coin deposit irrelevant to our issue. Additionally, a coin is validated by the bank during the deposit process in e-cash [BFQ21]. In the case of e-ticketing, a ticket can be validated by one of the many terminals. All of these disparities substantiate the necessity for a specific approach to electronic ticketing. E-ticketing systems have been studied both in the industry [Ave16, Pro16] for practical purposes and in academia for understanding social behaviours regarding ticket resale and the incentives behind this process [LS14, CDŞ14]. The majority of the present ticketing systems that have been developed by the industry are centred on the "standard" functionality of electronic tickets, which entails the provision and validation of tickets. These systems are not designed to ensure secure ticket transfer. Hence, an honest client has high chances of buying a duplicated ticket from a malicious user. More advanced systems, attempting to address this issue, provide authentication via a distributed architecture, such as the blockchain [Ave16, Pro16]. Even if ticket validity is now ensured during a transfer (by checking if the ticket is valid in the blockchain), it moves away from the current e-ticketing system organisation where the tickets are stored in a database. These methods however, are in opposition to the currently centralised event organisation and implies a larger energy consumption [dV18, SBFK20]. Users' anonymity in e-ticketing systems has not been considered in the latest protocols [Ave16, HAY12, NAJ15, Pro16]. Nevertheless, anonymity is guaranteed by physical tickets, and it remains crucial to prevent the event organisers from collecting the identity of ticket purchasers unless necessary. Blockchain-based solutions can serve this purpose [Ave16, Pro16], but for the aforementioned argumentation, we exclude them from our investigation.

Summary of Contribution

Considering the above problems and the lack of existing formalism, we design an e-ticket scheme with proven security. It features three central aspects as it is *centralised*, *transferable* and *anonymous*, with the latter property that can be mitigated by *auditable* feature at need. We discuss the requirements of such a protocol, its capabilities, and limitations, and encompass it in a model. We introduce the first security model for *E-Ticket Scheme* (ETS). The security is formalised through experiments, modelling *unforgeability*, *ticket privacy*, *anonymity of users*, and *no-double-spending*, preventing to execute a refund, transfer or validate twice with the same ticket.

Our *provably secure E-Ticket* scheme is called Applause. It allows users to purchase, refund, validate tickets, but also transfer their tickets to another user. During all the interactions, users' anonymity from the event organiser point-of-view is ensured. We

propose a protocol addressing all the above-mentioned properties. Its security is proven based on computational proofs in the random oracle.

Payments can be made during at least three steps of our process: The purchase, the refund, and the transfer of tickets. Traditional payment does not guarantee anonymity, therefore, using it in our construction would trivially break the anonymity of users. For instance, the most standard payment protocol EMV [EMV11] does not protect privacy. In our protocol, the payment is modelled as a generic cryptographic building block, requiring *anonymity*, a mandatory assumption only required to ensure full anonymity. More precisely, our protocol is as anonymous as the payment method in use. We stress that it is not specific for our protocol but can be applied for every protocol ensuring anonymity and involving payment. Alternative payment protocols that guarantee anonymity of users exist, ranging from simple anonymous payment method [Pay18] to more advanced constructions [DGGB22, MSJP22], can be used in our protocol. To reveal the identity of users, reaching *auditability*, we present SPOTLIGHT, an extended version of APPLAUSE, where the identity of *all* users can be revealed *only* by a third-party called the judge, employing auditable anonymous credentials.

Related Work

Most of the production-ready deployed systems guarantee the "standard" functionality of ticket payment and delivery. Our system provides users with the ability to transfer a ticket to another user, while maintaining anonymity, in addition to the standard functionalities. All existing systems that allow for additional functionalities that we identify rely on blockchains [Ave16, Pro16]. Blockchain-based ticketing systems can easily achieve transfer of a ticket by checking if the ticket is still valid on the blockchain, and can achieve anonymity by the design of the blockchain. However, the blockchain requires the upkeep of a distributed ledger and numerous signatures, which means that numerous servers are needed to safeguard the validity of the network. Low-consumption and optimised computation has been a keystone of cryptography. As shown in [dV18, SBFK20], blockchains are more computationally demanding than a centralised design, but also require large-scale procedures. Our system is centralised to enhance efficiency and align with the existing topology of event organisation.

Previous works, initiated by [LLG01] in 2001 and followed up by a number of papers, essentially evaluating either the practicality [HCE05], the interface design [XTB04] or the security [SSV08] of existing systems in the public transport. Subsequent works have proposed a novel ticketing system. They can be divided into three categories. The first category focuses on the design of a ticketing system [HAY12, NAJ15], ensuring the validity of a ticket, without considering privacy. The second category aims to guarantee the privacy of the user, however it does not permit transfer or auditability. In [KLG13], they study the possibility of a ticketing system with privacy of users, including the billing step. The authors of [Gud13, GSK14] have studied the possibility of using RFID and NFC while ensuring privacy of users, using unlinkable certified tokens [MDND15] for public transport ticketing or anonymous credentials [HCS$^+$21] for general purposes tickets. Furthermore, construction of [HCS$^+$21] takes a few seconds whereas our protocol APPLAUSE requires 130 milliseconds to purchase, transfer and validate a ticket, and 250

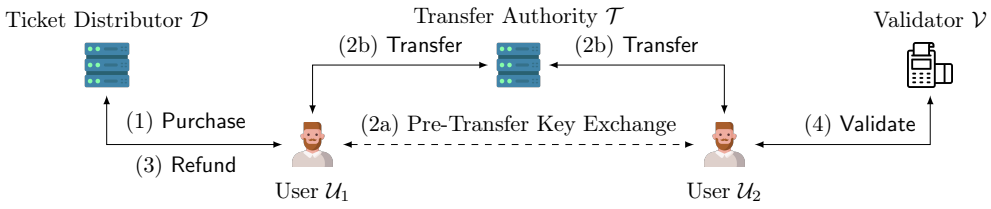


Figure 4.1: Representation of the ETS model.

milliseconds for SPOTLIGHT, making our contribution scalable. Unlike [HCS+21], we include the payment process in the protocol as well. The third line of research proposes ticketing systems based on a distributed ledger, which are less efficient compared to a centralised setting, due to the number of involved servers and communication time. The first paper to propose such a system is presented in [LNGH19], which is based on blockchain. Recently, a new ticketing system has been proposed in [YZ22] based on NFT. For practical and efficiency reasons depicted above, our work moves away from the distributed approach.

This work is compared to publications in less specialised areas, which includes *e-cash* [BCFK15, BFQ21, TH16] and *e-coupon* [CES+05, LMY14, Ngu06], in which a trusted authority generates tokens attached to a value. Due to double-spending enforcement checked only by the bank, followed by a compensation process, they miss the guarantees of ticket validity upon transfer. On the other hand, *n-times anonymous authentication*[TFS04, CHK+06] allows a limited number of authentications before leaking user's identity, limiting the ticket purchasing, as well as the interest of this approach for ticketing. In [CHK+06], restrictions of the concept are augmented by time frames that further limit the authentication process, yet it falls short in ensuring transfer guarantees.

Considerations could also be given to *Anonymous Credentials* [Cha85, CDLPK22, MSM23] or *Attribute-Based Signatures* [BK19, MPR11]. These mechanisms are dedicated for authentication based on predicate matching, identifying information about the signer. Yet again, transferability remains unaddressed as a standalone aspect in this particular work.

Chapter Organisation

In Section 4.1, we present the general overview of a ticketing system, including the desired architecture and the setting in which this work takes place. In Section 4.2, we introduce the security model defining the features and the security that an anonymous transferable ticketing system is expected to achieve. In Section 4.3, we introduce our first transferable ticketing protocol APPLAUSE, ensuring ticket unforgeability, ticket privacy as well as anonymity for users. In Section 4.4, we introduce our second transferable ticketing protocol SPOTLIGHT in which an additional incorruptible third-party called the judge is able to reveal the identity of users, allowing SPOTLIGHT to achieve auditability. Finally, in Section 4.5 we study the overhead due to the cryptographic operations by studying our implementation of APPLAUSE and SPOTLIGHT written in Rust.

4.1 Overview of Electronic Tickets

We move away from the decentralised approach used in previous work to focus on the currently used and centralised architecture. Our system divides the ticket handling into three phases, namely ticket purchase, ticket transfer, and ticket refund or validation. These phases can be increased with a judge for auditability.

Architecture. A *user* $\mathcal{U}$ willing to purchase a ticket for an event, contacts the *ticket distributor* $\mathcal{D}$, which issues a ticket tk to $\mathcal{U}$ in exchange of a payment. Once $\mathcal{U}$ holds a ticket, it can authenticate itself to a *validator* terminal $\mathcal{V}$ in order to get access to the designated event. In some cases, a user $\mathcal{U}_1$ holding a ticket might not be able to benefit from its purchase, in such cases our protocol offers two options: A *refund*, or a *transfer* of its ticket to another user *e.g.,* to $\mathcal{U}_2$. The refund is proceeded between user $\mathcal{U}_1$ and the ticket distributor $\mathcal{D}$. The transfer scenario encompasses both the cases where $\mathcal{U}_1$ sells its ticket to another user $\mathcal{U}_2$ or offers it. The latter consisting of the same transfer protocol without the payments. The ticket transfer is modeled as an interaction between $\mathcal{U}_1$, $\mathcal{U}_2$, and a *transfer authority* $\mathcal{T}$ acting as a guarantee of the exchange. $\mathcal{T}$ ensures the validity of the ticket to $\mathcal{U}_2$, prevents $\mathcal{U}_1$ to resell a ticket for profit by controlling the price, but also prevents $\mathcal{U}_2$ to obtain a ticket without paying it. While ticket sale is a fairly straightforward process, its transfer can be achieved through multiple scenarios. It has been shown in [EY80] that a fair transfer between two users is impossible without a third party. To make the designation of the ticket receiver possible, we assume a communication between $\mathcal{U}_1$ and $\mathcal{U}_2$ before the transfer of the ticket, allowing them to exchange keys. This communication channel can be obtained for instance via Bluetooth during a physical meeting. Last, to attend to an event, $\mathcal{U}$ has to validate a ticket against a validator $\mathcal{V}$ through an anonymous validation. At any time during the process, the *Judge* $\mathcal{J}$ can open a ticket to recover the associated user's identity.

Anonymity and Auditability. Anonymity of users is ensured during every interaction with the system (*i.e.,* the ticket distributor $\mathcal{D}$, the transfer authority $\mathcal{T}$ and the validator $\mathcal{V}$) and all over the process. During the protocol, either authentication is not required or users authenticate themselves using randomised identities *i.e.,* randomised keys. If desired, our protocol Applause can be turned into an auditable version called Spotlight, where an external authority refers as the *judge* denoted $\mathcal{J}$, recovers the identity of $\mathcal{U}$. The auditability setting remains consistent with the one presented for *Auditable Anonymous Credentials* in [CDLPK22]. In a nutshell, given a certificate issued by the judge $\mathcal{J}$, the user $\mathcal{U}$ computes a so-called "proof" denoted π, attesting the ownership of the certificate to the validator $\mathcal{V}$. Given such a proof π, kept on a record, $\mathcal{J}$ retrieves the public key of the user $\mathcal{U}$ having generated the proof π, hence obtaining the identity of $\mathcal{U}$. The identity of a user still remains private against $\mathcal{D}$, $\mathcal{T}$ or $\mathcal{V}$, thanks to the anonymity property of the proof π. Notably, π is being publicly verifiable meaning that anyone can verify the ownership of the certificate by the user having generated π. We stress that if the judge is compromised, only the users' anonymity would be at risk. The tickets remain valid, ensuring that honest individuals can utilize their tickets without any issues. A compromised judge $\mathcal{J}$ would not have the capability to forge new

tickets or alter the validity status of previously issued tickets. The purpose of auditability is not to prevent double spending, but rather to fulfill legal mandates concerning participant identity recovery. This aspect is crucial in unforeseen circumstances such as force majeure or disasters.

Validation Setting. Large scale events often result in an overloaded network due to the number of attendees. Multiple terminals are needed to validate the tickets simultaneously. Hence, communications coming to and from the validation terminals are limited in size and number. One would like to assume that the validator is offline after an initial setup. As a single ticket should be valid for any terminal, and without communication between the terminals, the same ticket could be accepted by each of them, constituting a forgery for any ticketing system. Hence, the validators must be online and communicate. For efficiency, we rely on a central authority instead of a distributed ledger to agree on valid tickets at time t. A validation protocol has to ensure anonymity of users, and thus cannot authenticate the user. However, an adversary able to relay all communications between a legitimate user and a validator, can easily perform what is called a *relay attack* [RNTS07], where the adversary can have the access granted without paying a ticket, by simply blocking and sending all messages sent by a legitimate user to the validator, in its own name. As a result, the validator grants the access to the adversary instead of the legitimate user. Badly, neither the user or the validator can notice this attack. To prevent this issue, we have chosen to construct a validation protocol based on a physical channel (such as QRcode [Int06] scanned by the validator or Bluetooth's password shared verbally [NSI+15]) during the last step of the protocol to prevent relay attacks by an adversary. $\mathcal{U}$ gets a token, compared through this channel to the token sent by $\mathcal{V}$.

4.2 Definition of Ticketing System

Most ticketing systems are designed to allow the sale of a seat at an event, with each seat being associated with a metadata such as the seat number. This can be more general, and we consider a scenario where tickets are characterised by an *event identifier* $\mathsf{ide} \in \mathsf{ID}_{\mathcal{E}} \subset \mathbb{N}$ and by a *serial number* $\mathsf{idp} \in \mathsf{ID}_{\mathcal{P}} \subset \mathbb{N}$. The set of identifiers and the set of serial numbers referred to event and seat number, are assumed publicly known. Below, we define an *E-Ticket Scheme* based on a security parameter 1^λ. Recall that by $\mathsf{P}\langle \mathsf{E}_1(i_1), \ldots, \mathsf{E}_n(i_n)\rangle \rightarrow \mathsf{E}_1(o_1), \ldots, \mathsf{E}_n(o_n)$; we denote the protocol P played between parties E_j, taking as input i_j and outputting o_j. For simplicity, we may omit a party in the output part of the notation if the party does not produce output.

Definition 25 (ETS)**.** An *E-Ticket Scheme* $\Pi = (\mathcal{D}\mathsf{KeyGen}, \mathcal{T}\mathsf{KeyGen}, \mathcal{V}\mathsf{KeyGen}, \mathcal{U}\mathsf{KeyGen}, \mathsf{Purchase}, \mathsf{Refund}, \mathsf{Transfer}, \mathsf{Validate})$ is a tuple of PPT algorithms:

- $\underline{\mathcal{D}\mathsf{KeyGen}/\mathcal{T}\mathsf{KeyGen}/\mathcal{V}\mathsf{KeyGen}/\mathcal{U}\mathsf{KeyGen}(1^\lambda) \rightarrow (sk, pk)}$: Given a security parameter 1^λ, outputs a key pair (sk, pk).
- $\underline{\mathsf{Purchase}\langle \mathcal{U}(sk_{\mathcal{U}}, (\mathsf{ide}, \mathsf{idp})), \mathcal{D}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{U}(\mathsf{tk}), \mathcal{D}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{st})}$: At the beginning of the ticket purchase protocol between the user $\mathcal{U}$ and the ticket distributor $\mathcal{D}$, $\mathcal{U}$ identifies the desired seat of the event using the assumed publicly-

available couple of event and seat identifiers (ide, idp). At the end of the purchase protocol, the user obtains and returns the purchased ticket tk, while $\mathcal{D}$ updates st and returns a success bit b, the event and seat identifiers (ide, idp), the ticket tk and the updated shared state st.

- $\underline{\mathsf{Refund}\langle\mathcal{U}(sk_{\mathcal{U}}, \mathsf{tk}), \mathcal{D}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{U}(b), \mathcal{D}(b, \mathsf{tk}, \mathsf{st})}$: Given a ticket tk and its key $sk_{\mathcal{U}}$, $\mathcal{U}$ asks the ticket distributor $\mathcal{D}$ for a refund. Both entities output a success bit b, while $\mathcal{D}$ additionally updates st and returns tk.
- $\underline{\mathsf{Transfer}\langle\mathcal{U}_1(sk_{\mathcal{U}_1}, \mathsf{tk}), \mathcal{T}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{U}_2(sk_{\mathcal{U}_2}, p)\rangle \rightarrow \mathcal{U}_1(b), \mathcal{T}(b, p, \mathsf{tk}, \mathsf{tk}', \mathsf{st}), \mathcal{U}_2(\mathsf{tk}')}$: The transfer protocol allows the user $\mathcal{U}_1$ owning a secret key $sk_{\mathcal{U}_1}$ and a ticket tk, to transfer tk' to the user $\mathcal{U}_2$ holding a secret key $sk_{\mathcal{U}_2}$. The protocol relies on $\mathcal{T}$ inputting the secret key $sk_{\mathcal{T}}$ and the shared state st. As a result, $\mathcal{U}_1$ and $\mathcal{T}$ output a success bit b, while $\mathcal{T}$ also updates the shared state st and outputs $p = (\mathsf{ide}, \mathsf{idp})$, the transferred tk, the new ticket tk' and the updated shared state st. Finally $\mathcal{U}_2$ obtains and returns a new ticket tk' for the same event and seat identifiers (ide, idp).
- $\underline{\mathsf{Validate}\langle\mathcal{U}(sk_{\mathcal{U}}, \mathsf{tk}), \mathcal{V}(sk_{\mathcal{V}}, \mathsf{st})\rangle \rightarrow \mathcal{U}(b), \mathcal{V}(b, \mathsf{tk}, \mathsf{st})}$: User $\mathcal{U}$ inputs a ticket tk and its key $sk_{\mathcal{U}}$, and interacts with $\mathcal{V}$ inputting its key $sk_{\mathcal{V}}$ and a state st in order to validate the ticket tk. The protocol ends with $\mathcal{U}$ and $\mathcal{V}$ returning a validation bit b, a ticket tk and the state st from the validator $\mathcal{V}$.

The Shared State. The above model includes a state st, shared between the ticket distributor $\mathcal{D}$, the transfer authority $\mathcal{T}$ and the validator $\mathcal{V}$. The shared state allows to synchronise ticket status among the different entities. It prevents, for instance, a double validation or a validation after a refund. This shared state could be seen as a white-list or a blacklist, the latter being used in our protocol. In our model, this state is not kept secret: All adversaries have *read-only* access to the state at any time using an oracle called OLeakState. This state can be implemented as a dedicated server maintaining a database. Note that $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{V}$ share a common state while being represented as distinct entities. This modelling choice aligns with the actual structure of the organisation under consideration, reflecting its physical reality.

Security Properties of Electronic Ticket

The most basic property that a ticketing system is required to provide is *correctness*, which briefly states that an honestly purchased or ticket received via a ticket transfer should be either refunded or validated. Note that a ticket should not be accepted for both, otherwise leading to a double spending of the same ticket. This double acceptation of a ticket constitutes a property on its own and is referred in this work as the *double-spending* property. Listening the communication between a user and the ticketing system should not allow another user to forge a new ticket but also to reveal the ticket owned by user. These two properties are called respectively *ticket unforgeability* and *ticket privacy*. Finally, the property at the heart of our contribution is the user anonymity. This anonymity property is subdivided into two distinct properties: First, we have the *pseudonymity* property stating in a nutshell that interacting with a user does not reveal its identity, and *unlinkability* stating that an interaction with a user cannot be

linked with a previous interaction, preventing the system to identify if two interactions is coming from the same user.

Correctness

The correctness of an ETS scheme consists, for any ticket tk obtained via the honest execution of Purchase or Transfer protocols to be accepted by the Refund or Validate protocols, producing a success bit b equalling 1, even if Transfer is executed sequentially over any sequence of n users $\mathcal{U} = \mathcal{U}_1, \ldots, \mathcal{U}_n$ where each user $\mathcal{U}_i$ is equipped of a key pair (sk_i, pk_i). Therefore, for every security parameter $\lambda \in \mathbb{N}$, any state st where the place (ide, idp) is not already purchased, it holds that:

$$
\begin{aligned}
(sk_{\mathcal{D}}, pk_{\mathcal{D}}) \leftarrow \mathcal{D}\mathsf{KeyGen}(1^\lambda), (sk_{\mathcal{T}}, pk_{\mathcal{T}}) \leftarrow \mathcal{T}\mathsf{KeyGen}(1^\lambda), (sk_{\mathcal{V}}, pk_{\mathcal{V}}) \leftarrow \mathcal{V}\mathsf{KeyGen}(1^\lambda) \\
\mathsf{Purchase}\langle \mathcal{U}_1(sk_1, (\mathsf{ide}, \mathsf{idp})), \mathcal{D}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{U}(\mathsf{tk}_1), \mathcal{D}(1, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}_1, \mathsf{st}) \\
\forall i \in [\![1, n-1]\!], j \leftarrow i+1, p \leftarrow (\mathsf{ide}, \mathsf{idp}) : \\
\mathsf{Transfer}\langle \mathcal{U}_i(sk_i, \mathsf{tk}_i), \mathcal{T}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{U}_j(sk_j, p)\rangle \rightarrow \mathcal{U}_i(1), \mathcal{T}(1, p, \mathsf{tk}_i, \mathsf{tk}_j, \mathsf{st}) \\
\textbf{then } \mathsf{Refund}\langle \mathcal{U}(sk_n, \mathsf{tk}_n), \mathcal{D}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{U}_n(1), \mathcal{D}(1, \mathsf{tk}_n, \mathsf{st}) \\
\textbf{or } \mathsf{Validate}\langle \mathcal{U}(sk_n, \mathsf{tk}_n), \mathcal{V}(sk_{\mathcal{V}}, \mathsf{st})\rangle \rightarrow \mathcal{U}_n(1), \mathcal{V}(1, \mathsf{tk}_n, \mathsf{st})
\end{aligned}
$$

Oracles Description

A ticketing system is considered secure with respect to a security definition if it is proven resistant against any adversary having capabilities. In our game-based security definition, the capabilities of an adversary are modelled using oracles. Since the introduced oracles require specific parameters such that set of users to work properly, we begin our description by providing the meaning of the symbols used. The symbol $\mathcal{SU} = \{\mathcal{U}_k\}_k$ denote the set of initialised users, potentially empty at the beginning. Each user $\mathcal{U}_k$ is associated with a set TK_k representing the tickets it owns. Observe that a ticket in TK_k is not only composed of a real ticket tk, but is a triplet consisting of the ticket tk itself (produced by the ETS scheme during the purchase or the transfer), the targeted event, the set of identifiers and a boolean, equalling true if the ticket is valid. By st, we denote the shared state among $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{V}$, initially empty and gradually populated with elements corresponding to revoked tickets.

User Creation and Corruption Oracles. During the execution of the security experiments where the challenger runs the system, the adversary $\mathcal{A}$ has access to two oracles OCreateUser and OCorruptUser depicted in Figure 4.2 allowing respectively to create an honest user but also to corrupt a user of its choice. The first oracle OCreateUser does not expect any parameter from the adversary and runs the key generation algorithm $\mathcal{U}$KeyGen, before storing the obtained keys in the $\mathcal{SU}$ set where each entry $\mathcal{U}_i$ is composed of four elements: Its private key, its public key, its set of tickets TK_i and a bit corr_i to remember if $\mathcal{U}_i$ is corrupted. The user corruption oracle OCorruptUser expects as an input from the adversary the index i, modelling the desire for the adversary to corrupt

$\mathcal{U}_i$. The corruption oracle ends by returning the secret information of $\mathcal{U}_i$ namely its secret key $sk_{\mathcal{U}_i}$ as well as its set of tickets TK_i.

$\mathsf{OCreateUser}(\mathcal{SU})$

1 : $i \leftarrow |\mathcal{SU}|, (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}) \leftarrow \mathcal{U}\mathsf{KeyGen}(pp)$

2 : $\mathcal{SU} \leftarrow \mathcal{SU} \cup \{\mathcal{U}_i = (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \emptyset, 0)\}$

3 : **return** $(i, pk_{\mathcal{U}})$

$\mathsf{OCorruptUser}(\mathcal{SU}; i)$

1 : $\mathcal{SU} \xrightarrow{p} \{\mathcal{U}_k\}_k, \mathcal{U}_i \xrightarrow{p} (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i, \mathsf{corr}_i)$

2 : $\mathsf{corr}_i \leftarrow 1$

3 : **return** $(sk_{\mathcal{U}_i}, \mathsf{TK}_i)$

Figure 4.2: Description of OCreateUser and OCorruptUser oracles.

State Leakage Oracle. The security of a ticketing system should be secure *even* in case where the shared state is made publicly available. For this reason, to allow the adversary to read the shared state, we have introduced the OLeakState oracle depicted in Figure 4.3 expecting no parameter from the adversary and returning the shared state. Note that the adversary is not allowed to write on the shared state.

$\mathsf{OLeakState}(st)$

1 : **return** st

Figure 4.3: Description of the OLeakState oracle.

Ticket Purchase Oracle. The ticket purchase oracle OPurchase depicted in Figure 4.4 expects from the adversary an index i as well as a place identifier $(\mathsf{ide}, \mathsf{idp})$. This oracle simulates the purchase of a ticket for the designated place provided by the adversary. The oracle supports the ticket purchase for honest users since the secret key of the user is owned by the challenger, but also for corrupted users since the ticket purchase requires the involvement of the ticket distributor $\mathcal{D}$, whose secret key $sk_{\mathcal{D}}$ is owned by the challenger. In case where the ticket purchase succeeds, the state of the user $\mathcal{U}_i$ is updated to integrate the freshly purchased ticket.

Ticket Refund Oracle. The ticket refund oracle ORefund depicted in Figure 4.5 works similarly to the ticket purchase oracle OPurchase, except that the executed protocol is the Refund protocol.

Ticket Transfer Oracle. The ticket transfer oracle OTransfer depicted in Figure 4.6 expects from the adversary two indexes i and j representing respectively the user $\mathcal{U}_i$ being the sender of the ticket and $\mathcal{U}_j$ being the receiver of the ticket. In addition, the adversary provides the seat identifier $(\mathsf{ide}, \mathsf{idp})$. As previously done, the ticket transfer oracle OTransfer runs the Transfer protocol between the ticket transfer authority $\mathcal{T}$ and the two designated users. The oracle behaviour depends on the corruption state of

$\mathsf{OPurchase}(\mathcal{SU}, sk_{\mathcal{D}}; i, (\mathsf{ide}, \mathsf{idp}))$

1 : $\mathcal{SU} \xrightarrow{p} \{\mathcal{U}_k\}_k, \mathcal{U}_i \xrightarrow{p} (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i, \mathsf{corr}_i)$
2 : **if** $\mathsf{corr}_i = 1$: /* *Corrupted user* */
3 : $\quad \mathsf{Purchase}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{st})$
4 : $\quad$ **if** $b = \top : \mathcal{U}_i \leftarrow (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i \cup \{(\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 1)\}, \mathsf{corr}_i)$
5 : **else** : /* *Honest user* */
6 : $\quad (\mathsf{ide}, \mathsf{idp}) \xleftarrow{\$} \mathsf{ID}_{\mathcal{E}} \times \mathsf{ID}_{\mathcal{P}}$
7 : $\quad \mathsf{Purchase}\langle \mathcal{C}(sk_{\mathcal{U}_i}, (\mathsf{ide}, \mathsf{idp})), \mathcal{C}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{C}(\mathsf{tk}), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{st})$
8 : $\quad$ **if** $b = \top : \mathcal{U}_i \leftarrow (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i \cup \{(\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 1)\}, \mathsf{corr}_i)$

Figure 4.4: Description of the OPurchase oracle.

$\mathsf{ORefund}(\mathcal{SU}, sk_{\mathcal{D}}; i, (\mathsf{ide}, \mathsf{idp}))$

1 : $\mathcal{SU} \xrightarrow{p} \{\mathcal{U}_k\}_k, \mathcal{U}_i \xrightarrow{p} (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i, \mathsf{corr}_i)$
2 : **if** $\mathsf{corr}_i = 1$: /* *Corrupted user* */
3 : $\quad \mathsf{Refund}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b, \mathsf{tk}, \mathsf{st})$
4 : $\quad$ **if** $b = \top : \mathcal{U}_i \leftarrow (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i \cup \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 0\}, \mathsf{corr}_i)$
5 : **else** : /* *Honest user* */
6 : $\quad \mathsf{TK}_i \xrightarrow{p} \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), v\}$
7 : $\quad \mathsf{Refund}\langle \mathcal{C}(sk_{\mathcal{U}}, \mathsf{tk}), \mathcal{C}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{C}(b), \mathcal{C}(b, \mathsf{tk}, \mathsf{st})$
8 : $\quad$ **if** $b = \top : \mathcal{U}_i \leftarrow (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i \cup \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 0\}, \mathsf{corr}_i)$

Figure 4.5: Description of the ORefund oracle.

the users. For instance, if both users are corrupted then Transfer protocol is executed between the challenger simulating the ticket transfer authority $\mathcal{T}$ and the adversary simulating both users. At the opposite, if no user is corrupted, then all interactions are simulated by the challenger running the OTransfer oracle.

Ticket Validation Oracle. The validation oracle OValidate depicted in Figure 4.7 expects from the adversary the index i designating the user $\mathcal{U}_i$, as well as a seat identifier $(\mathsf{ide}, \mathsf{idp})$. Similarly to ORefund and OValidate, the ticket validation oracle runs the ticket validation protocol Validate. Again, depending on the corruption state of the designated user $\mathcal{U}_i$, the oracle runs either the Validate protocol with the adversary if $\mathcal{U}_i$ is corrupted, or with himself in case where the adversary is not corrupted.

Oracles for Anonymity. All the previously introduced oracles are specifically designed for the case where the challenger simulates the system including $\mathcal{D}$, $\mathcal{T}$ and $\mathcal{V}$. These oracles are not particularly suited for security experiments where the challenger does not simulate the system, for example with the anonymity. Indeed, in this case, the anonymity of a ticketing system has to be provided even against the system. For this reason, we introduce four anonymity-focused oracles $\mathsf{OPurchase}_b, \mathsf{ORefund}_b, \mathsf{OTransfer}_b$ and $\mathsf{OValidate}_b$.

The ticket purchase oracle $\mathsf{OPurchase}_b$ depicted in Figure 4.8 expects from the adversary a seat identifier $(\mathsf{ide}, \mathsf{idp})$ and executes the Purchase protocol where the ticket

$\mathsf{OTransfer}(\mathcal{SU}, sk_{\mathcal{T}}; i, j, (\mathsf{ide}, \mathsf{idp}))$

1 : $\mathcal{SU} \xrightarrow{p} \{\mathcal{U}_k\}_k, \mathcal{U}_i \xrightarrow{p} (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i, \mathsf{corr}_i)$,

2 : $\mathcal{U}_j \xrightarrow{p} (sk_{\mathcal{U}_j}, pk_{\mathcal{U}_j}, \mathsf{TK}_j, \mathsf{corr}_j)$

3 : **if** $\mathsf{corr}_i = 1 \wedge \mathsf{corr}_j = 1$: /* *Corrupted users* */

4 : $\quad \mathsf{Transfer}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{A}(\cdot)\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{tk}', \mathsf{st}), \mathcal{A}(\cdot)$

5 : **else if** $\mathsf{corr}_i = 1$: /* *Corrupted sender* */

6 : $\quad \mathsf{Transfer}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{C}(sk_{\mathcal{U}_j}, (\mathsf{ide}, \mathsf{idp}))\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{tk}', \mathsf{st}), \mathcal{C}(\mathsf{tk}')$

7 : **else if** $\mathsf{corr}_j = 1$: /* *Corrupted receiver* */

8 : $\quad \mathsf{TK}_i \rightarrow \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), v\}$

9 : $\quad \mathsf{Transfer}\langle \mathcal{C}(sk_{\mathcal{U}_i}, \mathsf{tk}), \mathcal{C}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{A}(\cdot)\rangle \rightarrow \mathcal{C}(b), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{tk}', \mathsf{st}), \mathcal{A}(\cdot)$

10 : **else** : /* *No corruption* */

11 : $\quad \mathsf{TK}_i \rightarrow \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), v\}$

12 : $\quad \mathsf{Transfer}\langle \mathcal{C}(sk_{\mathcal{U}_i}, \mathsf{tk}), \mathcal{C}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{C}(sk_{\mathcal{U}_j}, (\mathsf{ide}, \mathsf{idp}))\rangle$
$\qquad \rightarrow \mathcal{C}(b), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{tk}', \mathsf{st}), \mathcal{C}(\mathsf{tk}')$

13 : **if** $b = \top$: $\mathsf{TK}_i \leftarrow \mathsf{TK}_i \cup \{(\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 0)\}, \mathsf{TK}_j \leftarrow \mathsf{TK}_j \cup \{(\mathsf{tk}', (\mathsf{ide}, \mathsf{idp}), 1)\}$

Figure 4.6: Description of the OTransfer oracle.

$\mathsf{OValidate}(\mathcal{SU}, sk_{\mathcal{V}}; i, (\mathsf{ide}, \mathsf{idp}))$

1 : $\mathcal{SU} \xrightarrow{p} \{\mathcal{U}_k\}_k, \mathcal{U}_i \xrightarrow{p} (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i, \mathsf{corr}_i)$

2 : **if** $\mathsf{corr}_i = 1$: /* *Corrupted user* */

3 : $\quad \mathsf{Validate}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{V}}, \mathsf{st})\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b, \mathsf{tk}, \mathsf{st})$

4 : $\quad$ **if** $b = \top : \mathcal{U}_i \leftarrow (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i \cup \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 0\}, \mathsf{corr}_i)$

5 : **else** : /* *Honest user* */

6 : $\quad \mathsf{TK}_i \rightarrow \{\mathsf{tk}', (\mathsf{ide}, \mathsf{idp}), b\}$

7 : $\quad \mathsf{Validate}\langle \mathcal{C}(sk_{\mathcal{U}_i}, \mathsf{tk}'), \mathcal{C}(sk_{\mathcal{V}}, \mathsf{st})\rangle \rightarrow \mathcal{C}(b), \mathcal{C}(b, \mathsf{tk}, \mathsf{st})$

8 : $\quad$ **if** $b = \top : \mathcal{U}_i \leftarrow (sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}, \mathsf{TK}_i \cup \{\mathsf{tk}, (\mathsf{ide}, \mathsf{idp}), 0\}, \mathsf{corr}_i)$

Figure 4.7: Description of the OValidate oracle.

distributor $\mathcal{D}$ is executed by the adversary. Note that in contrast with previous oracles, the challenger provides $sk_{\mathcal{U}_b}$ the secret key of the user $\mathcal{U}_b$ generated by the challenger.

$\mathsf{OPurchase}_b(\mathsf{TK}, sk_{\mathcal{U}_b}; (\mathsf{ide}, \mathsf{idp}))$

1 : $\mathsf{Purchase}\langle \mathcal{C}(sk_{\mathcal{U}_b}, (\mathsf{ide}, \mathsf{idp})), \mathcal{A}(\cdot,\cdot)\rangle \rightarrow \mathcal{C}(\mathsf{tk}_{|\mathsf{TK}|}), \mathcal{A}(\cdot)$

2 : **if** $\mathsf{tk}_{|\mathsf{TK}|} \neq \perp : \mathsf{TK} \leftarrow \mathsf{TK} \cup \{(\mathsf{ide}, \mathsf{idp}, \mathsf{tk}_{|\mathsf{TK}|})\}$

Figure 4.8: Description of $\mathsf{OPurchase}_b$ oracle.

The ticket refund oracle Refund_b depicted in Figure 4.9 expects from the adversary an index i designating the ticket tk_i to refund. The challenger provides to the oracle, in addition to the set of tickets TK and the secret key $sk_{\mathcal{U}_b}$ a seat identifier $(\mathsf{ide}, \mathsf{idp})$. This seat identifier is used to prevent a challenge ticket to be refunded. This is a necessary condition to prevent trivial attack consisting to verify if a runs owns a ticket by checking the output of the Refund protocol (or any other protocol).

$\mathsf{ORefund}_b(\mathsf{TK}, sk_{\mathcal{U}_b}, (\mathsf{ide}, \mathsf{idp}); i)$

1 : $\mathsf{TK} \xrightarrow{p} \{(\mathsf{ide}_j, \mathsf{idp}_j, \mathsf{tk}_j)\}_j$
2 : **if** $(\mathsf{ide}, \mathsf{idp}) = (\mathsf{ide}_i, \mathsf{idp}_i)$: **return** $\perp$ /* *Prevent trivial distinguishing attack* */
3 : $\mathsf{Refund}\langle \mathcal{C}(sk_{\mathcal{U}_b}, \mathsf{tk}_i), \mathcal{A}(\cdot,\cdot)\rangle \rightarrow \mathcal{C}(b), \mathcal{A}(\cdot,\cdot,\cdot)$

Figure 4.9: Description of $\mathsf{ORefund}_b$ oracle.

The ticket transfer oracle $\mathsf{OTransfer}_b$ depicted in Figure 4.10 expects from the adversary an index i designating the ticket to transfer, a role `role` and a seat identifier $(\mathsf{ide}'_i, \mathsf{idp}'_i)$. The provided role specifies the behaviour of the oracle during the transfer, either simulating a user transferring a ticket or simulating a user receiving a ticket. In case where the user is expected to transfer its i-th ticket, the oracle runs the ticket transfer where both the system and the user receiving the ticket are executed by the adversary. The second case is similar except that the oracle simulates the user receiving the ticket instead. The oracle rejects ticket transfer request from the adversary when the transferred ticket is the challenge ticket, explicitly handled by the challenger.

$\mathsf{OTransfer}_b(\mathsf{TK}, sk_{\mathcal{U}_b}, (\mathsf{ide}, \mathsf{idp}); i, \mathtt{role}, (\mathsf{ide}'_i, \mathsf{idp}'_i))$

1 : **if** role $= \mathtt{sell}$:
2 : $\quad \mathsf{TK} \xrightarrow{p} \{(\mathsf{ide}_j, \mathsf{idp}_j, \mathsf{tk}_j)\}_j$
3 : $\quad$ **if** $(\mathsf{ide}, \mathsf{idp}) = (\mathsf{ide}_i, \mathsf{idp}_i)$: **return** $\perp$
4 : $\quad \mathsf{Transfer}\langle \mathcal{C}(sk_{\mathcal{U}_b}, \mathsf{tk}_i), \mathcal{A}(\cdot,\cdot), \mathcal{A}(\cdot)\rangle \rightarrow \mathcal{C}(b), \mathcal{A}(\cdot,\cdot,\cdot), \mathcal{A}(\cdot)$
5 : **if** role $= \mathtt{buy}$:
6 : $\quad \mathsf{Transfer}\langle \mathcal{A}(\cdot,\cdot), \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{U}_b}, (\mathsf{ide}'_i, \mathsf{idp}'_i))\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{A}(\cdot,\cdot,\cdot), \mathcal{C}(\mathsf{tk}_{|\mathsf{TK}|})$
7 : $\quad$ **if** $\mathsf{tk}_{|\mathsf{TK}|} \neq \perp$: $\mathsf{TK} \leftarrow \mathsf{TK} \cup \{(\mathsf{ide}'_i, \mathsf{idp}'_i, \mathsf{tk}_{|\mathsf{TK}|})\}$

Figure 4.10: Description of $\mathsf{OTransfer}_b$ oracle.

The ticket validation oracle $\mathsf{OValidate}_b$ depicted in Figure 4.11 expects from the adversary an index i designating the i-th ticket that $\mathcal{U}_b$ is asked to validate. The oracle runs the ticket validation protocol with the system simulated by the adversary. Note that the challenger simulating the oracle rejects ticket validation attempts from the adversary when the designated ticket is the challenge seat.

$\mathsf{OValidate}_b(\mathsf{TK}, sk_{\mathcal{U}_b}, (\mathsf{ide}, \mathsf{idp}); i)$

1 : $\mathsf{TK} \xrightarrow{p} \{(\mathsf{ide}_j, \mathsf{idp}_j, \mathsf{tk}_j)\}_j$
2 : **if** $(\mathsf{ide}, \mathsf{idp}) = (\mathsf{ide}_i, \mathsf{idp}_i)$: **return** $\perp$
3 : $\mathsf{Validate}\langle \mathcal{C}(sk_{\mathcal{U}_b}, \mathsf{tk}_i), \mathcal{A}(\cdot,\cdot)\rangle \rightarrow \mathcal{C}(b), \mathcal{A}(\cdot,\cdot,\cdot)$

Figure 4.11: Description of $\mathsf{OValidate}_b$ oracle.

Security Experiments Description

Unforgeability The *unforgeability* of a ticket depicted in Figure 4.12 prevents an adversary $\mathcal{A}$ from creating a new valid ticket. This property is described in $\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}$. The

adversary wins the unforgeability game if it produces a ticket not been produced by the system (*i.e.,* $\mathsf{tk} \notin \mathsf{TK}$) that is accepted either for a refund, a transfer, or a validation. In the security experiment, these winning conditions are represented respectively by the bit b_0 and b_1. This property must be ensured for any PPT adversary, with read-only access to the shared state and the oracles OCreateUser, OCorruptUser, OLeakState, OPurchase, ORefund, OTransfer and OValidate, corresponding to possible actions of the users and represented in the set of oracles $\mathcal{O}$.

$\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}(1^\lambda)$

1 : $(sk_{\mathcal{D}}, pk_{\mathcal{D}}), (sk_{\mathcal{T}}, pk_{\mathcal{T}}), (sk_{\mathcal{V}}, pk_{\mathcal{V}}) \leftarrow \mathcal{D}\mathsf{KeyGen}/\mathcal{T}\mathsf{KeyGen}/\mathcal{V}\mathsf{KeyGen}(1^\lambda)$
2 : $\mathcal{SU} \leftarrow \emptyset, \mathsf{st} \leftarrow \emptyset$
3 : $(\mathsf{tk}, \mathsf{Alg}) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_{\mathcal{D}}, pk_{\mathcal{T}}, pk_{\mathcal{V}})$
4 : $b_0 \leftarrow \mathsf{tk} \notin \mathsf{TK}$ // *Ticket not produced by challenger*
5 : **if** $\mathsf{Alg} \in \{\mathsf{Refund}, \mathsf{Validate}\}$:
6 : $\quad sk \leftarrow sk_{\mathcal{D}}$ **if** $\mathsf{Alg} = \mathsf{Refund}$ **else** $sk \leftarrow sk_{\mathcal{V}}$
7 : $\quad \mathsf{Alg}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk, \mathsf{st})\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b_1, \mathsf{tk}, \mathsf{st})$
8 : **else** : $\mathsf{Transfer}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{A}(\cdot)\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b_1, (\mathsf{ide}_1, \mathsf{idp}_1), \mathsf{tk}, \mathsf{tk}', \mathsf{st}), \mathcal{A}(\cdot)$
9 : **return** $b_0 \wedge b_1$

Figure 4.12: Description of the unforgeability experiment.

Ticket privacy. The *ticket privacy* depicted in Figure 4.13 prevents from an adversary $\mathcal{A}$, external to the system, stealing a ticket from a designated user. The adversary has the capability to generate, manipulate through oracles OCreateUser, OCorruptUser, OLeakState, OPurchase, ORefund, OTransfer and OValidate contained in the $\mathcal{O}$ set. The ticket privacy is focused against entities that are external to the system. In the associated experiment, $\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}$, user $\mathcal{U}_1$ is the adversary's target. $\mathcal{U}_1$ purchases a ticket tk and $\mathcal{A}$ wins if (1) the purchase went through, (2) $\mathcal{A}$ outputted a ticket tk^* such that $\mathsf{tk}^* = \mathsf{tk}$ and (3) $\mathcal{A}$ did not corrupt $\mathcal{U}_1$. During this process, the adversary has *read-only* access to the shared state at any point during the experiment. The challenger simulates the user purchasing the ticket targeted for recovery by the adversary, but also $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{V}$, otherwise making the ticket privacy trivially broken as the system requires the ticket for verification purposes.

No-double-spending Once purchased, a ticket should be usable only once: The ticket can be refund once, transferred once or validated once. In other words, as done in our security model, none of Refund, Validate or Transfer executed by the challenger against a corrupted user would accept the same ticket twice. This notion differs from what has been formalised in *e-cash* [BCFK15]. Taking as example the protocol introduced by Baldimtsi *et al.* [BCFK15], their model allows execution of a function Spend twice for the same coin and postpone the double spending verification to a second algorithm call Deposit. Applied to ticketing, since the verification occurs after the ticket spent, the consequence for users is the possibility to buy already spent ticket, leading to a ticket rejection (during the second execution of Deposit of the same ticket). Following our notion, an honest client should not acquire a already transferred ticket.

$\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}(1^\lambda)$

1 : $(sk_{\mathcal{D}}, pk_{\mathcal{D}}), (sk_{\mathcal{T}}, pk_{\mathcal{T}}), (sk_{\mathcal{V}}, pk_{\mathcal{V}}) \leftarrow \mathcal{D}\mathsf{KeyGen}/\mathcal{T}\mathsf{KeyGen}/\mathcal{V}\mathsf{KeyGen}(1^\lambda)$

2 : $(sk_{\mathcal{U}_1}, pk_{\mathcal{U}_1}) \leftarrow \mathcal{U}\mathsf{KeyGen}(1^\lambda)$

3 : $\mathsf{st} \leftarrow \emptyset, \mathcal{SU} \leftarrow \mathcal{U}_1 = \{(sk_{\mathcal{U}_1}, pk_{\mathcal{U}_1}, \perp, 0)\}$

4 : $(\mathsf{ide}, \mathsf{idp}) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_{\mathcal{U}}, pk_{\mathcal{D}}, pk_{\mathcal{T}}, pk_{\mathcal{V}})$

5 : $\mathsf{Purchase}\langle \mathcal{C}(sk_{\mathcal{U}}, (\mathsf{ide}, \mathsf{idp})), \mathcal{C}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{C}(\mathsf{tk}), \mathcal{C}(b, (\mathsf{ide}, \mathsf{idp}), \mathsf{tk}, \mathsf{st})$

6 : $\mathsf{tk}^* \leftarrow \mathcal{A}^{\mathcal{O}}(pk_{\mathcal{U}}, pk_{\mathcal{D}}, pk_{\mathcal{T}}, pk_{\mathcal{V}})$

7 : $\mathcal{SU} \xrightarrow{p} \{\mathcal{U}_k\}_k, \mathcal{U}_1 \xrightarrow{p} (sk_{\mathcal{U}_1}, pk_{\mathcal{U}_1}, \mathsf{TK}_1, \mathsf{corr}_1)$

8 : **return** $b \wedge (\mathsf{tk} \stackrel{?}{=} \mathsf{tk}^*) \wedge (\mathsf{corr}_1 \stackrel{?}{=} 0)$

Figure 4.13: Description of the ticket privacy experiment.

The security notion preventing the adversary to double-transfer a ticket, and more generally to interact twice with the system using the same ticket is denoted as Double-Spending (DS), formally introduced in the experiment $\mathsf{Exp}^{\mathsf{DS}}_{\mathcal{A}}$ depicted in Figure 4.14: The challenger simulates the system and allows $\mathcal{A}$ to invoke actions through OCreateUser, OCorruptUser, OLeakState, OPurchase, ORefund, OTransfer and OValidate oracles, thereby providing a view of the shared state. The adversary subsequently outputs a ticket tk and two actions, Alg_1 and Alg_2, selected from {Validate, Refund, Transfer}. Remark that the two specified algorithms Alg_1 and Alg_2 are not required to be the same, modelling every possible combination of attack. The adversary succeeds if the following three winning conditions are respected: (1) Both algorithm executions are successful, (2) the tickets tk_1 and tk_2 presented to the validator match the committed ticket tk, and (3) the tickets share identical identifiers ide and idp. Both executions are dependent since the shared state st is updated after the first execution and used the second time.

$\mathsf{Exp}^{\mathsf{DS}}_{\mathcal{A}}(1^\lambda)$

1 : $(sk_{\mathcal{D}}, pk_{\mathcal{D}}), (sk_{\mathcal{T}}, pk_{\mathcal{T}}), (sk_{\mathcal{V}}, pk_{\mathcal{V}}) \leftarrow \mathcal{D}\mathsf{KeyGen}/\mathcal{T}\mathsf{KeyGen}/\mathcal{V}\mathsf{KeyGen}(1^\lambda)$

2 : $\mathcal{SU} \leftarrow \emptyset, \mathsf{st} \leftarrow \emptyset$

3 : $(\mathsf{tk}, \mathsf{Alg}_1, \mathsf{Alg}_2) \leftarrow \mathcal{A}^{\mathcal{O}}(pk_{\mathcal{D}}, pk_{\mathcal{T}}, pk_{\mathcal{V}})$

4 : **for** $k \in \{1, 2\}$:

5 : **if** $\mathsf{Alg}_k = \mathsf{Refund}$: $\mathsf{Refund}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{D}}, \mathsf{st})\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b_k, \mathsf{tk}_k, \mathsf{st})$

6 : **if** $\mathsf{Alg}_k = \mathsf{Validate}$: $\mathsf{Validate}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{V}}, \mathsf{st})\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b_k, \mathsf{tk}_k, \mathsf{st})$

7 : **else** : $\mathsf{Transfer}\langle \mathcal{A}(\cdot,\cdot), \mathcal{C}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{A}(\cdot)\rangle \rightarrow \mathcal{A}(\cdot), \mathcal{C}(b_1, (\mathsf{ide}_k, \mathsf{idp}_k), \mathsf{tk}_k, \cdot, \mathsf{st}), \mathcal{A}(\cdot)$

8 : **return** $b_1 \wedge b_2 \wedge ((\mathsf{ide}_1, \mathsf{idp}_1) \stackrel{?}{=} (\mathsf{ide}_2, \mathsf{idp}_2)) \wedge (\mathsf{tk} \stackrel{?}{=} \mathsf{tk}_1 \stackrel{?}{=} \mathsf{tk}_2 \neq \perp)$

Figure 4.14: Description of the double-spending experiment.

Anonymity. To model the properties of non-nominative physical tickets, an ETS should preserve the anonymity of a ticket holder $\mathcal{U}$ against the system. This is modelled with two levels of anonymity properties called respectively *pseudonymity* and *unlinkability*. In a nutshell, pseudonymity ensures that the identity of users cannot be associated to an identity. In contrast, unlinkability models a stronger anonymity property, in which a ticket could not be linked by the system as coming from the same holder nor be linked

to a user. We stress that unlinkability is not intended to prevent a ticket to be tracked by the system, but rather to put forward the independence of protocols in which a ticket is inputted. Said differently, our definition of unlinkability guarantees that two distinct tickets cannot be linked as being purchased or used by the same user. For both pseudonymity and unlinkability, the adversary controls the system *i.e.,* $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{V}$ and thus can generate and control other users. For the sake of clarity in the following experiments, we introduce a subroutine $\mathsf{DefineAnoOracles}_b$ depicted in Figure 4.15 which given the secret key $sk_{\mathcal{U}}$ of a user, a set of tickets TK and a challenge seat identifier $(\mathsf{ide}, \mathsf{idp})$, returns a configured set of oracles.

$\mathsf{DefineAnoOracles}(sk_{\mathcal{U}}, \mathsf{TK}, (\mathsf{ide}, \mathsf{idp}))$

1 : $\mathcal{O}_1 \leftarrow \mathsf{OPurchase}_b(\mathsf{TK}, sk_{\mathcal{U}}; \cdot)$
2 : $\mathcal{O}_2 \leftarrow \mathsf{ORefund}_b(\mathsf{TK}, sk_{\mathcal{U}}, (\mathsf{ide}, \mathsf{idp}); \cdot)$
3 : $\mathcal{O}_3 \leftarrow \mathsf{OTransfer}_b(\mathsf{TK}, sk_{\mathcal{U}}, (\mathsf{ide}, \mathsf{idp}); \cdot, \cdot, \cdot)$
4 : $\mathcal{O}_4 \leftarrow \mathsf{OValidate}_b(\mathsf{TK}, sk_{\mathcal{U}}, (\mathsf{ide}, \mathsf{idp}); \cdot)$
5 : **return** $\{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4\}$

Figure 4.15: Description of the $\mathsf{DefineAnoOracles}$ subroutine.

Experiment $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$ depicted in Figure 4.16 models *pseudonymity*: The challenger generates two users, denoted as $\mathcal{U}_0$ and $\mathcal{U}_1$, represented by their respective public keys $pk_{\mathcal{U}_0}$ and $pk_{\mathcal{U}_1}$. $\mathcal{A}$ can invoke oracles $\mathsf{OPurchase}_b$, $\mathsf{ORefund}_b$, $\mathsf{OTransfer}_b$, and $\mathsf{OValidate}_b$, for a given $b \in \{0, 1\}$. Note that in pseudonymity, there is no challenge seat, justifying the $\perp$ symbol in the oracles declaration. For $\mathcal{A}$ to succeed, it must produce a bit b^* that matches the input bit b provided in the oracles. Then determining which one of $\mathcal{U}_0$ and $\mathcal{U}_1$ responded to the oracle calls.

$\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda)$

1 : $(sk_{\mathcal{U}_0}, pk_{\mathcal{U}_0}), (sk_{\mathcal{U}_1}, pk_{\mathcal{U}_1}) \leftarrow \mathcal{U}\mathsf{KeyGen}(1^\lambda)$
2 : $\mathsf{TK} \leftarrow \emptyset, b \xleftarrow{\$} \{0, 1\}$
3 : $\mathcal{O}_b \leftarrow \mathsf{DefineAnoOracles}(sk_{\mathcal{U}_b}, \mathsf{TK}, \perp)$
4 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_b}(pk_{\mathcal{U}_0}, pk_{\mathcal{U}_1})$
5 : **return** $b \stackrel{?}{=} b^*$

Figure 4.16: Description of the pseudonymity experiment.

Experiment $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ depicted in Figure 4.17 models *unlinkability*, in which the challenger once again simulates the behaviours of two users, denoted as $\mathcal{U}_0$ and $\mathcal{U}_1$. Initially, the adversary interacts with both users by making calls to the oracles $\mathsf{OPurchase}_b$, $\mathsf{ORefund}_b$, $\mathsf{OTransfer}_b$ and $\mathsf{OValidate}_b$ for every bit $b \in \{0, 1\}$. Following this preliminary phase, the adversary provides a seat identifier $(\mathsf{ide}, \mathsf{idp})$. Then, given this seat identifier, the challenger randomly chooses a bit $b \in \{0, 1\}$ and purchases a ticket the designated seat in the belief of $\mathcal{U}_b$. Then, it executes an action from the set $\{\mathsf{Validate}, \mathsf{Refund}, \mathsf{Transfer}\}$ for the obtained ticket. The adversary succeeds in the experiment if it can correctly guess which user performed the action.

$\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}(1^\lambda)$

1 : $(sk_{\mathcal{U}_0}, pk_{\mathcal{U}_0}), (sk_{\mathcal{U}_1}, pk_{\mathcal{U}_1}) \leftarrow \mathcal{U}\mathsf{KeyGen}(1^\lambda)$
2 : $\mathcal{O}_0, \mathcal{O}_1 \leftarrow \mathsf{DefineAnoOracles}(sk_{\mathcal{U}_0}, \mathsf{TK}, \bot), \mathsf{DefineAnoOracles}(sk_{\mathcal{U}_1}, \mathsf{TK}, \bot)$
3 : $(\mathsf{ide}, \mathsf{idp}) \leftarrow \mathcal{A}^{\mathcal{O}_0, \mathcal{O}_1}(pk_{\mathcal{U}_0}, pk_{\mathcal{U}_1})$
4 : $\mathcal{O}_0, \mathcal{O}_1 \leftarrow \mathsf{DefineAnoOracles}(sk_{\mathcal{U}_0}, \mathsf{TK}, (\mathsf{ide}, \mathsf{idp})), \mathsf{DefineAnoOracles}(sk_{\mathcal{U}_1}, \mathsf{TK}, (\mathsf{ide}, \mathsf{idp}))$
5 : $b \xleftarrow{\$} \{0, 1\}$
6 : $\mathsf{Purchase}\langle \mathcal{C}(sk_{\mathcal{U}_b}, (\mathsf{ide}, \mathsf{idp})), \mathcal{A}(\cdot, \cdot)\rangle \rightarrow \mathcal{C}(\mathsf{tk}), \mathcal{A}(\cdot, \cdot, \cdot)$
7 : $\mathsf{Alg} \leftarrow \mathcal{A}^{\mathcal{O}_0, \mathcal{O}_1}()$
8 : **if** $\mathsf{Alg} \in \{\mathsf{Refund}, \mathsf{Validate}\}$: $\mathsf{Alg}\langle \mathcal{C}(sk_{\mathcal{U}_b}, (\mathsf{ide}, \mathsf{idp})), \mathcal{A}(\cdot, \cdot)\rangle \rightarrow \mathcal{C}(\mathsf{tk}), \mathcal{A}(\cdot, \cdot, \cdot)$
9 : **else** : $\mathsf{Transfer}\langle \mathcal{C}(sk_{\mathcal{U}_b}, \mathsf{tk}), \mathcal{A}(\cdot, \cdot), \mathcal{A}(\cdot, \cdot, \cdot)\rangle \rightarrow \mathcal{C}(b), \mathcal{A}(\cdot, \cdot, \cdot, \cdot), \mathcal{A}(\cdot)$
10 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_0, \mathcal{O}_1}(b)$
11 : **return** $b \stackrel{?}{=} b^*$

Figure 4.17: Description of the unlinkability experiment.

Auditability. We mitigate anonymity by introducing *Auditable E-Ticket scheme* enabling user's identity recovery under the supervision of a judge. Auditable E-Ticket scheme achieves the previous properties but Audit requests for $\mathcal{U}_0$ and $\mathcal{U}_1$ are not allowed for the adversary during the pseudonymity and unlinkability experiments.

Definition 26 (Auditable ETS)**.** An *Auditable E-Ticket scheme* ETS is an E-Ticket scheme increased with the following algorithms:

- $\underline{\mathcal{J}\mathsf{KeyGen}(1^\lambda) \rightarrow (sk_{\mathcal{J}}, pk_{\mathcal{J}})}$: Given 1^λ, outputs a key pair.
- $\underline{\mathsf{Audit}(sk_{\mathcal{J}}, \mathsf{tk}) \rightarrow pk_{\mathcal{U}}}$: Given a secret key $sk_{\mathcal{J}}$ and a ticket tk, returns the public key associated to user $\mathcal{U}$.

Definition of Anonymous Payment

Ticket purchasing, transferring, and refunding necessitate payments. However, conventional online payment methods (such as card payments based on the EMV protocol [EMV11]) disclose the user's identity, posing a challenge to ensure user anonymity in a ETS setting. Therefore, the anonymity within an ETS protocol is contingent on the anonymity provided by the underlying payment protocol. Rather than omitting the payment process, as observed in [HCS+21], we've opted to incorporate the payment protocol. Specifically, to maintain user anonymity, the incorporated payment, termed *Anonymous Payment* and utilised as a foundational component, must enable participants to make payments without disclosing their identities. This building block is voluntary generic to let any anonymous payment method, to be plugged in our protocol.

Definition 27 (Anonymous Payment, AP)**.** An anonymous payment protocol AP is defined by the tuple $(\mathsf{KeyGen}, \mathsf{GenToken}, \mathsf{AP})$ where:

- $\underline{\mathsf{KeyGen}(1^\lambda) \rightarrow (stk, spk, pd)}$: Given the unary representation of the security parameter 1^λ, outputs (stk, spk, pd) where stk is the secret token generation key, spk is the secret payment key used to approve payment and pd is the publicly accessible data. This last parameter, not used later, is intended to model existing public (and static) parameters involved by the payment protocol.

- $\underline{\mathsf{GenToken}(stk) \to tkn}$: Given the secret token generation key stk outputs a payment token tkn.
- $\underline{\mathsf{AP}\langle \mathcal{U}_1(spk_1, tkn), \mathcal{U}_2(spk_2, tkn)\rangle \to \mathcal{U}_1(b), \mathcal{U}_2(b)}$: Let two users $\mathcal{U}_1$ and $\mathcal{U}_2$ where $\mathcal{U}_1$ proceeds to a payment with $\mathcal{U}_2$. On a hand, the user $\mathcal{U}_1$ obtains as an input the secret payment key spk_1 and a token tkn generated via the token generation algorithm $\mathsf{GenToken}(stk)$ inputted with an arbitrary secret token generation key stk. On the other hand, the user $\mathcal{U}_2$ obtains as an input the secret payment key spk_2 as well as the token tkn. At the end, a success bit b is returned, confirming the success of the payment. In this model, we allow the payment to be performed both from $\mathcal{U}_1$ to $\mathcal{U}_2$ and from $\mathcal{U}_2$ to $\mathcal{U}_1$, even with a *negative* amount (which is just a refund). This choice is motivated to keep our definition simple and flexible. A valid payment occurs when both parties agree on the payment, whose the setting of the payment (*e.g.,* source of the payment, destination of the payment, the amount) is contained in the payment token tkn.

In order to be agnostic of the used anonymous payment scheme, we employ generic security definitions for anonymous payment: *Unlinkability* ensuring that two transactions from the same user are not linkable, *pseudonymity* ensuring that a user identity remains undisclosed., but also *unforgeability* ensuring that it is infeasible to retrieve a token to accept a payment. During the experiments, the adversary has access to the oracle OAP in which the adversary provides a payment token tkn and then runs the payment protocol with the user:

Oracle $\mathsf{OPay}(spk, stk; tkn)$

1 : **if** $tkn = \perp$ **then** $tkn \leftarrow \mathsf{GenToken}(stk)$
2 : $\mathsf{AP}\langle \mathcal{C}(spk, tkn), \mathcal{A}(tkn)\rangle$

Description of Pseudonymity. The experiment of pseudonymity depicted in Figure 4.18 starts by the generation of two keys for two distinct users and provide the both public data to the adversary. At any time, the adversary has access to the oracle to interact with both users. At some point, the adversary is asked to produce a payment token tkn that is used to generate a payment with the user $\mathcal{U}_b$ for a bit b randomly chosen by the challenger. Ultimately, the adversary is asked to produce a prediction bit b^* used by the challenger, returning 1 if the prediction bit equals the random bit b.

$\mathsf{Exp}^{\mathsf{APPse}}_{\mathcal{A}}(1^\lambda)$

1 : $(stk_0, spk_0, pd_0), (stk_1, spk_1, pd_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2 : $b \xleftarrow{\$} \{0, 1\}$
3 : $b^* \leftarrow \mathcal{A}^{\mathsf{OPay}(spk_b, stk_b; \cdot)}(pd_0, pd_1)$
4 : **return** $b \stackrel{?}{=} b^*$

Figure 4.18: Description of pseudonymity for anonymous payment.

Definition 28 (Pseudonymity for AP)**.** An anonymous payment scheme AP defined by the triplet of PPT algorithm $(\mathsf{KeyGen}, \mathsf{GenToken}, \mathsf{AP})$ ensures pseudonymity if for every security parameter λ and any PPT adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{APPse}} = \left| \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{APPse}}(1^{\lambda}) \to 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Description of Unlinkability. The unlinkability property depicted in Figure 4.19 works similarly to the pseudonymity property, except that the challenger generates two distinct users $\mathcal{U}_0$ and $\mathcal{U}_1$, and allows the adversary to interact with *both* users using dedicated payment oracles. Then, the challenger randomly selects a random bit b, and allows the adversary to interact only with $\mathcal{U}_b$ using the payment oracle OAP inputted with parameters of the user $\mathcal{U}_b$. Eventually, the adversary responds with a prediction bit b^*, and wins the experiment if the prediction bit b^* equals the randomly chosen bit b.

$\mathsf{Exp}_{\mathcal{A}}^{\mathsf{APUnl}}(1^{\lambda})$

$(stk_0, spk_0, pd_0), (stk_1, spk_1, pd_1) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$b \xleftarrow{\$} \{0,1\}$
$\mathcal{O}_0, \mathcal{O}_1 \leftarrow \mathsf{OPay}(spk_0, stk_0; \cdot), \mathsf{OPay}(spk_1, stk_1; \cdot)$
$b \xleftarrow{\$} \{0,1\}$
$tkn \leftarrow \mathcal{A}^{\mathcal{O}_0, \mathcal{O}_1}(pd_0, pd_1)$
if $tkn = \perp$ **then** $tkn \leftarrow \mathsf{GenToken}(stk)$
$\mathsf{AP}\langle \mathcal{C}(spk_b, tkn), \mathcal{A}(tkn) \rangle$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_0, \mathcal{O}_1}()$
return $b \stackrel{?}{=} b^*$

Figure 4.19: Description of unlinkability for anonymous payment.

Definition 29 (Unlinkability for AP)**.** An anonymous payment scheme AP defined by the triplet of PPT algorithm $(\mathsf{KeyGen}, \mathsf{GenToken}, \mathsf{AP})$ ensures unlinkability if for every security parameter λ and any PPT adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{APUnl}} = \left| \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{APUnl}}(1^{\lambda}) \to 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Description of Unforgeability. The unforgeability property starts with the challenger generating a payment key pair, and allows the adversary to interact with the generated payment key using the payment oracle OPay. Later, the challenger generates a payment token tkn using the secret token generation key stk. Finally, without having any information on the token, the adversary is asked to proceed to a payment with the challenger. The resulting success bit obtained at the end of the payment protocol execution is returned as the output bit of the unforgeability experiment. In other words, the adversary breaks the unforgeability property of the anonymous payment if it is possible for the adversary to force the payment without having any information on the token

and hence on the current transaction. This property is particularly useful to ensure authentication of the payment between both parties.

$\mathsf{Exp}^{\mathsf{APUF}}_{\mathcal{A}}(1^\lambda)$

1 : $(stk, spk, pd) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2 : $\mathcal{O} \leftarrow \mathsf{OPay}(spk, stk; \cdot)$
3 : $tkn \leftarrow \mathsf{GenToken}(stk)$
4 : $\mathsf{AP}\langle \mathcal{C}(spk, tkn), \mathcal{A}^{\mathcal{O}}(pd)\rangle \rightarrow \mathcal{C}(b)$
5 : **return** b

Figure 4.20: Description of unforgeability for anonymous payment.

Definition 30 (Unforgeability for AP)**.** An anonymous payment scheme AP defined by the triplet of PPT algorithm $(\mathsf{KeyGen}, \mathsf{GenToken}, \mathsf{AP})$ ensures unforgeability if for every security parameter λ and any PPT adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}^{\mathsf{APUF}}_{\mathcal{A}} = \Pr\left[\mathsf{Exp}^{\mathsf{APUF}}_{\mathcal{A}}(1^\lambda) \rightarrow 1\right] \leq \mathsf{negl}(\lambda)$$

We do not cover how to plug anonymous payment formally, since deciding which payment protocol to use is highly dependent of the context where our ticketing system is deployed. One straightforward yet inefficient payment method that maintains user anonymity is through a one-time payment card like PaySafeCard [Pay18], purchasable locally with physical currency. Blockchain based methods such as Monero [Sab13] provides payment unlinkability and pseudonymity using one-time payment identity in a sense that may differ slightly from our definitions. Another example uses cash or its digital counterpart like CashApp [Cas18], where payments are executed using randomly generated identifiers. Anonymous transferable e-cash [BCFK15] also matches our requirements, providing a solution to exchange anonymously coins between users.

We stress that the anonymity in Applause and Spotlight directly relies on the anonymity of the payment scheme. A secure anonymous payment scheme perfectly fitting our definition allows our protocol to ensure the anonymity of users. At the opposite, a payment scheme ensuring only pseudonymity allows our protocol to ensure only pseudonymity and not unlinkability. An already-in-used protocol, the EMV specification for *Tokenization* [EMV22], provides pseudonymity but falls short in providing unlinkability. If the context in which our protocol is deployed does not require unlinkability, one may relies on such a payment method.

4.3 Anonymous Ticketing System with Applause

In the protocol, the participants interact using personal encryption keys $(sk^{\mathsf{PKE}}, pk^{\mathsf{PKE}})$ and signature keys $(sk^{\mathsf{Sign}}, pk^{\mathsf{Sign}})$ and proceed to payment using dedicated keys denoted $(stk^{\mathsf{AP}}, spk^{\mathsf{AP}}, pd^{\mathsf{AP}})$. A shared state st is updated by the ticket distributor $\mathcal{D}$, the transfer authority $\mathcal{T}$ and the validator $\mathcal{V}$, acting as a blacklist. To purchase a ticket through $\mathsf{Purchase}$, the user first selects an event and a free seat $(\mathsf{ide}, \mathsf{idp})$, and it also draws a nonce r_c. It obtains a signature σ_c on the hash of the triple $(\mathsf{ide}, \mathsf{idp}, r_c)$. The event

and seat identifiers $(\mathsf{ide}, \mathsf{idp})$ along the nonce r_c and the signature σ_c represent the ticket of the client. Hence a ticket tk is defined as $((\mathsf{ide}, \mathsf{idp}, r_c), \sigma_c)$. The ticket validation process involves the showing of these two values to the validator and the agreement on a challenge value through a physical channel to authenticate the ticket owner. The first showing is similar to the Refund process before the client is given back its funds or during the $\mathsf{Transfer}$, as the transfer authority $\mathcal{T}$ refunds the owner of the ticket, and performs the ticket purchase protocol with the new owner of the ticket.

4.3.1 Description of Applause

Subroutines. Before explaining our protocol, we first describe subroutines that we use to increase the readability and the clarity of the overall protocol description. In particular, we introduce three suggestive algorithms KeyGen, $\mathsf{CheckTk}$ and $\mathsf{RandKey}$, allowing respectively to generate a key pair, to verify the validity of a ticket and finally to randomize a key pair.

- $\underline{\mathsf{KeyGen}(1^\lambda)}$: This algorithm starts by generating a signature key pair denoted $(sk^{\mathsf{Sign}}, pk^{\mathsf{Sign}}) \leftarrow \mathsf{Sign.KeyGen}(1^\lambda)$, as well as an encryption key pair denoted $(sk^{\mathsf{PKE}}, pk^{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. In addition, a payment key composed of the triplet $(stk^{\mathsf{AP}}, spk^{\mathsf{AP}}, pd^{\mathsf{AP}}) \leftarrow \mathsf{AP.KeyGen}(1^\lambda)$ is generated. Finally, it returns the key pair (sk, pk) where the secret key sk is composed of the triplet $(sk^{\mathsf{Sign}}, sk^{\mathsf{PKE}}, stk^{\mathsf{AP}}, spk^{\mathsf{AP}})$ and where the public key pk is composed of the triplet $(pk^{\mathsf{Sign}}, pk^{\mathsf{PKE}}, pd^{\mathsf{AP}})$.

The ticket verification subroutine $\mathsf{CheckTk}$ expects as an input a ticket tk and the current shared state st (containing only hashes c) checks if the provided ticket tk should be considered valid. This subroutine, denoted $\mathsf{CheckTk}$ works as follows:

- $\underline{\mathsf{CheckTk}(\mathsf{tk}, \mathsf{st})}$: First, this algorithm parses the provided ticket tk defined as $((\mathsf{ide}, \mathsf{idp}, r_c), \sigma_c)$. Then, it computes the hash c as $H(\mathsf{ide}, \mathsf{idp}, r_c)$ using the hash function H over the event and seat identifiers ide and idp, but also the random r_c. Finally, the algorithm checks if one among the ticket distributor $\mathcal{D}$ and the transfer authority $\mathcal{T}$ is authenticating this ticket, by executing both the signature verification function $\mathsf{Sign.Verif}(pk_{\mathcal{D}}^{\mathsf{Sign}}, \sigma_c, c)$ using the signature public key of $\mathcal{D}$, and the signature verification function $\mathsf{Sign.Verif}(pk_{\mathcal{T}}^{\mathsf{Sign}}, \sigma_c, c)$ using the verification public key $pk_{\mathcal{T}}^{\mathsf{Sign}}$. This double signature verification is required since a ticket can be authenticated by the ticket distributor $\mathcal{D}$ during the purchase of the ticket, but also by the transfer authority $\mathcal{T}$ during the transfer of the ticket. If one of the two signature verifications succeeds, then the algorithm checks that the hash c is not contained in the set of invalid tickets st provided as an input, by checking if $c \notin \mathsf{st}$. If all the previously mentioned verifications succeed, then the algorithm returns $\top$ meaning that the ticket is valid, otherwise returns $\bot$ meaning that the ticket is invalid.

The randomization key algorithm $\mathsf{RandKey}$ is especially used in our protocol to randomize the key pair $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ of a user $\mathcal{U}$, providing randomized key that are statistical indistinguishability from the original key pair of the user. We notify that in Applause, a user key pair outputted by the user key generation algorithm $\mathcal{U}\mathsf{KeyGen}$ exactly corresponds to the key pair outputted by the KeyGen algorithm.

- $\underline{\mathsf{RandKey}(sk_{\mathcal{U}}, pk_{\mathcal{U}}) \rightarrow (sk_{\mathcal{U}'}, pk_{\mathcal{U}'})}$: Let $sk_{\mathcal{U}}$ be the secret key of user $\mathcal{U}$ composed of the signature secret siganture key $sk_{\mathcal{U}}^{\mathsf{Sign}}$, the secret decryption key $sk_{\mathcal{U}}^{\mathsf{PKE}}$, and two secret payment keys $stk_{\mathcal{U}}^{\mathsf{AP}}$ and $spk_{\mathcal{U}}^{\mathsf{AP}}$. The public key $pk_{\mathcal{U}}$ is composed of the public signature verification key $pk_{\mathcal{U}}^{\mathsf{Sign}}$, the public encryption key $pk_{\mathcal{U}}^{\mathsf{PKE}}$ and the public payment data $pd_{\mathcal{U}}^{\mathsf{AP}}$. We stress that depending on if the used signature and encryption schemes are randomizable, the $\mathsf{RandKey}$ subroutine may act differently without loss of security. For the moment, let suppose that both signature and encryption are equipped with some Rand algorithm which given a secret key sk, a random public key pk and a random r, outputs the randomized secret key sk' and randomized public key pk'. In our proof-of-concept, we use the ElGamal [Elg85a] encryption scheme, which is naturally adapted for such randomized algorithm: For a prime p, let $sk \leftarrow s$ where $s \xleftarrow{\$} \mathbb{Z}_p$ be a secret key, g be the generator of a group $\mathbb{G}$ and $pk \leftarrow g^s$ with $g^s \in \mathbb{G}$ be the public key. Given a random $r \in \mathbb{Z}_p$, randomizing the secret and public key naturally follows as $sk' \leftarrow r \cdot s$ and $pk' \leftarrow g^{r \cdot s}$. Assuming such Rand algorithm for both encryption and signature schemes, the randomization algorithm $\mathsf{RandKey}$ samples two randoms r and r', before to compute the two randomization $(sk_{\mathcal{U}'}^{\mathsf{PKE}}, pk_{\mathcal{U}'}^{\mathsf{PKE}}) \leftarrow \mathsf{Rand}(sk_{\mathcal{U}}^{\mathsf{PKE}}, pk_{\mathcal{U}}^{\mathsf{PKE}}, r)$ and $(sk_{\mathcal{U}'}^{\mathsf{Sign}}, pk_{\mathcal{U}'}^{\mathsf{Sign}}) \leftarrow \mathsf{Rand}(sk_{\mathcal{U}}^{\mathsf{Sign}}, pk_{\mathcal{U}}^{\mathsf{Sign}}, r')$. The outputted randomized secret key $sk_{\mathcal{U}'}$ is composed of the tuple $(sk_{\mathcal{U}'}^{\mathsf{Sign}}, sk_{\mathcal{U}'}^{\mathsf{PKE}}, stk_{\mathcal{U}}^{\mathsf{AP}}, spk_{\mathcal{U}}^{\mathsf{AP}})$ consisting of randomized signature and encryption keys, along the payment keys. The outputted randomized public key is composed of the couple $(pk_{\mathcal{U}'}^{\mathsf{Sign}}, pk_{\mathcal{U}'}^{\mathsf{PKE}})$. We do the following observations: First, the public data $pd_{\mathcal{U}}^{\mathsf{AP}}$ is not outputted in the randomized public key since it is not mandatory to proceed to the payment. Second, the payment keys are not randomized since the payment already satifies all of our expectations to ensure a full anonymity. Third and last, if the used signature and encryption schemes do not support randomization, in this case the Rand algorithm generates a new key signature or encryption key pair, without loss of security. Indeed, without knowledge of the used random to randomise, a freshly generated key pair and its randomised version are statistically indistinguishable. This argument, intensively used during the anonymity security proofs, allows us to never rely on the key pair of a user and hence to obtain unlinkability and pseudonimity.

Formal Description of Applause. Assuming the KeyGen subroutine introduced above, we provide the different key generation algorithms for the parties involved in our protocol:

- $\underline{\mathcal{D}\mathsf{KeyGen}(1^\lambda)}$: Given the unary representation of the security parameter λ, outputs the key pair $(sk_{\mathcal{D}}, pk_{\mathcal{D}})$ computed as $\mathsf{KeyGen}(1^\lambda)$.
- $\underline{\mathcal{T}\mathsf{KeyGen}(1^\lambda)}$: Given the unary representation of the security parameter λ, outputs the key pair $(sk_{\mathcal{T}}, pk_{\mathcal{T}})$ computed as $\mathsf{KeyGen}(1^\lambda)$.
- $\underline{\mathcal{U}\mathsf{KeyGen}(1^\lambda)}$: Given the unary representation of the security parameter λ, outputs the key pair $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ computed as $\mathsf{KeyGen}(1^\lambda)$.
- $\underline{\mathcal{V}\mathsf{KeyGen}(1^\lambda)}$: Given the unary representation of the security parameter λ, it generates the signature key pair $(sk_{\mathcal{V}}^{\mathsf{Sign}}, pk_{\mathcal{V}}^{\mathsf{Sign}}) \leftarrow \mathsf{Sign.KeyGen}(1^\lambda)$, and an encryption key pair $(sk_{\mathcal{V}}^{\mathsf{PKE}}, pk_{\mathcal{V}}^{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. To authenticate a validator, we provide a certificate $cert_{\mathcal{V}}$ consisting of the signature of the public signature key $pk_{\mathcal{V}}^{\mathsf{Sign}}$

and the public encryption key $pk_{\mathcal{V}}^{\mathsf{PKE}}$ signed by $\mathcal{D}$. Finally, it returns the triplet $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, cert_{\mathcal{V}})$ where the secret key $sk_{\mathcal{V}}$ is $(sk_{\mathcal{V}}^{\mathsf{Sign}}, sk_{\mathcal{V}}^{\mathsf{PKE}})$ and the public key $pk_{\mathcal{V}}$ is $(pk_{\mathcal{V}}^{\mathsf{Sign}}, pk_{\mathcal{V}}^{\mathsf{PKE}})$.

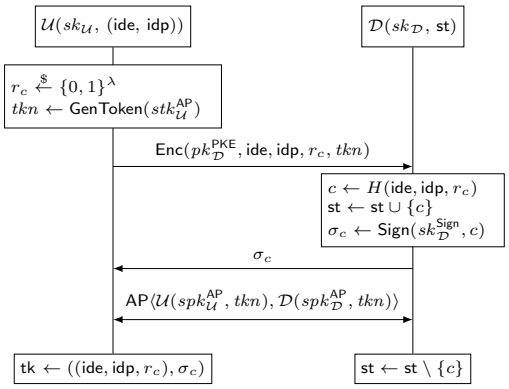


Figure 4.21: Diagram of Purchase protocol.

- $\underline{\mathsf{Purchase}\langle \mathcal{U}(sk_{\mathcal{U}}, (\mathsf{ide}, \mathsf{idp})), \mathcal{D}(sk_{\mathcal{D}}, \mathsf{st})\rangle}$: Before the execution of the Purchase protocol whose diagram of communications is depicted in Figure 4.21, the user $\mathcal{U}$ chooses the event of the ticket denoted by $\mathsf{ide} \leftarrow \mathsf{ID}_{\mathcal{E}}$, and a serial or seat number $\mathsf{idp} \leftarrow \mathsf{ID}_{\mathcal{P}}$. The protocol starts by the user $\mathcal{U}$ sampling a random r_c from space $\{0,1\}^{\lambda}$, and generating a new payment token tkn by executing the token generation algorithm $\mathsf{GenToken}(stk_{\mathcal{U}}^{\mathsf{AP}})$. Then it sends to the ticket distributor $\mathcal{D}$ a ciphertext denoted $\mathsf{Enc}(pk_{\mathcal{D}}^{\mathsf{PKE}}, \mathsf{ide}, \mathsf{idp}, r_c, tkn)$, encrypting the desired place specified by the identifier $(\mathsf{ide}, \mathsf{idp})$, the random r_c as well as the payment token, under the public encryption key $pk_{\mathcal{D}}^{\mathsf{PKE}}$. After the ciphertext decryption, the ticket distributor $\mathcal{D}$ checks that desired seat identified by $(\mathsf{ide}, \mathsf{idp})$ has not been purchased before. If the seat is still available, $\mathcal{D}$ computes $H(\mathsf{ide}, \mathsf{idp}, r_c)$ the hash c over the desired seat $(\mathsf{ide}, \mathsf{idp})$ along the random r_c. It includes the obtained hash c into the set of invalid tickets $\mathsf{st} \leftarrow \mathsf{st} \cup \{c\}$ preventing the ticket to be valid and used until the reception of the payment. The ticket distributor $\mathcal{D}$ computes the last element constituting a valid ticket, by signing the hash c as $\sigma_c \leftarrow \mathsf{Sign}(sk_{\mathcal{D}}^{\mathsf{Sign}}, c)$. This signature σ_c is forwarded to the user $\mathcal{U}$. At the reception of the signature σ_c, the user $\mathcal{U}$ verifies the validity of σ_c with the signature verification function $\mathsf{Sign.Verif}(pk_{\mathcal{D}}^{\mathsf{Sign}}, \sigma_c, c)$. In case where the signature is valid, guaranteeing the approval of the ticket distributor $\mathcal{D}$, the user $\mathcal{U}$ proceeds to the payment using the anonymous payment protocol with $\mathsf{AP}\langle \mathcal{U}(spk_{\mathcal{U}}^{\mathsf{AP}}, tkn), \mathcal{D}(spk_{\mathcal{D}}^{\mathsf{AP}}, tkn)\rangle \rightarrow \mathcal{U}(b), \mathcal{D}(b)$. As a result, both the user $\mathcal{U}$ and the ticket distributor $\mathcal{D}$ obtain a validation bit b, equaling 1 in case of successful payment. In such case, $\mathcal{U}$ returns the success bit b along with the ticket $\mathsf{tk} = ((\mathsf{ide}, \mathsf{idp}, r_c), \sigma_c)$. On the other side, still assuming a valid payment, the ticket distributor $\mathcal{D}$ removes the hash c from shared state $\mathsf{st} \leftarrow \mathsf{st} \setminus \{c\}$, allowing the created ticket tk to be used. As a response to the protocol, $\mathcal{D}$ returns the validation bit b as well as the updated shared state st.
- $\underline{\mathsf{Refund}\langle \mathcal{U}(sk_{\mathcal{U}}, \mathsf{tk}), \mathcal{D}(sk_{\mathcal{D}}, \mathsf{st})\rangle}$: The Refund protocol, depicted in Figure 4.22 starts with the user $\mathcal{U}$ generating a payment token tkn using the token generation algorithm $\mathsf{GenToken}(stk_{\mathcal{U}}^{\mathsf{AP}})$. Then the user encrypts its ticket $\mathsf{tk} = ((\mathsf{ide}, \mathsf{idp}, r_c), \sigma_c)$

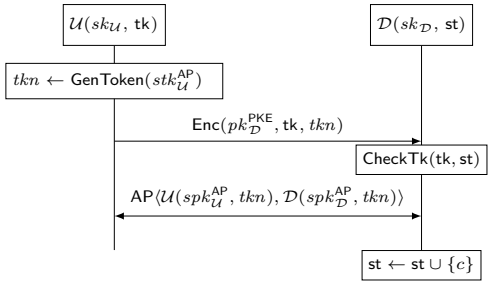


Figure 4.22: Diagram of Refund protocol.

along the payment token tkn under the public encryption key of the ticket distributor $\mathcal{D}$, resulting into the ciphertext $\mathsf{PKE.Enc}(pk_{\mathcal{D}}^{\mathsf{PKE}}, \mathsf{tk}, tkn)$. After the decryption, the ticket distributor $\mathcal{D}$ checks the validity of the received ticket by running the ticket verification algorithm $\mathsf{CheckTk}(\mathsf{tk}, \mathsf{st})$. If the ticket is valid, the ticket distributor $\mathcal{D}$ proceeds to the refund of the ticket by the mean of a payment via the anonymous payment protocol, resulting into the execution of the payment protocol $\mathsf{AP}\langle\mathcal{U}(spk_{\mathcal{U}}^{\mathsf{AP}}, tkn), \mathcal{D}(spk_{\mathcal{U}}^{\mathsf{AP}}, tkn)\rangle \rightarrow \mathcal{U}(b), \mathcal{D}(b)$. If the payment succeeds with b equals $\top$, then the ticket has been refund and should not be considered valid anymore: The hash of the ticket c (computed over the event and seat identifiers $\mathsf{ide}, \mathsf{idp}$ and the random r_c) is inserted in the shared state st. Observe that once inserted, the CheckTk procedure will always reject this ticket since it belongs to the set of invalid tickets. In case where the payment does not succeed, when b equals 0, the ticket distributor $\mathcal{D}$ does not insert the ticket inside the set of invalid tickets and aborts the refund. As a result of the refund procedure, both parties return b, and $\mathcal{D}$ additionally returns the shared state st along the ticket tk.

We now focus on the transfer of ticket between two users denoted $\mathcal{U}_1$ and $\mathcal{U}_2$, assisted by the ticket transfer authority $\mathcal{T}$, running the Transfer protocol. This tranfer protocol starts with a preliminary step in which users $\mathcal{U}_1$ and $\mathcal{U}_2$ are locally exchanging each other their randomised public key. At the end of this preliminary step, user $\mathcal{U}_1$ is owning the randomized public key $pk'_{\mathcal{U}_2}$ of user $\mathcal{U}_2$, and conversely. This step is required to guarantee that $\mathcal{U}_2$ will be the new owner of the ticket. These randomized public keys can be exchanged easily for instance via Bluetooth [NSI+15] during a physical meeting.

- $\underline{\mathsf{Transfer}\langle\mathcal{U}_1(sk_{\mathcal{U}_1}, \mathsf{tk}_1), \mathcal{T}(sk_{\mathcal{T}}, \mathsf{st}), \mathcal{U}_2(sk_{\mathcal{U}_2}, (\mathsf{ide}, \mathsf{idp}))\rangle}$: The ticket transfer protocol Transfer, depicted in Figure 4.23, starts with the user $\mathcal{U}_1$ authenticating the randomized public key of the user $\mathcal{U}_2$ and the random r_c with $\mathsf{Sign}(sk_{\mathcal{U}'_1}^{\mathsf{Sign}}, pk_{\mathcal{U}'_2}, r_c)$ later denoted $\sigma_{T,1}$. We do the following two observations: First, the secret signature key used by $\mathcal{U}_1$ to authenticate the randomized public key, is also randomized. Second, the signature not only authenticates the randomized public key of $\mathcal{U}_2$ but also the random r_c, being part of the ticket of $\mathcal{U}_1$. Whereas the first observation introduces a risk of usurpation by not authenticating the signing key of user, the second observation mitigates this potential authentication issue with the random r_c included in the signature $\sigma_{T,1}$, known only by the original owner of the ticket. In addition, the user $\mathcal{U}_1$ generates a payment token tkn_1 used at the end of the transfer protocol to obtain the refund of the ticket. Then, $\mathcal{U}_1$ initiates the first communication with the transfer authority $\mathcal{T}$ with the triplet $(pk_{\mathcal{U}'_1}, \sigma_{T,1}, \mathsf{Enc}(pk_{\mathcal{T}}^{\mathsf{PKE}}, c, \mathsf{tk}, tkn_1))$.

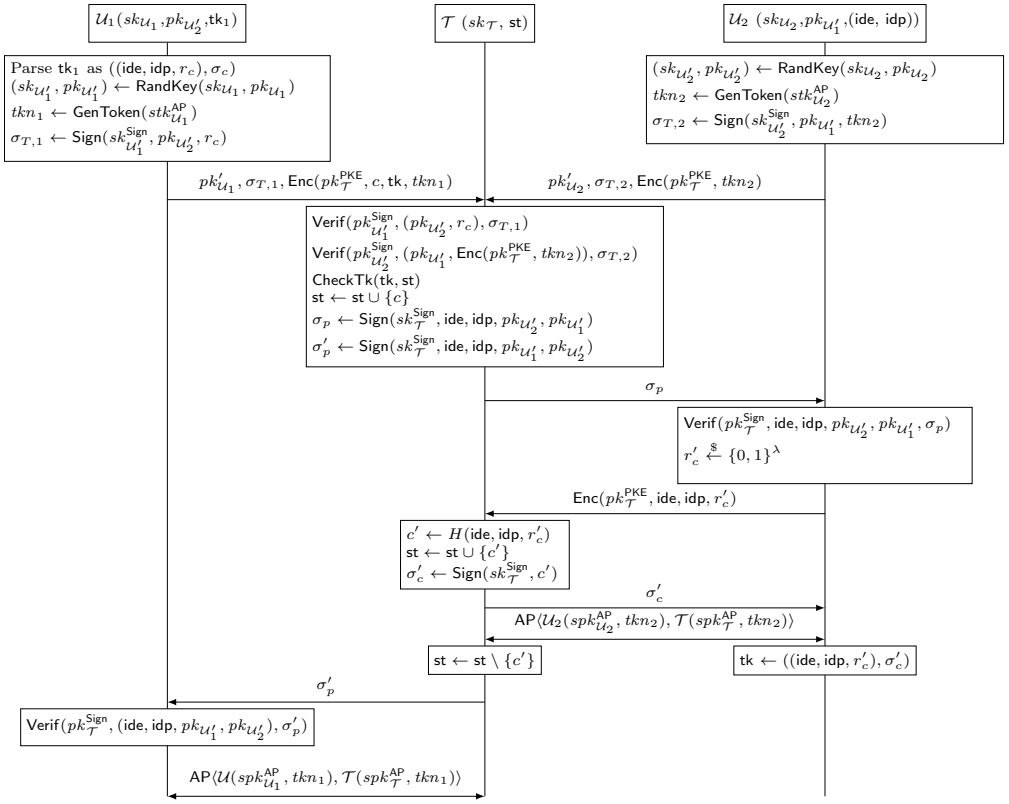


Figure 4.23: Diagram of sequences for the Transfer protocol.

At the same time, the user $\mathcal{U}_2$ generates a payment token tkn_2 used to purchase the ticket. Then, the user $\mathcal{U}_2$ authenticates the randomized public key of $\mathcal{U}_1$ as well as the payment token using the signing algorithm $\mathsf{Sign}(sk_{\mathcal{U}_2'}^{\mathsf{Sign}}, pk_{\mathcal{U}_1'}, tkn_2)$, producing the signature $\sigma_{T,2}$. The first communication with the transfer authority $\mathcal{T}$ from $\mathcal{U}_2$ is composed of the triplet $(pk_{\mathcal{U}_2'}, \sigma_{T,2}, \mathsf{Enc}(pk_{\mathcal{T}}^{\mathsf{PKE}}, tkn_2))$, composed of the randomized public key of $\mathcal{U}_2$, the signature $\sigma_{T,2}$ as well as the encryption of the payment token under the (authenticated) public encryption key of the transfer authority $\mathcal{T}$.

At this point, the transfer authority $\mathcal{T}$ has received the randomized public keys of both users $\mathcal{U}_1$ and $\mathcal{U}_2$, the signatures $\sigma_{T,1}$ and $\sigma_{T,2}$ but also the transferred ticket of $\mathcal{U}_1$ and the two payment tokens obtained after decryption of the two received ciphertexts. First, $\mathcal{T}$ verifies the validity of $\sigma_{T,1}$ and $\sigma_{T,2}$ by executing the signature verification algorithm Sign.Verif inputted with the previsouly obtained parameters. If the verification succeeds, both users are authenticating each other. Otherwise, the verification has failed and the transfer is aborted. Then, $\mathcal{T}$ checks the validity of the received ticket by executing the ticket verification algorithm $\mathsf{CheckTk}(\mathsf{tk}, \mathsf{st})$. Again, if the verification fails, the transfer is aborted. Otherwise, the transferred ticket (the ticket containing the random r_c) is inserted in the shared state of invalid ticket st, preventing the ticket to be used during the transfer. At any moment before the payment of $\mathcal{U}_2$, if any verification fails then the transferred ticket is removed from set of invalid tickets st and the transfer is aborted. $\mathcal{T}$

computes two signatures σ_p and σ'_p, expressing the agreement of $\mathcal{T}$ to proceed to the transfer. The first signature σ_p, signed using the signing key $sk_{\mathcal{T}}^{\mathsf{Sign}}$ of $\mathcal{T}$, authenticates the event and seat identifier ide and idp, the randomized public key $pk_{\mathcal{U}'_2}$ of $\mathcal{U}_2$ and the randomized public key $pk_{\mathcal{U}'_1}$ of $\mathcal{U}_1$, in this order. This signature σ_p is sent to the user $\mathcal{U}_2$ which verifies the authenticity of the signature. An invalid signature verification does not lead to an abort, but is rejected by $\mathcal{U}_2$ without any response. Then, the user $\mathcal{U}_2$ proceeds to the part of the $\mathsf{Transfer}$ protocol, close to our $\mathsf{Purchase}$ protocol, consisting of generating a new ticket for the same event and seat identifiers: A new random r'_c is sampled from $\{0,1\}^\lambda$ and sent to $\mathcal{T}$, carefully encrypted using the public encryption key $pk_{\mathcal{T}}^{\mathsf{PKE}}$ of $\mathcal{T}$, along the event and seat identifiers. Once received, the ticket transfer computes the hash c' as the evaluation of the hash function $H(\mathsf{ide}, \mathsf{idp}, r'_c)$, inserts c' in the set of invalid tickets st to prevent its usage before the payment, and responds to $\mathcal{U}_2$ with the signature of $\sigma_{c'}$ of c' using the signing key $sk_{\mathcal{T}}^{\mathsf{Sign}}$. Then, except if the $\sigma_{c'}$ is rejected by the signature verification algorithm, the user $\mathcal{U}_2$ proceeds to the payment using the payment token tkn_2. Again, if the payment does not succeed, the tranfer is aborted and the set of invalid tickets reverted to its initial state, allowing the transferred ticket (containing the random r_c) to be used. Otherwise, the payment is successful. Therefore, the hash c' is removed from the set of invalid tickets.

At this point, the ticket tk_1 owned by the user $\mathcal{U}_1$ has been transferred and hence should not be valid anymore. Observe that it is already ensured by the set of invalid tickets containing the hash c, computed from the ticket tk_1. Any attempt to use the ticket tk_1 would lead to a direct failure. The last step consists of performing the refund to the user $\mathcal{U}_1$. The signature σ'_p computed by the transfer authority $\mathcal{T}$ is sent to user $\mathcal{U}_1$, notifying the success of the ticket transfer. After a necessary verification of the authenticity of $\mathcal{T}$ using the signature verification algorithm, $\mathcal{U}_1$ proceeds to the refund with $\mathcal{T}$ using the payment token tkn_1 exchanged at the beginning of the transfer protocol execution.

The validation protocol requires for its last interaction a physical channel, to prevent relay attacks where an active external adversary blocks the ticket in the network, and play it in the validation process.

- $\underline{\mathsf{Validate}\langle\mathcal{U}(sk_{\mathcal{U}}, \mathsf{tk}), \mathcal{V}(sk_{\mathcal{V}}, \mathsf{st})\rangle}$: The validation protocol, depicted in Figure 4.24 starts with the validator $\mathcal{V}$, providing its public key $pk_{\mathcal{V}}$ along its certificate $cert_{\mathcal{V}}$ to the user $\mathcal{U}$ attempting to attend the event. Recall that the public key $pk_{\mathcal{V}}$ is the triplet containing the public encryption key $pk_{\mathcal{V}}^{\mathsf{PKE}}$ and the public signature key $pk_{\mathcal{V}}^{\mathsf{Sign}}$. The authenticity of the validator public key $pk_{\mathcal{V}}$ is verified by the user $\mathcal{U}$ using the signature verification algorithm $\mathsf{Verif}(pk_{\mathcal{D}}^{\mathsf{Sign}}, (pk_{\mathcal{V}}^{\mathsf{Sign}}, pk_{\mathcal{V}}^{\mathsf{PKE}}), cert_{\mathcal{V}})$. If the signature is rejected, the user $\mathcal{U}$ aborts the verification protocol. Otherwise, $\mathcal{U}$ executes the key randomization algorithm $\mathsf{RandKey}(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ to obtain a new key pair $(sk_{\mathcal{U}'}, pk_{\mathcal{U}'})$, used to compute a ciphertext $\mathsf{Enc}(pk_{\mathcal{V}}^{\mathsf{PKE}}, pk_{\mathcal{U}'}, \mathsf{tk})$ encrypting the randomized public key $pk_{\mathcal{U}'}$ along the ticket tk, sent to the validator $\mathcal{V}$. After decryption, the validator executes the ticket verification $\mathsf{CheckTk}(\mathsf{tk}, \mathsf{st})$. If the ticket is valid, the validator $\mathcal{V}$ inserts the hash c computed from the received ticket tk in the state of invalid ticket st, and creates a *challenge* s sampled from $\{0,1\}^\lambda$, which is carefully encrypted under the provided randomized public encryption

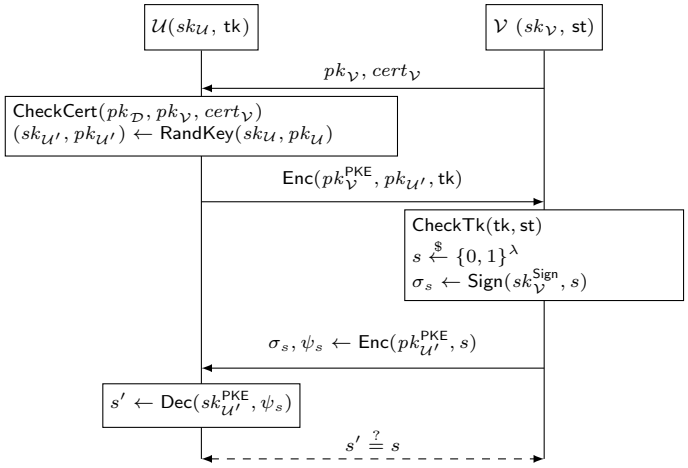


Figure 4.24: Diagram of Validate protocol. The dashed line is the physical channel.

key $pk_{\mathcal{U}'}^{\mathsf{PKE}}$. The resulting ciphertext, denoted ψ_s, is signed by the signature key $sk_{\mathcal{V}}^{\mathsf{Sign}}$ of the validator $\mathcal{V}$, producing the signature σ_s. The ciphertext ψ_s and the signature σ_s are sent as a response to $\mathcal{U}$. After a necessary verification of the authenticity of the signature σ_s, and the decrypted challenge s', values s and s' are compared through a physical channel. If the verification fails, then the validator $\mathcal{V}$ removes the hash c from the set of invalid tickets st and halts. Otherwise, both parties commonly terminate with a success bit b at $\top$. Recall that the validation protocol relies on a physical channel, as explained in Section 4.1. This necessary hypothesis allows us to prevent an arbitrary to block the ultimate interaction from $\mathcal{U}$ and replay the obtained response to the challenge to attend the event, instead of the legimate user. We stress that this physical channel, preserving the anonymity of the user, is already used in practice using QRCode scanner. Eventually, other approachs such that Bluetooth [NSI+15] or NFC [CHY23] are plausible candidates to achieve our expectations.

4.3.2 Security Proofs of Applause

We present the security proofs for Applause, with respect to the security model presented in Section 4.2. Through this section, we operate proofs divided into sequences of games $\mathsf{G}^0, \ldots, \mathsf{G}^n$ for some $n \in \mathbb{N}$. We refer to the i-th game as $\mathsf{G}^i_{\mathcal{A}}(1^{\lambda})$ for a given PPT algorithm $\mathcal{A}$ and a security parameter 1^{λ}. Recall that the security proofs of Applause presented in Section 4.3, rely on the *Random Oracle Model* (ROM).

Theorem 6. Assuming an EUF-CMA-secure signature scheme, then for every polynomial time adversary $\mathcal{A}$, Applause provides *unforgeability* with the following bound:

$$\Pr[\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}(1^{\lambda}) \to 1] \leq 2 \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}$$

Proof of Unforgeability. Let $\mathcal{A}$ be a PPT adversary and assume by contradiction that the following probability $\Pr[\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}(1^{\lambda}) \to 1]$ is non-negligible. This proof relies on the ROM, meaning that $\mathcal{A}$ has access to an oracle Ohash replacing the function H and

programmed by the challenger $\mathcal{C}$. The random oracle maintains an internal storage consisting of a list of message-hash pairs of the form (m, h). On a call for a message m, the oracle checks if there exists a hash h such that the pair (m, h) is stored, and returns h in such case. Otherwise, it samples a random a value h representing the hash associated to m, it stores the pair (m, h) and returns the sampled value h to the caller.

Game G^0. The initial game corresponds to the $\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}$ experiment. That is, we observe a perfect indistinguishability (denoted $\triangleq$) between our initial game and the $\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}$ experiment:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}(1^\lambda) \to 1\right]$$

Game G^1. In this game, we now reject tickets with valid signatures from $\mathcal{D}$ that are not produced by the challenger. To distinguish such signature from the originally produced ones, $\mathcal{C}$ keeps updated a set $\mathcal{S}_{\mathcal{D}} = \{m_i, \sigma_i\}$ containing all the signatures produced by $\mathcal{D}$ during the protocol. A signature is deemed valid if the message-signature pair is contained in this set, otherwise invalid. This prevents $\mathcal{A}$ from forging a signature, which was previously possible with probability $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}$. Therefore by the difference lemma [PS00], we obtain the following upper bound:

$$\left|\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}$$

Game G^2. The game works as the previous game, except that we now reject valid signatures of $\mathcal{T}$ that are not produced by the challenger, by following the same approach of the previous game. As a result, no forged signature can be injected by the adversary $\mathcal{A}$. Again, by the difference lemma [PS00], we obtain the following upper bound:

$$\left|\Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^2_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}$$

In G^2, every received ticket tk, containing a signature σ_c either computed by the ticket distributor $\mathcal{D}$ or transfer authority $\mathcal{T}$, has been produced by the challenger, otherwise rejected. Observe that, by construction of the protocol, this argument is not sufficient to ensure unforgeability of the ticket, the adversary being able to obtain any signature of the hash c computed over $(\mathsf{ide}, \mathsf{idp}, r_c)$ for a random r_c of its choice. Indeed, during the $\mathsf{Purchase}$ protocol execution, the ticket distributor $\mathcal{D}$ distributes the signature with the user, and waits until the success of the payment, in which the hash c is removed from the state of invalid tickets. The adversary $\mathcal{A}$ does not proceed to the payment, it obtains the signature σ_c of c, allowing adversary $\mathcal{A}$ to constitute the apparently valid ticket $\mathsf{tk} = ((\mathsf{idp}, \mathsf{idp}, r_c), \sigma_c$. The same scenario occurs with the $\mathsf{Transfer}$ protocol as well.

However, it does not mean that the adversary $\mathcal{A}$ has successfully obtained a *valid* ticket from the point of the view of the system: Whereas all parts of the produced ticket tk is valid, the crucial and missing last step, performed by the system, is the supression of the hash c from the shared state st. Since the payment does not occur, the ticket is not removed from the set of invalid tickets and hence is not directly usable. As a result, the evaluation of $\mathsf{CheckTk}$ over the ticket tk always leads to an invalid validation result, preventing the ticket to be accepted, aborting the running protocol.

As a direct consequence of the above remarks, the probability for the adversary $\mathcal{A}$ to produce a valid ticket not crafted by the challenger is $\Pr\left[\mathsf{G}^2_{\mathcal{A}}(1^\lambda) \to 1\right] = 0$. As a result, we obtain the following upper bound:

$$\Pr\left[\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}(1^\lambda) \to 1\right] \leq 2 \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}$$

By hypothesis on the EUF-CMA property of the signature Sign, we know that $\mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A}}$ is negligible, leading to a negligible quantity on the right hand side of the inequation, giving a direct contraction with the initial hypothesis. Hence, $\Pr[\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A}}(1^\lambda) \to 1]$ is negligible and Applause has Unforgeability.

□

Theorem 7. Instantiated with an IND-CPA encryption scheme, for any PPT $\mathcal{A}$, Applause provides *privacy* under the random oracle model with the following upper bound:

$$\Pr\left[\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}(1^\lambda) \to 1\right] \leq 2^{-\lambda} + 8 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A}}$$

Proof of Privacy. We prove secrecy of r_c through this proof. Assume $\mathcal{A}$ has a PPT algorithm. If an adversary $\mathcal{A}$ is unable to recover r_c, then it is unable to win against $\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}(1^\lambda)$, or at least with a negligible probability. Notice that $\mathcal{U}$, the user simulated by $\mathcal{C}$ which has generated tk, cannot be corrupted, otherwise $\mathcal{A}$ would fail to the experiment under $\mathsf{corr}_1 = 0$. The following modification applied to the oracles are assumed to append only when the ticket tk is involved during the execution of the oracle.
Game G^0. The initial game corresponds to the experiment $\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}$, hence:

$$\Pr\left[\mathsf{G}^0(1^\lambda) \to 1\right] \triangleq \Pr[\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}(1^\lambda) \to 1]$$

Game G^1. First, we modify the $\mathsf{Purchase}$ algorithm happening in the experiment. When tk is purchased, the challenger sends a random element as the first message from $\mathcal{U}_1$, thus replacing $\mathsf{Enc}_{pk^{\mathsf{PKE}}_{\mathcal{D}}}(\mathsf{ide}, \mathsf{idp}, r_c, c)$ by a random sampled uniformly from $[\mathsf{Enc}_{pk^{\mathsf{PKE}}_{\mathcal{D}}}]$. $\mathcal{D}$ being also simulated by $\mathcal{C}$ it still has access to $\mathsf{ide}, \mathsf{idp}, r_c, c$ and then is able to continue the $\mathsf{Purchase}$ protocol.

We observe that the difference between G^0 and G^1 occurs only in the encryption of this message. Under the IND-CPA hypothesis, a distinguisher would have negligible chances to distinguish between these two experiments. Hence we obtain the

$$\left|\Pr\left[\mathsf{G}^0(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^1(1^\lambda) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{PKE}}$$

Game G^2. The second game replaces the encryption of tk under the key $pk^{\mathsf{E}}_{\mathcal{D}}$ during a call to $\mathsf{ORefund}$ by a random element sampled uniformly from $[\mathsf{Enc}_{pk^{\mathsf{PKE}}_{\mathcal{D}}}]$. This is the same argument as before, we directly conclude to

$$\left|\Pr\left[\mathsf{G}^1(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^2(1^\lambda) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{PKE}}$$

Game G^3. The third game replaces the encryption of tk under the key $pk^{\mathsf{E}}_{\mathcal{T}}$ during a call to $\mathsf{OTransfer}$ by a random element sampled uniformly from $[\mathsf{Enc}_{pk^{\mathsf{PKE}}_{\mathcal{D}}}]$. As seen in

G^1 and in G^2, we have

$$\left|\Pr\left[\mathsf{G}^2(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^3(1^\lambda) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{PKE}}$$

Game G^4. In the last game, concluding this proof, we replace the encryption of $pk'_{\mathcal{U}}, \mathsf{tk}, cert'_{\mathcal{U}}$ under $pk^{\mathsf{E}}_{\mathcal{V}}$ by a random element sampled uniformly from $[\mathsf{Enc}_{pk^{\mathsf{PKE}}_{\mathcal{D}}}]$. Still using the indistinguishability argument we have

$$\left|\Pr\left[\mathsf{G}^3(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^4(1^\lambda) \to 1\right]\right| \leq 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{PKE}}$$

Since $\mathcal{A}$ has no more advantage, and since r_c is randomly picked at uniformly in $\{0,1\}^\lambda$, $\mathcal{A}$ has probability $2^{-\lambda}$ to guess correctly, hence $\Pr\left[\mathsf{G}^4(1^\lambda) \to 1\right] = 2^{-\lambda}$. Finally, we observe that:

$$\mathsf{Adv}^{\mathsf{PRIV}}_{\mathcal{A}}(1^\lambda) = \Pr[\mathsf{Exp}^{\mathsf{PRIV}}_{\mathcal{A}}(1^\lambda) \to 1] \leq 8 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathsf{PKE}} + 2^{-\lambda}$$

□

Theorem 8. APPLAUSE provides *no-double-spending* unconditionally.

Proof of No-Double-Spending. Let $\mathcal{A}$ be a PPT adversary. We show that the probability $\Pr[\mathsf{Exp}^{\mathsf{DS}}_{\mathcal{A}}(1^\lambda) \to 1]$ is negligible. In the game $\mathsf{Exp}^{\mathsf{DS}}_{\mathcal{A}}$, $\mathcal{A}$ is free to take advantage of any sequence of execution it has chosen. We show that under any chosen sequence, it is impossible for $\mathcal{A}$ to success the same algorithm twice with the same ticket tk. The proof relies on the consistency of the state st, updated by the challenger $\mathcal{C}$ during the protocol. We argue that this state prevents any double-spending, indeed acting as a blacklist containing every ticket tk used in any of the protocols. We proceed by analyzing the update of the state through the algorithms: In the Refund protocol, on the reception of tk, $\mathcal{D}$ executes the $\mathsf{CheckTk}$ algorithm which checks that σ_c is signed either by $\mathcal{D}$ or by $\mathcal{T}$, but also that c is not contained in st. If not, c is inserted in st. Suppose now that $\mathcal{A}$ replays the refund protocol again, then the signature still verifies, but since tk is now contained in st, then the refund fails. The ticket transfer holds on the same principle. When transferring tk, algorithm $\mathsf{CheckTk}$ is executed, ensuring that tk is not already contained in st. Exactly as the previous protocols, on the reception of ticket tk, the validation executes the $\mathsf{CheckTk}$ checking that tk is not contained in st. When the algorithm successfully ends, then tk is required to be contained in st to notify that the ticket is now considered as invalid.

All algorithms taking a ticket tk as an input ends with a state st containing tk. Since the state st is fully controlled by $\mathcal{C}$ and cannot be modified by $\mathcal{A}$, we can directly conclude that for every PPT adversary, $\Pr[\mathsf{Exp}^{\mathsf{DS}}_{\mathcal{A}}(1^\lambda) \to 1]$ is zero, meaning that the double-spending, in any sequence, is prevented. □

Theorem 9. Instantiated with statistically indistinguishable randomisable keys, and an anonymous payment scheme, for every security parameter $\lambda \in \mathbb{N}$ and every adversary $\mathcal{A}$, APPLAUSE provides *pseudonymity* with the upper bound: $\left|\Pr\left[\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \leq 8 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}$.

Proof of Pseudonymity. We start by considering an initial game G^0 consisting of the $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$ experiment, in which the adversary has access to the public keys of both users and interact only with the user $\mathcal{U}_b$ for a random bit b chosen by the challenger. To prove the pseudonymity, we replace all the keys associated to both users $\mathcal{U}_0$ and $\mathcal{U}_1$ by a new key pair which are not associated with any user. Therefore, the adversary does not have any advantage to identify the chosen user and hence no chance to break the pseudonymity.

Game G^0. The initial game corresponds to experiment $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$, hence we have a perfect indistinguishability between this game and the $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$ experiment:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] \triangleq \Pr\left[\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda) \to 1\right]$$

Game G^1. In our initial game, the adversary (adversary) can interact with either user $\mathcal{U}_0$ or $\mathcal{U}_1$. In this modified game, we introduce the generation of new anonymous payment keys $stk^{\mathsf{AP}}_{\mathsf{new}}$ and $spk^{\mathsf{AP}}_{\mathsf{new}}$. Instead of using the payment keys of $\mathcal{U}_0$ for all payments, the challenger now uses freshly generated payment keys. The indistinguishability between this game, denoted as G^1, and our initial game, denoted as G^0, holds assuming our anonymous payment ensures pseudonymity, where the adversary has access to a payment oracle for only one of the two users.

For contradiction, suppose there exists an adversary $\mathcal{A}$ for which the advantage in distinguishing between G^0 and G^1 is non-negligible. Specifically, let p_0 be the probability of success of $\mathcal{A}$ in G^0 and p_1 be the probability of failure of $\mathcal{A}$ in G^1. Denote by ϵ the distinguishing probability $p_0 - p_1$.

We construct an adversary $\mathcal{B}$ which utilizes $\mathcal{A}$ to break the pseudonymity property of the anonymous payment scheme. Let $\mathcal{C}$ represent the challenger running the experiment $\mathsf{Exp}^{\mathsf{APPse}}_{\mathcal{B}}$. At the beginning, $\mathcal{C}$ generates two payment keys: One for $\mathcal{U}_0$ and another independent of $\mathcal{U}_0$. On the other hand, $\mathcal{B}$ executing $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$ holds the payment keys for $\mathcal{U}_1$. At the start of the experiment, $\mathcal{B}$ samples a bit b' and interacts with $\mathcal{A}$ using the payment key of $\mathcal{U}_1$ if b' equals one, or it uses the payment keys chosen by $\mathcal{C}$ based on a randomly chosen bit b. If b is zero, $\mathcal{C}$ provides a payment oracle for the payment keys of $\mathcal{U}_0$, and $\mathcal{B}$ runs G^0. Otherwise, when b is one, the provided payment oracle relies on the freshly generated payment keys, and $\mathcal{B}$ runs G^1.

During the payment oracle execution, when b' is zero, $\mathcal{B}$ acts as a proxy between $\mathcal{C}$ and $\mathcal{A}$ to enable $\mathcal{A}$ to proceed with a payment using the payment keys held by $\mathcal{C}$. However, when b' is one, the payment oracle is not called by $\mathcal{B}$ since $\mathcal{B}$ owns the payment keys of $\mathcal{U}_1$. At the end of $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$, $\mathcal{A}$ (assumed to distinguish between G^0 and G^1) outputs a prediction bit b^* to $\mathcal{B}$, which is forwarded to the challenger $\mathcal{C}$.

$$\begin{aligned}
\Pr[b = b^*] &= \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 1] \\
&+ \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 1] \\
&= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\
&+ \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 1] - \Pr[b^* = 0 | b = 1 \wedge b' = 1]) \\
&= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\
&= \frac{1}{2} + \frac{1}{4}(p_0 - p_1)
\end{aligned}$$

And since the success probability $\Pr[b = b^*]$ equals $\Pr\left[\mathsf{Exp}^{\mathsf{APPse}}_{\mathcal{B}}(1^\lambda) \to 1\right]$, it follows that:

$$\mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}} = \left|\Pr\left[\mathsf{Exp}^{\mathsf{APPse}}_{\mathcal{B}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| = \left|\frac{1}{2} + \frac{1}{4}\epsilon - \frac{1}{2}\right| = \frac{1}{4}\epsilon$$

And since we have assumed that the following advantage $\mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}$ is negligible for any adversary, then it follows that ϵ is negligible which enter in contradiction with our hypothesis. Therefore, under the assumption that $\mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}$ is negligible for all polynomial time adversary $\mathcal{A}$ then G^0 and G^1 are computationally indistinguishable:

$$\left|\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}(1^\lambda)$$

Game G^2. In this game, we proceed exactly as done previously, except that we now replace the payment keys of $\mathcal{U}_1$. Note that in this game, the payment keys of $\mathcal{U}_0$ is no more used to proceed for a payment. Since the reduction works exactly as before, we directly conclude that the upper-bound to distinguish between this game G^2 and game G^1 is defined as follows:

$$\left|\Pr\left[\mathsf{G}^1_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^2_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}(1^\lambda)$$

Game G^3. Instead of randomizing the key pair in $\mathsf{Validate}$ when $\mathcal{A}$ calls $\mathsf{OValidate}_b$, we generate a new signature and encryption key pairs. The resulting keys are unrelated to the key pair of the user. Since randomized keys are statistical indistinguishability from freshly generated keys, we obtain a statistical indistinguishability (denoted $\overset{\mathrm{s}}{\approx}$) between the games G^2 and G^3:

$$\Pr\left[\mathsf{G}^3_{\mathcal{A}}(1^\lambda) \to 1\right] \overset{\mathrm{s}}{\approx} \Pr\left[\mathsf{G}^2_{\mathcal{A}}(1^\lambda) \to 1\right]$$

Game G^4. On each call of $\mathcal{A}$ to $\mathsf{OTransfer}_b$, the same modification is put in place in $\mathsf{Transfer}$. As for the previous modification, games are therefore statistically indistin-

guishable. This leads to the statistical indistinguishability between G^4 and G^3:

$$\Pr\left[\mathsf{G}^4_{\mathcal{A}}(1^\lambda) \to 1\right] \overset{s}{\approx} \Pr\left[\mathsf{G}^3_{\mathcal{A}}(1^\lambda) \to 1\right]$$

In this game G^4, none of $(sk_{\mathcal{U}_0},\ pk_{\mathcal{U}_0})$ or $(sk_{\mathcal{U}_1},\ pk_{\mathcal{U}_1})$ are used by the challenger during the experiment. Hence, the adversary has no mean to recover b and then no advantage in breaking the pseudonymity. Hence, we obtain $\Pr\left[\mathsf{G}^4_{\mathcal{A}}(1^\lambda) \to 1\right] = \frac{1}{2}$. We now derive the upper bound as follows:

$$\begin{aligned}\left|\Pr\left[\mathsf{G}^0(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^4(1^\lambda) \to 1\right]\right| &= \left|\Pr\left[\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \\ &\leq 8 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}\end{aligned}$$

By hypothesis on the anonymity property of considered anonymous payment, the advantage $\mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}$ is negligible. Therefore, assuming that our anonymous payment scheme ensures pseudonymity, Applause provides pseudonymity.

□

Theorem 10. Instantiated with an anonymous payment having the property of randomisable keys, then for any PPT $\mathcal{A}$, Applause provides *unlinkability* with the following upper bound:

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \leq 16 \cdot \mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}}$$

Recall that in our construction, a ticket tk is defined by the place $(\mathsf{ide}, \mathsf{idp})$, the random r_c and the signature σ_c. From an information theory perspective, tk is completely unrelated from any user $\mathcal{U}$. Then, our proof simply consists to show that at any moment, the identity of $\mathcal{U}$ (*i.e.,* the public key $pk_{\mathcal{U}}$) is used during the communication.

Proof of Unlinkability. Our proof is designed as a sequence of games where the initial game G^0 corresponds the game $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$, and our last game corresponds to the game $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ where $\mathcal{C}$ does not rely on the users keys. We show that both games are indistinguishable through a sequence of negligible modifications. During this sequence, we focus our attention on the part of the experiment in which the adversary $\mathcal{A}$ interacts with the user $\mathcal{U}_b$. In this challenge phase, a single ticket is purchased by $\mathcal{U}_b$ and later transfered, refunded or validated, based on the choice of $\mathcal{A}$. Let $\mathcal{A}$ be a PPT adversary, we show that it has probability to succeed at $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ close to $1/2$.

Game G^0. The initial game corresponds to the experiment $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ Hence, for every adversary $\mathcal{A}$, we have:

$$\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] \overset{\mathrm{p}}{=} \Pr\left[\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}(1^\lambda) \to 1\right]$$

Game G^1. In this game, we focus on the anonymous payment performed during the execution of the ticket purchase via the $\mathsf{Purchase}$ protocol in the challenge phase. At the beginning of the experiment $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ of the ticketing system, two user key pairs are generated, one for the user $\mathcal{U}_0$ and a second for the user $\mathcal{U}_1$. The adversary $\mathcal{A}$ is allowed, via specific oracles $\mathsf{OPurchase}_l, \mathsf{ORefund}_l, \mathsf{OTransfer}_l, \mathsf{OValidate}_l$ oracles for $l \in \{0, 1\}$, to

interact with both users. Then, at some point, the experiment selects a random bit b and purchases a ticket for a seat $(\mathsf{ide}, \mathsf{idp})$ designated by the adversary.

In this game, we introduce the two payment keys generation, one for each user $\mathcal{U}_0$ and $\mathcal{U}_1$. For clarity, let denote these keys by (stk'_0, spk'_0) and (stk'_1, spk'_1). Instead of using the payment keys of user $\mathcal{U}_0$, in this game we rely on the freshly generated payment keys (stk'_0, spk'_0). That is, the payment keys would not be used in the $\mathsf{Purchase}$ protocol anymore. Note that in the $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ experiment, a bit b is randomly chosen and lead to the usage of payment keys of the user $\mathcal{U}_b$. When b equals zero, then the used payment key is changed whereas when b equals one, the view of the adversary remains unchanged.

Let suppose an adversary against the $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ experiment. We construct an adversary $\mathcal{B}$ which relies on $\mathcal{A}$ to break the unlinkability of the anonymous payment scheme, whose experiment $\mathsf{Exp}^{\mathsf{APUnl}}_{\mathcal{B}}$ is simulated by the challenger $\mathcal{C}$. In the $\mathsf{Exp}^{\mathsf{APUnl}}_{\mathcal{B}}$ experiment of the anonymous payment scheme, simulated by the challenger $\mathcal{C}$, two payment keys are generated. During the challenge phase, one of these two payment keys is used, the decision to use the first or the second key is based on a randomly chosen bit b, chosen by $\mathcal{C}$. These two keys are associated by the adversary $\mathcal{B}$ to the payment keys of $\mathcal{U}_b$ and to the freshly generated payment keys (being independent of $\mathcal{U}_0$ and $\mathcal{U}_1$). In addition, to this decision bit b, the adversary $\mathcal{B}$ chooses a random bit b'. When, b' equals zero, then $\mathcal{B}$ relies either on the payment keys of $\mathcal{U}_0$ or on the freshly generated payment keys, both payment keys being owned by $\mathcal{C}$. In contrast, when b' equals one, then it relies on the payment keys of user $\mathcal{U}_1$, generated and owned by $\mathcal{B}$. To be a valid reduction, the adversary $\mathcal{B}$ has to provide valid $\mathsf{OPurchase}_l, \mathsf{ORefund}_l, \mathsf{OTransfer}_l, \mathsf{OValidate}_l$ oracles for $l \in \{0, 1\}$ to $\mathcal{A}$. The adversary $\mathcal{B}$ running the $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ experiment, it has access to the payment oracles OPay_0 and OPay_1, working respectively with the payment keys of $\mathcal{U}_0$ and the freshly generated payment keys (stk'_0, spk'_0). During the running of $\mathcal{B}$, an execution of the oracle $\mathsf{OPurchase}_0$ involves payments using the payment oracle OPay_0, whereas the execution of the oracle $\mathsf{OPurchase}_1$ involves payments using the payment key of user $\mathcal{U}_1$ owned by $\mathcal{B}$. Other oracles in $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ follows the same principle. Under the assumption that b' equals zero, when the challenger $\mathcal{C}$ running the $\mathsf{Exp}^{\mathsf{APUnl}}_{\mathcal{B}}$ experiment randomly chooses a bit b which equals zero then $\mathcal{B}$ runs the game G^0 since the used payment keys belong to $\mathcal{U}_0$. Still assuming that b' equals zero, when b equals one, then $\mathcal{B}$ runs the game G^1 in which the freshly generated payment keys replace the payment keys of $\mathcal{U}_0$. However, when b' equals one, $\mathcal{B}$ relies on the payment keys of user $\mathcal{U}_1$ during the $\mathsf{Purchase}$ protocol evaluation in the challenge phase. In this case, there is no difference between G^0 and G^1, replacing only the payment keys of $\mathcal{U}_0$. For this reason, when b' equals one, the adversary $\mathcal{A}$ has no advantage to distinguish.

The indistinguishability between these two games holds under the assumption that the used anonymous payment scheme ensures unlinkability. Suppose that our adversary $\mathcal{A}$ is the probability p_0 to win at the game G^0 and p_1 to fail at the game G^1. At the end of the $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ experiment, the adversary $\mathcal{B}$ receives a prediction bit b^* from $\mathcal{A}$ that is forwarded to the challenger $\mathcal{C}$. The probability of success of the adversary $\mathcal{B}$ based on

the adversary $\mathcal{A}$ against the challenger $\mathcal{C}$ running the $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{APUnl}}$ is defined as follows:

$$\begin{aligned}
\Pr[b = b^*] &= \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 1] \\
&+ \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 1] \\
&= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\
&+ \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 1] - \Pr[b^* = 0 | b = 1 \wedge b' = 1]) \\
&= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\
&= \frac{1}{2} + \frac{1}{4}(p_0 - p_1)
\end{aligned}$$

Hence, the advantage $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{APUnl}}$ equals $\frac{1}{4}(p_0 - p_1)$. Therefore, the indistinguishability between G^0 and G^1 is bounded by the upper-bound $4\mathsf{Adv}_{\mathcal{B}}^{\mathsf{APUnl}}$, or more formally:

$$\left|\Pr\left[\mathsf{G}_{\mathcal{A}}^0(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}_{\mathcal{A}}^1(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{APUnl}}$$

Game G^2. In this game, we proceed exactly as in G^1 except that we focus on the payment keys of $\mathcal{U}_1$. Since the indistinguishability between G^1 and G^2 holds the unlinkability of the anonymous payment scheme. And since the reduction follows the same methodology as done previously, we directly deduce that the following upper-bound:

$$\left|\Pr\left[\mathsf{G}_{\mathcal{A}}^1(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}_{\mathcal{A}}^2(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{APUnl}}$$

Game G^3. In this game, we proceed exactly as done in G^1 and G^2 except that we focus on the payment keys of $\mathcal{U}_0$, this time on the Refund and Transfer protocols (the Validate protocol does not require a payment). In all these protocols, we replace the payment keys of $\mathcal{U}_0$ by the generated payment keys (stk'_0, spk'_0) introduced in G^1. Observe that we have to replace payments keys of $\mathcal{U}_0$ in both protocols. We proceed to the replacement via a case disjunction, one for each protocol, the choice of the protocol to use is let to the adversary $\mathcal{A}$. In each case, the reduction methology is the same as seen previously. Hence, for each of these two cases (Refund and Transfer), we have the upper-bound $4\mathsf{Adv}_{\mathcal{B}}^{\mathsf{APUnl}}$. By summing the upper-bound for these two cases, we obtain the upper-bound to distinguish between G^2 and G^3, defined below:

$$\left|\Pr\left[\mathsf{G}_{\mathcal{A}}^2(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}_{\mathcal{A}}^3(1^\lambda) \to 1\right]\right| \leq 8 \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{APUnl}}$$

Game G^4. In this game, we replace the randomized signature and encryption keys of users $\mathcal{U}_0$ and $\mathcal{U}_1$ by freshly generated keys. Among the Purchase, Refund, Transfer and Validate protocols, only the Transfer and Validate protocols are modified. Indeed, both the validator in Validate and the other user in Transfer receive the randomized public key of $\mathcal{U}_b$ (for a $b \in \{0, 1\}$ chosen in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{UNL}}$). By the statistical indistinguishability of the randomised keys, we obtain statistical indistinguishability between G^3 and G^4:

$$\Pr\left[\mathsf{G}_{\mathcal{A}}^3(1^\lambda) \to 1\right] \overset{s}{\approx} \Pr\left[\mathsf{G}_{\mathcal{A}}^4(1^\lambda) \to 1\right]$$

Remark that in G^4, neither the public key of $\mathcal{U}_0$ or the public key of $\mathcal{U}_1$ are involved, all replaced by freshly generated keys being independent of both $\mathcal{U}_0$ and $\mathcal{U}_1$. Therefore, in G^4 the adversary $\mathcal{A}$ has no advantage to distinguish between $\mathcal{U}_0$ and $\mathcal{U}_1$ leading to the probability $\Pr\left[\mathsf{G}^4_{\mathcal{A}}(1^\lambda) \to 1\right] = \frac{1}{2}$. Therefore, by summing all the previously mentioned bounds, we conclude our sequence of games as follows:

$$\left|\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^4_{\mathcal{A}}(1^\lambda) \to 1\right]\right| = \left|\Pr\left[\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \leq 16 \cdot \mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}}$$

By hypothesis, for every polynomial time adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}}$ is negligible, meaning that the probability to distinguish is negligible as well. Hence, Applause provides unlinkability.

□

4.4 Auditability with Spotlight

As shown in Section 4.3, Applause provides full anonymity for users. This property is desirable in general, but there exist edge cases in which the full anonymity is not desirable, such as fire during an event. Identifying missing attendees is crucial in this context, but is hard using a fully anonymous ticketing system like Applause. To deal with this issue, we introduce a method to convert Applause into an *auditable* scheme called Spotlight, using Protego [CDLPK22], an auditable anonymous credential, based on *structure-preserving signatures on equivalence classes* [FHS19].

4.4.1 Overview of Protego

In a nutshell, Protego [CDLPK22] is an Attribute-Based Credentials (ABC) in which a user obtains a *credential* from an organisation, attesting that the user owns a particular set of attributes $\mathcal{X}$. Traditionally, this credential is transferred to a verifier, accepting the credential if and only if the provided credential is valid. In Protego, the "showing" of this credential slightly differs in order to provide several features on which we leverage to construct Spotlight: The credential is still provided by an organisation but is not shared with the verifier anymore. Instead, given its secret key and its credential $\mathcal{X}$, the user computes a proof attesting that it owns a subset of attributes $\mathcal{S} \subseteq \mathcal{X}$ and that it *not* own a set of attributes $\mathcal{D} \nsubseteq \mathcal{X}$. Even more, an accepted proof is valid if the proof has been generated by a user holding its private key. In other words, a user cannot create a proof using only public data from another user. The key point of Protego is that the proof is at the same time publicly verifiable, but privacy-preserving in the sense that the identity of the user having generated the proof is never involved in the proof verification. In other words, Protego ensures *anonymity of users*. Protego provides an additional feature at the heart of Spotlight: Auditability. An additional authority having its own key pair, is able to extract the identity of the user having generated the proof π. Before introducing Protego formally, we highlight that Protego supports revocation of the users. This revocation feature is not desired in our construction. For this reason, we voluntary omit RAKeyGen, RSetup and Revoke algorithms. We stress that this only limits the features of Protego and does not affect security.

Definition 31 (Protego [CDLPK22])**.** Assuming a security parameter λ and an universe of attributes Ω, then Protego is defined as follows:

- $\mathsf{Setup}(1^\lambda, aux) \to pp$: Given the security parameter λ and some auxiliary information defining the universe of attribute Ω, it outputs the public parameters pp.
- $\mathsf{OKeyGen}(pp) \to (osk, opk)$: Given the public parameters, it outputs the key pair for the organisation.
- $\mathsf{AAKeyGen}(pp) \to (ask, apk)$: Given the public parameters, it outputs the key pair for the auditor.
- $\mathcal{U}\mathsf{KeyGen}(pp) \to (usk, upk)$: Given the public parameters, it outputs the key pair for the user.
- $\mathsf{Issue}\langle \mathcal{U}(usk, \mathcal{X}, nym), \mathcal{O}(osk, \mathcal{X}, nym)\rangle \to \mathcal{U}(cred)$: The Issue protocol is running between a user $\mathcal{U}$ expecting a credential and the organisation $\mathcal{O}$ issuing the credential. The obtained credential *cred* authenticates the user with the set of attributes $\mathcal{X} \subset \Omega$. Note that even if we do not rely on the revocation feature, we have included the pseudonym *nym*, used only during the revocation, for completeness of the provided definition.
- $\mathsf{Show}(usk, cred, \mathcal{X}, \mathcal{S}, \mathcal{D}, aux) \to \pi$: The showing algorithm, taking the private key *usk* of the user $\mathcal{U}$, the credential *cred*, the set of attributes $\mathcal{X}$ and two sets $\mathcal{S}$ and $\mathcal{D}$, returns a proof π proving that the user $\mathcal{U}$ owns the attributes $\mathcal{S}$ and does not owns the attributes $\mathcal{D}$.
- $\mathsf{Verify}(opk, \pi, \mathcal{X}, \mathcal{S}, \mathcal{D}, aux) \to b$: Given the organization's public key *opk* and the proof π, it outputs the verification bit b which equals 1 if the proof is valid, 0 otherwise.
- $\mathsf{AuditEnc}(upk, apk) \to (c, \alpha)$: Given the user's public key *upk* and the auditor's public key *apk*, it outputs the encryption of the user's public key under the auditor's public key, and some auxiliary information α.
- $\mathsf{AuditDec}(ask, c) \to upk$: Given the auditor's private key *ask* and the ciphertext c, it outputs the user's public key.
- $\mathsf{AuditPrv}(usk, c, \alpha) \to \pi'$: Given the user's private key, the ciphertext c encrypting the user's public key under the auditor's public key and the auxiliary information α, it outputs a proof π'.
- $\mathsf{AuditVerify}(apk, \pi') \to b$: Given the auditor's public key and proof π', it outputs a bit b which equals one if the provided proof π' is valid, zero otherwise.

Auditability Algorithms in Protego. As the definition suggests, Protego is composed of algorithms used to generate and prove the ownership of a credential (*i.e.,* Issue, Show and Verif) and algorithms to perform the audit (*i.e.,* AuditEnc, AuditDec, AuditPrv and AuditVerify). While these two categories of algorithms seem at first glance disjoint in their functionalities, the construction of Protego reveals that the AuditEnc, AuditPrv and AuditVerify algorithms are subroutines of the Show and Verify algorithms. Indeed, in Protego the auditability-related ciphertext c and proof π' are generated and integrated in the computation of the proof π. To be more concrete, the ciphertext c and the proof π' are contained in the proof π. Note that π' is tied with π since π' is inputted into an hash function, whose obtained hash is used to compute π. A verifier running the Verify algorithm over the proof π has access to c and π'. Therefore, since

Symbol	Definition
$\mathcal{N}$	Set of all pseudonyms
NYM	Set of assigned pseudonyms
HU	Set of honest users
CU	Set of corrupted users
USK	Private keys of users
UPK	Public keys of users
CRED	List of credentials
OWNR	Owner of the credential ($\|\mathsf{OWNR}\| = \|\mathsf{CRED}\|$)
ATTR	Attributes authenticated by the credentials ($\|\mathsf{ATTR}\| = \|\mathsf{CRED}\|$)
I_{LoR}	Users involved in the challenge
J_{LoR}	Credentials involved in the challenge

Table 4.1: Definition of symbols used in the anonymity property of Protego.

AuditEnc, AuditPrv and AuditVerify are subroutines of Show and Verify, we remove these three algorithms and introduce $\mathsf{Audit}(usk, \pi)$ which extracts c from π and returns the output of $\mathsf{AuditDec}(usk, c)$. This modification does not alter the security, in particular the anonymity property in which no oracle for the audit is considered in [CDLPK22]. Hence, this modification is purely functional and does not affect the security at all.

Correctness of Protego. The correctness of Protego states an honestly generated proof π for any security parameter $\lambda > 0$, for any set of attributes $\mathcal{X}$ with $0 < |\mathcal{X}| \leq q$, for any non-empty subset of attributes $\mathcal{S} \subset \mathcal{X}$, any non-empty disjoint set of attributes $\mathcal{D} \nsubseteq \mathcal{X}$, for any non-empty set of valid pseudonyms $\mathsf{NYM} \subseteq \mathcal{N}$ with $0 < \mathcal{N} \leq q'$, for all $nym \in \mathsf{NYM}$ it holds that:

$$
\begin{aligned}
pp &\leftarrow \mathsf{Setup}(1^\lambda, (1^q, 1^{q'}));\\
(ask, apk) \leftarrow \mathsf{AAKeyGen}(pp); (osk, opk) &\leftarrow \mathsf{OKeyGen}(pp), (usk, upk) \leftarrow \mathcal{U}\mathsf{KeyGen}(pp);\\
\mathsf{Issue}\langle \mathcal{U}(usk, \mathcal{X}, nym), \mathcal{O}(osk, \mathcal{X}, nym)\rangle &\rightarrow \mathcal{U}(cred)\\
\pi \leftarrow \mathsf{Show}(usk, cred, \mathcal{S}, \mathcal{D}); 1 &\leftarrow \mathsf{Verify}(opk, \pi, \mathcal{X}, \mathcal{S}, \mathcal{D})
\end{aligned}
$$

Anonymity in Protego. The anonymity property of Protego states that an adversary is not able to distinguish the showings of multiple proofs π coming from a user $\mathcal{PU}_0$ or a user $\mathcal{PU}_1$. In other words, the proof is not tied with the identity of the user having generated the proof. The security definition of anonymity introduces five oracles OHonestUser, OCorrUser, OIssue, OShow and OLoR that we describe in the following paragraphs. All important symbols are summarised in Table 4.1.

Oracles for Honest and Corrupted Users. During the execution of the anonymity experiment, the challenger provides two oracles OHonestUser and OCorrUser depicted in Figure 4.25 to respectively create honest users and corrupt users.

The oracle OHonestUser expects as an input from the adversary an integer i, which has to be read as the i-th user. The oracle aborts if i has already be assigned to a user. This test is performed using the set of honest users HU and corrupted users CU. In case where the index i has not been assigned yet, then the oracle generates a new user key

pair (usk, upk) added into the sets USK and UPK respectively, and it adds the index i in HU. The freshly generated user's public key upk is returned to the adversary.

The oracle $\mathsf{OCorrUser}$ takes as an input an index i as well as a user public key upk. The oracle checks that the i-th user has not already been corrupted (otherwise no additional work is necessary), but also checks that i does not belong to the I_{LoR} set. In a nutshell, this set contains all users involved in the OLoR oracle, as will see later. If the user is honest, then the oracle transfers the index i from the set HU to the set CU and leaks to the adversary the private key of the user as well as all its obtained credentials. In case where the index i does not belong to the set $\mathsf{HU} \cup \mathsf{CU}$, the user has not been created by the challenger. In this case, the challenger creates a corrupted user whose secret key is not known and whose public key is the one provided by the adversary.

Oracle $\mathsf{OHonestUser}(i)$

1 : **if** $i \in \mathsf{HU} \cup \mathsf{CU}$ **then return** $\perp$
2 : $(\mathsf{USK}[i], \mathsf{UPK}[i]) \leftarrow \mathcal{U}\mathsf{KeyGen}(pp)$
3 : **add** i **in** HU
4 : **return** $\mathsf{UPK}[i]$

Oracle $\mathsf{OCorrUser}(i, upk)$

1 : **if** $i \in \mathsf{CU} \vee i \in I_{\mathsf{LoR}}$ **then return** $\perp$ */* Corrupted or involved in the challenge */*
2 : **if** $i \in \mathsf{HU}$ **then** */* Corruption of an honest user */*
3 : **remove** i **from** HU
4 : **add** i **in** CU
5 : $\mathcal{C} \leftarrow \{\mathsf{CRED}[j]\}$**for all** j **where** $\mathsf{OWNR}[j] = i$
6 : **return** $(\mathsf{USK}[i], \mathcal{C})$ */* Leakage of user's private key and obtained credentials */*
7 : **else** */* $i \notin \mathsf{HU} \wedge i \notin \mathsf{CU}$ */*
8 : **add** i **in** CU
9 : $\mathsf{UPK}[i] \leftarrow upk$

Figure 4.25: Description of the $\mathsf{OHonestUser}$ and $\mathsf{OCorrUser}$ oracles.

Oracles for Credential Issuing and Showing. The anonymity of Protego allows the adversary to ask the i-th user to obtain a credential for a set of attributes $\mathcal{X}$ of its choice, but also to obtain a proof π for the credential of its choice with respect to a set of owned $\mathcal{S}$ and disjoint attributes $\mathcal{D}$. These two features are modelled respectively as the oracles OIssue and OShow depicted in Figure 4.26.

The oracle OIssue expects as an input from the adversary an index i (corresponding to the i-th user), a pseudonym nym and a set of attributes $\mathcal{X}$. The oracle runs the Issue protocol only for honest users with an organisation played by the adversary $\mathcal{A}$. For this reason, the user has to be honest. At condition that the Issue protocol ended successfully, the generated credential $cred$ is inserted in the CRED set containing all the generated credentials for all users. To link the ownership of the credential, the index i is inserted in the OWNR set. The oracle also inserts the attributes $\mathcal{X}$ in the ATTR set and the used pseudonym nym in the NYM set.

The oracle OShow expects as an input from the adversary the index j referring to the j-th credential in the CRED set, as well as two sets of attributes $\mathcal{S}$ and $\mathcal{D}$. The

oracle runs the Show algorithm for the i-th user being the owner the j-th credential, resulting into a proof π that is returned to the adversary. The i-th user is required to be honest, otherwise leading to an abort of the oracle.

Oracle $\mathsf{OIssue}(i, nym, \mathcal{X})$

1 : **if** $i \notin \mathsf{HU}$ **then return** $\perp$
2 : $\mathsf{Issue}\langle \mathcal{U}(\mathsf{USK}[i], nym, \mathcal{X}), \mathcal{A}\rangle \rightarrow \mathcal{U}(cred)$
3 : **if** $cred = \perp$ **then return** $\perp$
4 : **add** $(i, cred, \mathcal{X}, nym)$ **in** $(\mathsf{OWNR}, \mathsf{CRED}, \mathsf{ATTR}, \mathsf{NYM})$

Oracle $\mathsf{OShow}(j, \mathcal{S}, \mathcal{D})$

1 : $i \leftarrow \mathsf{OWNR}[j]$
2 : **if** $i \notin \mathsf{HU}$ **then return** $\perp$
3 : $\pi \leftarrow \mathsf{Show}(\mathsf{USK}[i], \mathsf{CRED}[j], \mathsf{ATTR}[j], \mathcal{S}, \mathcal{D})$
4 : **return** π

Figure 4.26: Description of the OIssue and OShow oracles.

Left-or-Right Oracle. The Left-or-Right oracle, denoted as OLoR and depicted in Figure 4.27, allows the adversary to obtain a proof π for one among two credentials indexed respectively by j_0 and j_1. Hence, the oracle OLoR expects as an input j_0 and j_1, as well as the two sets of attributes $\mathcal{S}$ and $\mathcal{D}$ inputted into the proof generation algorithm. The oracle works using two specific sets denoted I_{LoR} and J_{LoR}. The first set I_{LoR} of cardinality two contains the index of (honest) users holding the credentials referred by the index j_0 and j_1. These two credential indexes are saved in the J_{LoR} set. The anonymity property holds even in case of multiple showings of the same credential. This is exactly for this reason, that the challenge takes the form of a Left-or-Right oracle. However, the credentials has to be the same over the time, which explains why the OLoR oracle aborts when the adversary inputs a pair of credentials (j_0', j_1') different of (j_0, j_1). The oracle also aborts in case where at least one user holding either $\mathsf{CRED}[j_0]$ or $\mathsf{CRED}[j_1]$ is corrupted. Indeed, in such case, no security can be proven. Additional verifications are performed by the oracle on the provided sets of attributes $\mathcal{S}$ and $\mathcal{D}$, required to satisfy the attributes $\mathsf{ATTR}[j_0]$ and $\mathsf{ATTR}[j_1]$. The oracle generates a proof π for the credential $\mathsf{CRED}[j_b]$ for the i_b-th user where b is randomly chosen by the challenger at the beginning of the experiment, once and for all the oracle executions.

Description of Anonymity in Protego. The experiment for the anonymity of Protego denoted $\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{A}}$ and depicted in Figure 4.28, takes as an input the security parameter 1^λ as well as two integers q and q', limiting respectively the number of defined attributes and pseudonyms. The challenger running the experiment initiates with the Setup execution to generate the public parameters pp. The auditor key pair is generated, only the auditor's public key is shared with the adversary. A random bit b is chosen by the challenger and runs the adversary with an access to the oracles OHonestUser, OCorrUser, OIssue, OShow and OLoR. Given all this oracles, the adversary eventually returns a prediction bit b^* and wins the anonymity experiment if b equals b^*.

Oracle $\mathsf{OLoR}(j_0, j_1, \mathcal{S}, \mathcal{D})$

1 : **if** $J_{\mathsf{LoR}} \neq \emptyset \wedge J_{\mathsf{LoR}} \neq \{j_0, j_1\}$ **then return** $\perp$
2 : $(i_0, i_1) \leftarrow (\mathsf{OWNR}[j_0], \mathsf{OWNR}[j_1])$
3 : **if** $i_0 \notin \mathsf{HU} \vee i_1 \notin \mathsf{HU}$ **then return** $\perp$
4 : **if** $\mathcal{S} \not\subseteq \mathsf{ATTR}[j_0] \cap \mathsf{ATTR}[j_1]$ **then return** $\perp$
5 : **if** $\mathcal{D} \cap (\mathsf{ATTR}[j_0] \cup \mathsf{ATTR}[j_1]) \neq \emptyset$ **then return** $\perp$
6 : $J_{\mathsf{LoR}} \leftarrow \{j_0, j_1\}$
7 : $I_{\mathsf{LoR}} \leftarrow \{i_0, i_1\}$
8 : $\pi \leftarrow \mathsf{Show}(\mathsf{USK}[i_b], \mathsf{CRED}[j_b], \mathsf{ATTR}[j_b], \mathcal{S}, \mathcal{D})$
9 : **return** π

Figure 4.27: Description of the OLoR oracle.

$\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{A}}(1^\lambda, q, q')$

1 : $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^q, 1^{q'})$
2 : $(ask, apk) \leftarrow \mathsf{AAKeyGen}(pp)$
3 : $b \xleftarrow{\$} \{0,1\}$
4 : $b^* \leftarrow \mathcal{A}^{\mathsf{OHonestUser},\mathsf{OCorrUser},\mathsf{OIssue},\mathsf{OShow},\mathsf{OLoR}}(apk)$
5 : **return** $b \stackrel{?}{=} b^*$

Figure 4.28: Description of the anonymity experiment of Protego.

Definition 32 (Anonymity of the Simplified Protego)**.** Protego is said anonymous if for security parameter λ, every $q, q' > 0$ and every polynomial-time adversary $\mathcal{A}$, we have:

$$\mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}} = \left| \Pr\left[\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{A}}(1^\lambda, (1^q, 1^{q'})) \rightarrow 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

4.4.2 Description of Spotlight

We start by giving the description of our auditable anonymous ticketing system called Spotlight, standing in contrast with Applause, by allowing users to be authenticated by an additional, incorruptible authority called the judge and denoted $\mathcal{J}$. In a nutshell, the judge owns the key pair of both the organisation and the auditor as defined in the Protego definition. Therefore, the judge plays at the same time the role of the organisation delivering the credential and the role of the auditor revealing the identity of an attendee. Spotlight corresponds mostly to our original protocol Applause, the only difference being the integration of Protego used to generate a proof during the ticket validation protocol. The integration also supposes that during the key pair generation, a user $\mathcal{U}$ generates a Protego key pair (usk, upk), and runs the certification protocol $\mathsf{Certify}$ with the judge $\mathcal{J}$ to obtain a credential $cred$. Then, at every interaction, the user $\mathcal{U}$ generates a new proof π that is integrated in the communication. We will explain the integration of the proof later. Once received by the system, the proof π is stored and associated with the (purchased, transferred, refund or validated) ticket. Later, given a ticket tk containing the proof, the judge is allowed to recover the public key of the user. Observe here an interesting fact about the integration of Protego: The

desired degree of auditability can be modified depending on the integration of the proof π in the ticketing system. In other words, if every protocol execution (*i.e.,* Purchase, Transfer, Refund and Validate) contains the proof, then every action can be linked to the user initiating the protocol, only from the view of the auditor authority. Another possible integration of the proof followed in this work, is to integrate the proof π only during the ticket validation protocol Validate. This way, only users attending the event can be identified from the view of the $\mathcal{J}$. It follows that a user having purchased a ticket (or received from another user via the ticket transfer) cannot be identified if it does not attend the event. As explained during the introduction of this work, our ambition is to identify attendees of the event in case of a disaster and only this case. Hence, we will focus our attention on integrating the proof π only in the ticket validation protocol.

Dedicated State for Auditability. In APPLAUSE, the state st shared among all parties, contains hashes c of tickets. In SPOTLIGHT, we require a differently formatted state to store both hashes, used to ensure validity of tickets, but also to store received proofs during the validation. Formally, we defined the shared state st as the couple of states $(\mathsf{st}_c, \mathsf{st}_\pi)$, used respectively to store hashes c and to store couples of the form (c, π). We stress that our hypothesis on the public readability still holds, meaning that the judge $\mathcal{J}$ is able to read and reveal the identity of users attending the event.

Formal Description of SPOTLIGHT. To obtain auditability, we modify the $\mathcal{U}$KeyGen and Validate algorithms introduced in APPLAUSE, and introduce the two new algorithms $\mathcal{J}$KeyGen and Audit, used respectively to generate the key pair for the judge and to recover the public key of the user $\mathcal{U}$. In order to facilitate the readability, we omit the $\mathcal{D}$KeyGen, $\mathcal{T}$KeyGen, $\mathcal{V}$KeyGen, Purchase and Transfer definitions, which remain unchanged. For clarity, all algorithms and protocols referring to the Protego scheme are prefixed with PTGO.

- $\underline{\mathcal{J}\mathsf{KeyGen}(1^\lambda) \to (sk_\mathcal{J}, pk_\mathcal{J})}$: Given the unary representation of the security parameter λ, the judge key generation algorithm starts by generating the public parameters of the Protego scheme $\mathsf{PTGO.Setup}(1^\lambda, (1^q, 1^{q'}))$ producing the public parameters pp. The upper-bound q on the number of existing attributes is set at two while the upper-bound q' of the number of pseudonyms is set two one. We have chosen these values to respect the requirements to satisfy the correctness definition of Protego. Indeed, in SPOTLIGHT, we do not rely on the attributes property of Protego. A natural choice for the universe of attributes might be $\Omega = \{\texttt{user}\}$, but recall that Protego requires two non-empty sets $\mathcal{S}$ and $\mathcal{D}$ to generate a proof for a credential attesting the ownership of attributes $\mathcal{X}$. The set $\mathcal{S} \subseteq \mathcal{X}$ is supposed to contain attributes that are owned by the user having the credential *cred*, whereas $\mathcal{D} \nsubseteq \mathcal{X}$ contains the attributes not authenticated by the credential. And since $\mathcal{D}$ cannot be the empty set, we have to consider an additional attribute, never assigned and always inserted in $\mathcal{D}$. Therefore, the universe of attributes is defined as $\{x, \hat{x}\}$, which explains the upper-bound q at two for the possible attributes. The upper-bound q' is set to one since we only rely on a single pseudonym for all existing users. Note that pseudonyms are only used for the revocation feature that we do not require, and a pseudonym can be replayed several times without

loss of correctness and security as suggested by the definition of Protego. Hence we set the constant pseudonym $\mathtt{nym}$ once and for all users.

In addition to the setup of Protego, this algorithm generates the organisation key pair $(osk, opk) \leftarrow \mathsf{PTGO.OKeyGen}(1^\lambda)$ and the auditor key pair $(ask, apk) \leftarrow \mathsf{PTGO.AAKeyGen}(1^\lambda)$. The outputted key pair $(sk_{\mathcal{J}}, pk_{\mathcal{J}})$ where $sk_{\mathcal{J}}$ is composed of (osk, ask) and where $pk_{\mathcal{J}}$ is composed of (opk, apk, pp).

- $\underline{\mathcal{U}\mathsf{KeyGen}(1^\lambda) \rightarrow (sk_{\mathcal{U}}, pk_{\mathcal{U}})}$: Given the unary representation of the security parameter λ, this algorithm computes a signature key pair denoted $(sk_{\mathcal{U}}^{\mathsf{Sign}}, pk_{\mathcal{U}}^{\mathsf{Sign}}) \leftarrow \mathsf{Sign.KeyGen}(1^\lambda)$, as well as an encryption key pair denoted $(sk_{\mathcal{U}}^{\mathsf{PKE}}, pk_{\mathcal{U}}^{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$. In addition, a payment key $(stk_{\mathcal{U}}^{\mathsf{AP}}, spk_{\mathcal{U}}^{\mathsf{AP}}, pd_{\mathcal{U}}^{\mathsf{AP}})$ is generated using the anonymous payment key generation algorithm $\mathsf{AP.KeyGen}(1^\lambda)$. It runs the Protego's user key generation algorithm $\mathsf{PTGO}.\mathcal{U}\mathsf{KeyGen}(pp)$ of Protego to obtain the key pair (usk, upk). Once the user's key pair obtained for the Protego scheme, the credential issuing protocol $\mathsf{Issue}\langle\mathcal{U}(usk, \mathcal{X}, \mathtt{nym}), \mathcal{J}(osk, \mathcal{X}, \mathtt{nym})\rangle \rightarrow \mathcal{U}(cred)$ in order to obtain the credential $cred$ for the set of attributes $\mathcal{X} = \{x\}$. Finally, it returns the key pair $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ where the secret key $sk_{\mathcal{U}}$ is composed of $(sk^{\mathsf{Sign}}, sk^{\mathsf{PKE}}, stk^{\mathsf{AP}}, spk^{\mathsf{AP}}, usk, cred, \mathcal{X})$ and where the public key $pk_{\mathcal{U}}$ is composed of $(pk^{\mathsf{Sign}}, pk^{\mathsf{PKE}}, pd^{\mathsf{AP}}, upk)$. We stress that the $\mathsf{RandKey}$ ignores keys Protego's keys in the outputted randomised key.

The required modification in the ticket validation protocol for the user consists of generatating a proof π, integrated in the (already existing) ciphertext encrypting the randomised public key and the ticket. On its side the validator, in addition to the existing validations, verifies the validity of the proof. If the proof is valid, then it stores the hash c and the proof π in the shared state st_π.

- $\underline{\mathsf{Validate}\langle\mathcal{U}(sk_{\mathcal{U}}, \mathsf{tk}), \mathcal{V}(sk_{\mathcal{V}}, \mathsf{st})\rangle}$: The validation protocol starts with the validator $\mathcal{V}$, providing its public key $pk_{\mathcal{V}}$ along its certificate $cert_{\mathcal{V}}$ to the user $\mathcal{U}$ attempting to attend the event. Recall that the public key $pk_{\mathcal{V}}$ contains the public encryption key $pk_{\mathcal{V}}^{\mathsf{PKE}}$ and the public signature key $pk_{\mathcal{V}}^{\mathsf{Sign}}$. The authenticity of the validator public key $pk_{\mathcal{V}}$ is verified by the user $\mathcal{U}$ using the signature verification algorithm $\mathsf{Verif}(pk_{\mathcal{D}}^{\mathsf{Sign}}, (pk_{\mathcal{V}}^{\mathsf{Sign}}, pk_{\mathcal{V}}^{\mathsf{PKE}}), cert_{\mathcal{V}})$. If the signature is rejected, the user $\mathcal{U}$ aborts the verification protocol. Otherwise, the user $\mathcal{U}$ generates a proof π using the Protego's proof generation algorithm $\mathsf{PTGO.Show}(usk, cred, \mathcal{X}, \mathcal{S}, \mathcal{D})$ with the set of attributes $\mathcal{X} = \{x\}$, the set $\mathcal{S} = \{x\}$ and with the set $\mathcal{D} = \{\hat{x}\}$. The user also runs the key randomisation algorithm $\mathsf{RandKey}(sk_{\mathcal{U}}, pk_{\mathcal{U}})$ to obtain a new key pair $(sk_{\mathcal{U}'}, pk_{\mathcal{U}'})$. The ciphertext $\mathsf{Enc}(pk_{\mathcal{V}}^{\mathsf{PKE}}, pk_{\mathcal{U}'}, \mathsf{tk}, \pi)$ encrypting the randomised public key $pk_{\mathcal{U}'}$, the ticket tk and the proof π, is sent to the validator $\mathcal{V}$. After decryption, the validator executes the ticket verification $\mathsf{CheckTk}(\mathsf{tk}, \mathsf{st})$ and the proof verification $\mathsf{PTGO.Verify}(opk, \pi, \mathcal{X}, \mathcal{S}, \mathcal{D})$. If both the ticket and the proof are valid, the validator $\mathcal{V}$ inserts the hash c computed from the received ticket tk in the state of invalid ticket st, and creates a *challenge* s sampled from $\{0,1\}^\lambda$, which is carefully encrypted under the provided randomised public encryption key $pk_{\mathcal{U}'}^{\mathsf{PKE}}$. The resulting ciphertext, denoted ψ_s, is signed by the signature key $sk_{\mathcal{V}}^{\mathsf{Sign}}$ of the validator $\mathcal{V}$, producing the signature σ_s. The ciphertext ψ_s and the signature σ_s are sent as a response to $\mathcal{U}$. After a necessary verification of the authenticity of the signature σ_s, and the decrypted challenge s', values s and s' are compared through

a physical channel. If the verification fails, then the validator $\mathcal{V}$ removes the hash c from the set of invalid tickets st and halts. Otherwise, the validator inserts the couple (c, π) in the shared state st_π and both parties commonly terminates with a success bit b at $\top$.

- $\underline{\mathsf{Audit}(sk_{\mathcal{J}}, \pi) \to pk_{\mathcal{PU}}}$: The audit algorithm, run by the judge $\mathcal{J}$, expects as an input the secret key $sk_{\mathcal{J}}$ of the judge, but also a proof π. Observe that by hypothesis on the public readability of the shared state st, and hence st_π by construction, the judge can obtain the proof π by its own. To recover the Protego's public key upk of the user $\mathcal{U}$, the judge returns the output of $\mathsf{PTGO.Audit}(ask, \pi)$.

4.4.3 Security Proofs of Spotlight

Because of the clear similarity between Applause and Spotlight, most of the security claims remain unchanged. This is particularly true for the unforgeability of the ticket, ticket privacy and prevention of double spending property, which is not affected by this modification. Indeed, the added authentication material, consisting of new keys from Protego, a credential kept secret by the user, and a proof π sent to the validator. Since all these modifications does not affect the state (at least, not the state st_c used to verify validity of the ticket), the double-spending remains prevented. Similarly, the proof π is completely independent of any random r_c and the hash c is already in the view of the adversary and hence does not bring any additional advantage. As a result, the ticket privacy is not impacted. Finally, the signature mechanism, used to authenticate approved tickets, remains unchanged, hence provides unforgeability.

On the other side, the pseudonymity property, have to be restated for Spotlight. Indeed, we need to guarantee that the additional authentication and auditing material does not affect the pseudonymity of users. Recall that the pseudonymity property of an auditable ETS scheme remains *exactly* the same compared to a traditional ETS scheme, as no additional security property and no oracle are added. In particular, the judge is simulated by the challenger and does not provide any auditing oracle to the adversary.

Theorem 12. Instantiated with statistically indistinguishable randomisable keys, a secure anonymous payment scheme ensuring pseudnonimity and Protego, for every security parameter $\lambda \in \mathbb{N}$ and every polynomial-time adversary, Spotlight provides *pseudonymity* under probability:

$$\left| \Pr\left[\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2} \right| \leq 8 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}(1^\lambda) + 8 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}(1^\lambda)$$

Proof of Pseudonymity. By the construction of Spotlight, sharing many similarities with Applause, the pseudonymity security proof remains almost identical. For this reason, we omit the five first games in which no difference occurs. Hence, following our pseudonymity proof of Applause, the advantage to distinguish between G^0 and G^4 is

negligible based on the upper-bound defined as follows:

$$\left|\Pr\left[\mathsf{G}^0(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^4(1^\lambda) \to 1\right]\right| = \left|\Pr\left[\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \leq 8 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}$$

Game G^5. At this point, the two users $\mathcal{U}_0$ and $\mathcal{U}_1$ are still identified by their respective credentials denoted $cred_0$ and $cred_1$. In this game, we introduce a new key pair (usk'_0, upk'_0) and its associated credential $cred'_0$. Later, any showing of the proof π generated via the credential $cred_0$ of $\mathcal{U}_0$ is now performed using the credential $cred'_0$ being independent of the identity of $\mathcal{U}_0$.

By contradiction, suppose there exists an adversary $\mathcal{A}$ for which the advantage in distinguishing between G^4 and G^5 is non-negligible. Specifically, let p_0 be the probability of success of $\mathcal{A}$ in G^4 and p_1 be the probability of failure of $\mathcal{A}$ in G^5. Denote by ϵ the distinguishing probability $p_0 - p_1$.

We construct an adversary $\mathcal{B}$ which utilizes $\mathcal{A}$ to break the anonymity of Protego. Let $\mathcal{C}$ be the challenger running the $\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{B}}$ against the adversary $\mathcal{B}$. The challenger $\mathcal{C}$ holds the key pairs (usk_0, upk_0) and (usk'_0, upk'_0) as well as the credentials $cred_0$ and $cred'_0$. The adversary $\mathcal{B}$ obtains the keys (usk_1, upk_1) and the credential $cred_1$ of the user $\mathcal{U}_1$, via the execution of the honest user creation followed by the corruption of the user. During the execution of the $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$ experiment, especially during the execution of the $\mathsf{Validate}$ protocol in the $\mathsf{OValidate}_0$ oracle, the adversary $\mathcal{B}$ obtains a showing of the credential $cred_0$ using the OShow oracle provided by the challenger $\mathcal{C}$ in the $\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{B}}$ experiment. The execution of the $\mathsf{OValidate}_1$ oracle, on its side, is completely simulated by $\mathcal{B}$ having the secret key usk_1 of $\mathcal{U}_1$ and its credential $cred_1$, hence $\mathcal{B}$ is able to compute any showing if its choice. Note that the OShow oracle expects the index of a credential. By construction, only three credentials are involved: Two credentials $cred_0$ and $cred'_0$ hold by the challenger $\mathcal{C}$ and a credential $cred_1$ holds by the adversary $\mathcal{B}$.

At the beginning of $\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}$ experiment simulated by $\mathcal{B}$, a random bit b' is chosen designating between the user $\mathcal{U}_0$ (whose protego keys and credential are held by $\mathcal{C}$) and the user $\mathcal{U}_1$, supposed to interact with $\mathcal{A}$. In addition, at some point in the execution of the $\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{B}}$ experiment, a random bit b is chosen by $\mathcal{C}$ designating the credential to use between $cred_0$ and $cred'_0$. Without loss of generality, suppose that $\mathcal{C}$ uses $cred_0$ when b is zero, or uses $cred'_0$ otherwise. Observe that when b' equals one, the adversary $\mathcal{A}$ interacts with $\mathcal{U}_1$ and hence has no chance to guess the bit b. However, when b' equals zero, then $\mathcal{A}$ is interacting with $\mathcal{U}_0$, whose Protego keys and credential, held by $\mathcal{C}$ are either the real one (when b equals zero) or an independent keys and credential (when b equals one).

At the end of the experiment, the adversary $\mathcal{A}$ responds with a prediction bit b^* to $\mathcal{B}$ which is forwarded to $\mathcal{C}$. The probability of a correct guess, defined by the probability

$\Pr[b = b^*]$, follows:

$$\begin{aligned}\Pr[b = b^*] &= \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 1] \\ &+ \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 1] \\ &= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\ &+ \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 1] - \Pr[b^* = 0 | b = 1 \wedge b' = 1]) \\ &= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\ &= \frac{1}{2} + \frac{1}{4}(p_0 - p_1) = \frac{1}{2} + \frac{1}{4}\epsilon\end{aligned}$$

Therefore, assuming an adversary $\mathcal{A}$ having a non-negligible advantage ϵ to distinguish between G^4 and G^5, then there exist an adversary $\mathcal{B}$ able to break the anonymity of Protego with a non-neglible advantage $\frac{\epsilon}{4}$. Therefore, we have the following distinguishing advantage:

$$\left|\Pr\left[\mathsf{G}^4_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^5_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$$

Game G^6. In this game, we apply the exact same methology except that we now replace the credential $cred_1$ with a freshly generated credential $cred'_1$ obtained from an independent Protego user key pair (usk'_1, upk'_1). Following the same reduction, we obtain the following distinguishing advantage:

$$\left|\Pr\left[\mathsf{G}^5_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^6_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$$

At this point, the public key of both users $\mathcal{U}_0$ and $\mathcal{U}_1$ are not used anymore. It includes the signature key, the anonymous payment keys, the encryption key but also the Protego keys. Therefore, the success probability in game G^6 formally defined as $\Pr\left[\mathsf{G}^6(1^\lambda) \to 1\right]$ equals $\frac{1}{2}$. As a result, the distinguishing advantage between our initial game G^0 and our game G^6 is defined as follows:

$$\begin{aligned}\left|\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^6_{\mathcal{A}}(1^\lambda) \to 1\right]\right| &= \left|\Pr\left[\mathsf{Exp}^{\mathsf{PSE}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \\ &\leq 8 \cdot \mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}} + 8 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}\end{aligned}$$

By assumption, the advantages $\mathsf{Adv}^{\mathsf{APPse}}_{\mathcal{A}}$ and $\mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$ are negligible and hence, SPOTLIGHT provides pseudonymity.

□

Theorem 13. Instantiated with an anonymous payment ensuring pseudonimity, Protego, and randomisable keys, then for any PPT $\mathcal{A}$, SPOTLIGHT provides *unlinkability* under the following probability:

$$|\Pr[\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}(1^\lambda) \to 1] - \frac{1}{2}| \leq 16 \cdot \mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}} + 8 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$$

Proof. Because of the proximity between SPOTLIGHT and APPLAUSE, the unlinkability proof of SPOTLIGHT shares similarities with the proof of APPLAUSE. More concretely, we omit the five first games in which no difference occurs and extend the sequence of games to take in account Protego keys and showings in the Validate protocol. Recall that the unlinkability proofs of APPLAUSE is composed of five games ranging from G^0 to G^4 whose distinguishing advantage is upper-bounded as follows:

$$\left|\Pr\left[\mathsf{G}^0_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^4_{\mathcal{A}}\right]\right| \leq 16 \cdot \mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}}$$

Game G^5. In the previous game, all anonymous payements, signature and encryption keys, the only cryptographic material being linked to the identity of a user, are currently not used and replaced by completely independent keys. Only the Protego material remain, especially the triplet $(usk_0, upk_0, cred_0)$ and $(usk_1, upk_1, cred_1)$ associated respectively with user $\mathcal{U}_0$ and $\mathcal{U}_1$.

In this game, during the challenge phase in which the advantage $\mathcal{A}$ interacts only with $\mathcal{U}_b$ for some bit b chosen in the experiment $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$, we replace $(usk_0, upk_0, cred_0)$ associated by the user $\mathcal{U}_0$ by the triplet $(usk'_0, upk'_0, cred'_0)$ being independent of $\mathcal{U}_0$. Note that the triplet associated to $\mathcal{U}_1$ is still used.

For contradiction, suppose there exists an adversary $\mathcal{A}$ for which the advantage in distinguishing between G^4 and G^5 is non-negligible. Specifically, let p_0 be the probability of success of $\mathcal{A}$ in G^4 and p_1 be the probability of failure of $\mathcal{A}$ in G^5. Denote by ϵ the distinguishing probability $p_0 - p_1$.

We construct an adversary $\mathcal{B}$ which utilizes $\mathcal{A}$ to break the anonymity of Protego. Let $\mathcal{C}$ be the challenger running the $\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{B}}$ against the adversary $\mathcal{B}$. The challenger $\mathcal{C}$ holds the key pairs (usk_0, upk_0) and (usk'_0, upk'_0) as well as the credentials $cred_0$ and $cred'_0$. The adversary $\mathcal{B}$ obtains the keys (usk_1, upk_1) and the credential $cred_1$ of the user $\mathcal{U}_1$, via the execution of the honest user creation oracle OHonestUser followed by the corruption of the user via the user corruption oracle OCorrUser. During the execution of the $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ experiment, especially during the execution of the Validate protocol in the $\mathsf{OValidate}_0$ oracle, the adversary $\mathcal{B}$ obtains a showing of the credential $cred_0$ using the OShow oracle provided by the challenger $\mathcal{C}$ in the $\mathsf{Exp}^{\mathsf{ProAno}}_{\mathcal{B}}$ experiment. The execution of the $\mathsf{OValidate}_1$ oracle, on its side, is completely simulated by $\mathcal{B}$ having the secret key usk_1 of $\mathcal{U}_1$ and its credential $cred_1$, hence $\mathcal{B}$ is able to compute any showing if its choice. Note that the OShow oracle expects the index of a credential. By construction, only three credentials are involved: Two credentials $cred_0$ and $cred'_0$ hold by the challenger $\mathcal{C}$ and a credential $cred_1$ holds by the adversary $\mathcal{B}$.

During the challenge phase in the $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$ experiment simulated by $\mathcal{B}$, a random bit b' is chosen designating between the user $\mathcal{U}_0$ (whose protego keys and credential are held

by $\mathcal{C}$) and the user $\mathcal{U}_1$, supposed to interact with $\mathcal{A}$. In addition, at some point in the execution of the $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{ProAno}}$ experiment, a random bit b is chosen by $\mathcal{C}$ designating the credential to use between $cred_0$ and $cred'_0$. Without loss of generality, suppose that $\mathcal{C}$ uses $cred_0$ when b is zero, or uses $cred'_0$ otherwise. Observe that when b' equals one, the adversary $\mathcal{A}$ interacts with $\mathcal{U}_1$ and hence has no chance to guess the bit b. However, when b' equals zero, then $\mathcal{A}$ is interacting with $\mathcal{U}_0$, whose Protego keys and credential, held by $\mathcal{C}$ are either the real one (when b equals zero) or an independent keys and credential (when b equals one).

At the end of the experiment, the adversary $\mathcal{A}$ responds with a prediction bit b^* to $\mathcal{B}$ which is forwarded to $\mathcal{C}$. The probability of correct guess, defined by the probability $\Pr[b = b^*]$, follows:

$$\begin{aligned}
\Pr[b = b^*] &= \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 0 | b = 0 \wedge b' = 1] \\
&+ \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 0] + \frac{1}{4}\Pr[b^* = 1 | b = 1 \wedge b' = 1] \\
&= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\
&+ \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 1] - \Pr[b^* = 0 | b = 1 \wedge b' = 1]) \\
&= \frac{1}{2} + \frac{1}{4}(\Pr[b^* = 0 | b = 0 \wedge b' = 0] - \Pr[b^* = 0 | b = 1 \wedge b' = 0]) \\
&= \frac{1}{2} + \frac{1}{4}(p_0 - p_1) = \frac{1}{2} + \frac{1}{4}\epsilon
\end{aligned}$$

Therefore, assuming an adversary $\mathcal{A}$ having a non-negligible advantage ϵ to distinguish between G^4 and G^5, then there exist an adversary $\mathcal{B}$ able to break the anonymity of Protego with a non-neglible advantage $\frac{\epsilon}{4}$. Therefore, we have the following distinguishing advantage:

$$\left|\Pr\left[\mathsf{G}^4_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^5_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$$

Game G^6. In this game, we replace the triplet $(usk_1, upk_1, cred_1)$ being the Protego material associated with the user $\mathcal{U}_1$, with a freshly generated triplet $(usk'_1, upk'_1, cred'_1)$ being independent of $\mathcal{U}_1$. Again, since the indistinguishability between this game G^6 and the previous game G^5 holds under the same assumption following the same principle and hence following the same reduction method, we directly conclude that the distinguishing advantage between G^5 and game G^6 is upper-bounded by the following advantage:

$$\left|\Pr\left[\mathsf{G}^5_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^6_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 4 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$$

In our final game G^6, the keys of both $\mathcal{U}_0$ and $\mathcal{U}_1$ are not used anymore. Hence, in our last game G^6, the advantage to distinguish between $\mathcal{U}_0$ and $\mathcal{U}_1$ during the challenge phase of the experiment $\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}$, defined by the probability $\Pr\left[\mathsf{G}^6_{\mathcal{A}}(1^\lambda) \to 1\right]$ equals $\frac{1}{2}$. Therefore, the probability to distinguish between G^0 and G^6 is upper-bounded by the

following advantage:

$$\begin{aligned}\left|\Pr\left[\mathsf{G}^5_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{G}^6_{\mathcal{A}}(1^\lambda) \to 1\right]\right| &= \left|\Pr\left[\mathsf{Exp}^{\mathsf{UNL}}_{\mathcal{A}}(1^\lambda) \to 1\right] - \frac{1}{2}\right| \\ &\leq 16 \cdot \mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}} + 8 \cdot \mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}\end{aligned}$$

And by hypothesis, the advantages $\mathsf{Adv}^{\mathsf{APUnl}}_{\mathcal{A}}$ and $\mathsf{Adv}^{\mathsf{ProAno}}_{\mathcal{A}}$ are negligible, meaning that Spotlight provides unlinkability.

□

4.5 Implementation of Applause and Spotlight

A proof-of-concept of Applause and Spotlight has been written in Rust and is available in [LMMOA24]. This implementation showcases the protocol's completeness and highlights the anticipated low computational cost of the cryptographic operations within our proposal. In our implementation, we have measured the impact of up to a thousand participants event purchasing a ticket. After the purchase, all acquired tickets are transferred to load the shared state while generating the new (transferred) tickets. Subsequently, the refund and validation processes are executed with these tickets.

Our analysis specifically focuses on cryptographic operations, excluding communication time and payment processes. Communication time is highly dependent on the network infrastructure, and our protocol allows for multiple anonymous payment methods to be integrated. These presented execution times should be considered as a baseline.

We rely on standard cryptographic primitives: We used the Curve25519 elliptic curve [Ber06] for ElGamal [ElG85b] and Schnorr signatures [Sch91]. To achieve key randomisation for Schnorr scheme, we rely on the generic algorithm generating a new key pair. Benchmarks over 200 iterations from 1 ticket in the database up to 1000 tickets in it have highlighted the constant time execution of all process. On an Ubuntu laptop equipped with an Intel i7-12800H processor and 32 GB of RAM, our Applause implementation gives an average execution times of 17 ms for a purchase, 68 ms for a transfer, 25 ms for a refund and 45 ms for a validation, hence emphasising the high efficiency of all cryptographic operations. On a hand, since Spotlight is constructed using the same Purchase, Refund and Transfer protocols compared to Applause, the same execution times are expected which has been empirically confirmed. On the other hand, since the ticket validation protocol Validate is equipped on an additional proof generation and verification using Protego, Validate in Spotlight is supposed to be slower, which is confirmed by our implementation of Spotlight achieving 165 ms for a single execution of Validate.

Hence, the overhead brought by securing Applause and Spotlight is acceptable. Moreover, these timings are within the same order of magnitude as 1 *Round-Trip Time*. The overhead introduced by securing the protocol appears acceptable at any step of the process. Finally, the measurement of the shared state update shows that it is negligible, always lower than 1 millisecond. Therefore, our instantiation is efficient and scalable.

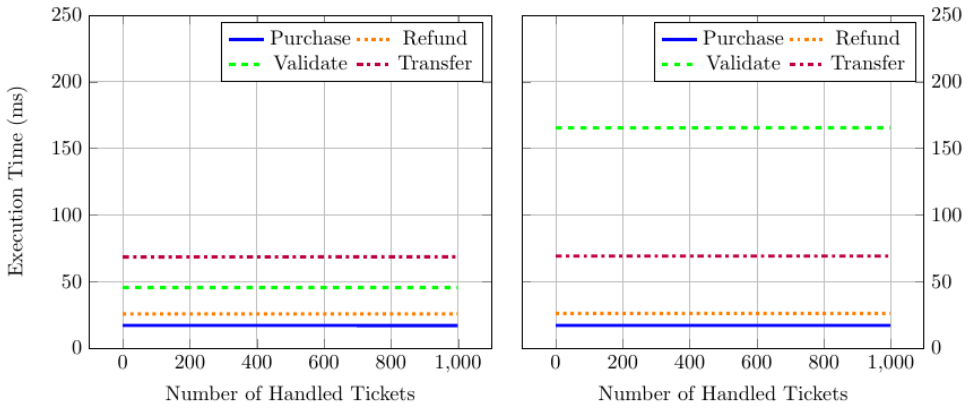


Figure 4.29: Mean execution times (in milliseconds) of the cryptographic operations involved in Applause (left) and Spotlight (right) over 200 iterations, depending on the number of handled tickets.

4.6 Conclusion

We presented Applause, a ticketing system that preserves the physical aspects of paper tickets while ensuring that any user can buy, refund, validate, or transfer a ticket. The system additionally safeguards the privacy of users. Applause achieves unforgeability, no-double-spending and privacy of the tickets. It also preserves anonymity of users, ensuring that the identity of users remains private but also that any interaction with the system cannot be linked to a previous interaction as coming from the same user. The anonymity ensured by Applause highly depends on the anonymity property of the used payment method that we chosen to include in a generic fashion. We have additionally extended Applause to obtain an auditable version called Spotlight, in which at the cost of a slighlty increased execution time overhead during the ticket validation (around 165 milliseconds instead of 45 milliseconds), an additional third-party called the judge can reveal the identity of all users attending an event, whereas other entities are unable to discern whether a user has purchased a ticket and for which event. Our proof-of-concept of Applause and Spotlight shows a great performance of our ticketing system, following the order of three-digits milliseconds for a complete execution of Applause and Spotlight.

Chapter 5

UC-Secure Distributed File Transfer

Attribute-based encryption (ABE) is a fine-grained access encryption scheme in which a user securely shares a message to a group of users once, every user of this group being able to recover the encrypted message while all other users out of this group do not. Briefly, the formal definition of the aforementioned "group" is realised by associating to each user an attribute x and by adding an access policy y to the ciphertext, the decryption procedure failing if the attribute x does not satisfy the access policy y. In ABE, the attribute x takes the form of a decryption key being computed and sent by a trusted authority holding a so-called master key pair. ABE has received lot of attention over the years to construct interesting primitives [AC17, HLWW23]. From a security standpoint, similarly to standard encryption schemes where indistinguishability holds only if the decryption key has not been corrupted, in ABE the indistinguishability for a ciphertext c holds only if the adversary does not have access to an attribute that can decrypt c. In game-based security, this is prevented by adding a winning condition preventing the adversary to decrypt the challenge ciphertext. Sadly, this mitigation cannot be transposed directly to the Universally Composable (UC) paradigm. Indeed in UC, traditional ideal functionalities for encryption aim to replace *all* plaintexts whose indistinguishability can be ensured by leakages. This way, the privacy of the message is direct in the sense that it is never revealed to the adversary. However, in ABE, several decryption keys are distributed among the users based on their attributes, even during the protocol execution. Observe that when the adversary has access to a decryption key associated to an attribute of its choice at any time, referred here as the *active* attribute corruption setting, a trivial attack to reveal if a ciphertext c encrypts a leakage or a real message arises: The adversary requires a decryption key whose associated attribute satifies the access policy of c, leading to the recovery of the underlying message. Hence, no security can be obtained with the standard ABE definition under active attribute corruption. This issue has already been noticed by [LW16] in a similar field of Role-Based Access Control (RBAC), where a user grants an access to some resources based on attributes. Security of ABE against active attribute corruption has already been achieved, for example by Camenisch *et al.* [CDEN12] using ABE equipped of an interactive decryption procedure between the user owning a ciphertext and a trusted third-party, owning decryption key for users. On one hand, security against active adversary is achieved, but on the other hand, protocols using standard ABE have to integrate a more complex ABE primitive to fit in UC. This replacement is not always desirable, for instance with protocols whose architecture requirements do not allow an additional third-party to support interactive decryption, and may prefer standard ABE, even at the cost of a restricted security setting. However, due to the need of additional features, we observe a natural gap where no ideal functionality for standard ABE has been provided yet.

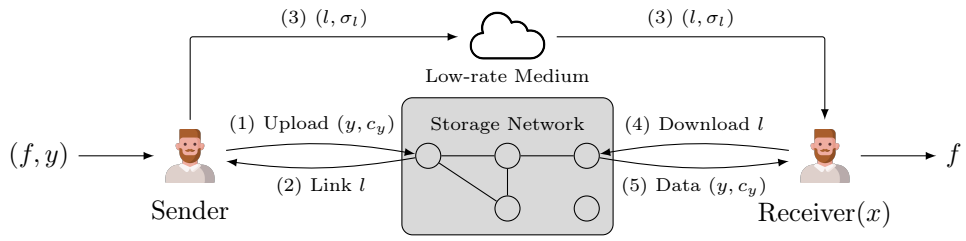


Figure 5.1: Representation of our system where a sender shares a file f to a receiver.

Summary of Contribution

Considering *standard* ABE, we propose the first ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ secure in the *static* attribute corruption setting. The protocol execution is divided into two distinct phases. The first phase, being viewed as a setup phase, consists for the adversary to instantiate any parties of its choice with the possibility to corrupt them, allowing it to recover any decryption key associated to an attribute of its choice. During the second phase, the adversary is still allowed to instantiate parties but corruption of parties asking for decryption keys is no longer accepted. Then, assuming this constraint and an IND-CCA2-secure ABE scheme [1], we prove that our real protocol $\mathcal{P}_{\mathsf{ABE}}$ securely realise our ideal functionality $\mathcal{F}_{\mathsf{ABE}}$. To increase usability, we have written $\mathcal{F}_{\mathsf{ABE}}$ and $\mathcal{P}_{\mathsf{ABE}}$ using the iUC framework, having the particularity to rely on the same formalism to express ideal, real and hybrid protocols. Based on the IITM model [KTR20a], this framework has been designed to be user-friendly, a welcomed feature to limit the complexity of reading and writing UC protocols.

To motivate the usability of our ideal functionality $\mathcal{F}_{\mathsf{ABE}}$, already strengthened by the easy-to-use iUC framework, we put it in practice to solve an apparently simple problem consisting of transferring a potentially large file between two or more users while ensuring integrity, authenticity, confidentiality and efficiency at the same time, which is particularly interesting for large-scaled companies. With this problem in mind, we have constructed a protocol allowing a user to share efficiently a potentially large document, let say, to all users working in a department. Attribute-based encryption is naturally designed to be used in this setting to provide confidentiality of the transferred files and hence is part of the protocol. To ensure authenticity and integrity of the transferred file in our file transfer protocol, our protocol also relies on digital signature. In addition to all these security properties obtained by the combination of attribute-based encryption and digital signature, we dedicate our system to be particularly efficient in the case of large transferred files. To reach this efficiency, our construction has the particularity to be constructed atop of a *distributed storage network*, a system composed of many servers whose general behaviour is similar to a graph. A neat feature, compared to the single-server setting, is that it faces communication delay and workload issues. Inter-Planetary File System (IPFS) is a hash-based distributed storage system in which a server maintains a list of link-file pair (l, f) where l is the link of the file f, computed with a cryptographic hash function h as $l \leftarrow h(f)$. Later on, given the link, the server

[1] An IND-CCA2-secure scheme can be efficiently derived from any IND-CPA-secure scheme via the Fujisaki-Okamoto transform [FO13].

easily recovers and returns the file. We give an overview of our protocol in Figure 5.1, acting between a sender and a receiver. The sender obtains as an input a (potentially large) file f as well as some access policy y, and sends the file to every receiver having an attribute x satisfying y. As depicted in Figure 5.1, during step (1), the sender computes an encryption of f denoted c_y using attribute-based encryption where y is the access policy, and sends the couple (y, c_y) on a storage server of its choice. At step (2), the storage server responds with a link l computed as the hash of (y, c_y). At step (3), the sender computes σ_l the signature of l and sends the tuple (l, σ_l) through a limited communication medium, restricted to transmit data having length independent of the message. When the receiver obtains the link l and the associated signature σ_l, it obtains a proof of integrity and authenticates the sender simultaneously. At step (4) and (5), the receiver downloads the couple (y, c_y) using the link l, decrypts the result and checks authenticity of l using σ_l. This protocol is proved to securely realise our Authenticated Attribute-based File Transfer ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$, but also to be highly-practical, confirmed by our proof-of-concept fully-written in Rust and available at [DAF24], sending up to 450 megabytes of data in 474 milliseconds.

Related Work

As explained above, the state-of-the-art for ABE in UC already proposes ideal functionality, but is always equipped with an additional property or having a different design to guarantee security against active adversary. Abe and Ambrona [AA22] introduced an ideal functionality for ABE where the key generation is replaced by a blind key generation procedure including a non-interactive zero-knowledge proof. To obtain active security, Camenisch *et al.* [CDEN12] propose to rely on a trusted third party owning the decryption key of users. To decrypt a ciphertext, the ABE protocol is equipped of an interactive decryption procedure with the third party, which is not able to identify which ciphertext is being decrypted. These two works need an ABE scheme having either a blind key generation or interactive decryption procedure, excluding every standard efficient attribute-based encryption scheme such as the Agrawal and Chase scheme [AC17]. To the best of our knowledge, there is no ideal functionality tailored for standard ABE.

We stress that our hybrid protocol $\mathcal{P}_{\mathsf{AAFT}}$, putting into application our ideal functionality $\mathcal{F}_{\mathsf{ABE}}$, constitutes a novel improvement in distributed file-transfer literature. Introduced by Garay *et al.* [GGJR98], distributed file transfer is a highly practical problem consisting of transmitting a data from a user to other users using several intermediate servers used for the data transmission. This problem is related to decentralised social media in which a user publishes content for users with whom it has relations, without relying on a centralized social medial owner. In both settings, the privacy of transmitted data is crucial [CMÖ11, CMÖ12] but the social medial setting brings context-oriented features that are not included in the distributed file transfer setting. For instance, the Safebook protocol [CMÖ11] allows a user to establish a trust relation with other users. This trust relation feature, inherent to social medial, is difficult to represent in a distributed file transfer system designed for a multi-site company.

In [GGJR98], Garay *et al.* proposes a system based on a verifiable secret sharing, ensuring both confidentiality by distributing shares among the storage servers, as done

in the more recent system called SAFE [BDF+20] with critical security and performance improvements. Due to the nature of secret sharing, confidentiality is only ensured with several storage servers and need to reconstruct the file. In contrast, by the confidentiality ensured by ABE, our system is still secure even with a (possibly corrupted) single storage server. Role-Based Access Control (RBAC) systems, a closely related topic, allows (or restricts) users to access some resources based on owned attribute. When a user accesses some content, it has first to be authenticated by an access-granting server. This is the case for the SESAME protocol [VGV97], a RBAC based itself on Kerberos. The work of Freudenthal *et al.* [FPP+02], proposes to check the access permission of users with multiple trust authorities (*e.g.,* a public key infrastructure). The role-based access for distributed storage system is presented in [HKN05]. They proposed a fix for the Object Store Devices specification [Web04] where unrestricted delegation is possible, in which confidentiality cannot be ensured. The proposed solution elegantly modifies the original protocol by adding secure channels and signature, to enforce role-based access to the files, without modifying the specification. All of these papers differ from our contribution by the introduction of authorities in charge of granting an access to some content. Our work requires trusted authorities to handle public keys and to provide decryption keys, but are involved only during the initialisation, no authorisation is required hereinafter. Introduced by Rizwan Ashgar *et al.* [AIRC13], ESPOON is a protocol working as RBAC but in outsourced environment with untrusted entities. Integrity is not ensured, whereas our work ensures the integrity and confidentiality of data in addition to sender authentication. The work of [FFW13] proposes a solution between RBAC and a storage system. They formally defined a new security definition of RBAC, in the spirit of encryption indistinguishability. The adversary is asked to guess an encrypted message, and is assumed to have a full-control on a file system where the encrypted message is stored. It can also corrupt any user of its choice. To achieve the proposed definition, a new protocol is introduced relying on attribute-based encryption, as done hereinafter. Our work adds more features: We ensure data integrity and data authentication (thanks to hash-based distributed storage and signatures, respectively) in addition to data confidentiality brought by the attribute-based encryption. Universal Composability (UC) has already been applied on the RBAC, initiated by Halevi *et al.* [HKN05] proposing a UC model which requires at every communication a secure channel between the two parties, even if the entity is corrupted (but in this case, the secret key might be leaked), a standard assumption in UC. In our work, we have chosen to avoid any particular property on communication channels for two reasons. First, secrecy is not always possible for example with anonymous protocols, or even desirable for example when transferred data can be read in clear by the adversary. Second, authenticity is traditionally achieved using digital signature, that can also be used to sign messages in other protocols. Introducing an ideal functionality for digital signature is hence more appropriate. Even if it does not constitute an issue, their UC model is built on the original UC model of Canetti [Can01] in which session identifier prevents communication between sessions. In comparison, our protocol is proven under the iUC framework, where every entity is allowed to communicate with the others without restriction, a useful property for example with signature ideal functionality, whose signing key are used in practice across multiple protocols.

Chapter Organisation

In Section 5.1, we briefly introduce all the necessary notions and terminology to understand our modelisation. In Section 5.2, we present our ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ and real protocol $\mathcal{P}_{\mathsf{ABE}}$ along the proof of realisation. In Section 5.3, we present the application of $\mathcal{F}_{\mathsf{ABE}}$ on the authenticated attribute-based file transfer with our ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ along our hybrid protocol $\mathcal{P}_{\mathsf{AAFT}}$ and our proof-of-concept.

5.1 On the iUC Framework

We provide an overview of the iUC framework. We refer readers interested by the iUC framework to the original paper [CKKR19]. A party pid involved in a protocol is traditionally equipped with session identifier sid, and acts in the protocol following a code specification called a role, and denoted role. The combination of the party identifier, the session identifier and the role, constitute the triplet $(\mathsf{pid}, \mathsf{sid}, \mathsf{role})$ and is called an *entity*. The existing role is specific to the designed protocol; for example a signature protocol consists of a role `signer` to sign messages and a role `verifier` to verify signed messages. The notion of entity is at the heart of the iUC framework, sharing similarities with object-oriented programming. In iUC, a *machine* denoted M_{role} implementing a role role corresponds to a class, both equipped with internal state used to store data. In a real protocol, a machine manages a single entity *i.e.,* represents a single party running in a single protocol execution. Yet, notice that a machine can be naturally extended to manage arbitrary number of entities, having different roles as well, the internal state being now used to share data across entities. For example, a signature ideal functionality benefits from this feature by adding authenticated messages in the internal state. In iUC, a machine, just like a class, can be instantiated several times, an instantiation being called an *instance*. Two important observations are to be made: First, the notion of entities and machines is sufficient to handle both real and ideal protocols. Second, a machine is not required to only handle entities sharing the same sid but any entities, a particularly useful property to handle cross-protocols party such as certificate authority.

The iUC framework provides algorithms to describe behaviour of instances *e.g.,* the number of accepted entities, the corruption model, the instance and entity initialisation, and more. When an entity with identity $(\mathsf{pid}, \mathsf{sid}, \mathsf{role})$ is contacted for the first time, the identity is submitted to every instance implementing the role role, until one instance accepts the entity, decided by the CheckID algorithm. If the instance does not have any accepted entity yet, then she executes the Initialization algorithm to initialise its internal state shared among all the accepted entities. When omitted, the CheckID accepts only a single entity, meaning that an instance a single entity. A similar initialisation function EntityInitialization is proposed to handle the initialisation for a single entity. Once initialised, an entity executes the Main block containing the code to be executed. The framework proposes several corruption types. We limit our introduction to the used corruption types *i.e.,* the *uncorruptible* type where no entity of the instance can be corrupted, and *static corruption* type where an entity can be corrupted only after the entity Initialization, and before to perform any action. We say that an entity "is determining its initial corruption status" when entity runs, after the initialisation and before

the main function, the DetermineCorrStatus function deciding if the entity is considered corrupted at the begins of the protocol. Once corrupted, the LeakedData specified what information has to be returned once corrupted, whereas AllowAdvMessage may prevents the adversary to send message to subroutines (of the same party), especially useful to model trusted execution environment. One may also prevent any corruption, for example for random oracle or certificate authorities, by programming the AllowCorruption algorithm to always return `false`. The code defined in Main is executed by *uncorrupted* entities. Each role is associated with either a *public* or a *private* visibility. A public role is accessible to the environment, whereas a private role is limited to entities inside the protocol. An entity accepts requests coming from the environment via the input-output interface `I/O`, but also requests coming from the adversary via the network interface `NET`, possibly modelling interactions of an ideal functionality with the simulator. When needed, an entity may also accept requests from more specific entities.

The iUC framework supports the notion of *responsive environment*, restricting the adversary to respond to a given request, before performing any interaction with the protocol. This feature prevents many attacks that are only conceptual. For example, suppose a signature ideal functionality $\mathcal{F}_{\mathsf{sig}}$, performing an initialisation request to the adversary to obtain signature and verification algorithms. Without response from the adversary, the ideal functionality is not initialised and hence the algorithms remains undefined, causing errors that the protocol designer has to care about. We stress that such errors are only conceptual and does not lead to practical attack on the protocol. Now, let perform the same initialisation request *responsively*. Then, the adversary is forced to respond as expected, preventing executions of undefined algorithms.

The current running entity $(\mathsf{pid}_{\mathsf{cur}}, \mathsf{sid}_{\mathsf{cur}}, \mathsf{role}_{\mathsf{cur}})$ is denoted $\mathsf{entity}_{\mathsf{cur}}$. A higher-protocol calling entity $(\mathsf{pid}_{\mathsf{call}}, \mathsf{sid}_{\mathsf{call}}, \mathsf{role}_{\mathsf{call}})$ is denoted $\mathsf{entity}_{\mathsf{call}}$. An instance has access to the set of managed entities that has been corrupted, denoted by CorruptionSet. An entity has to be considered if a subroutine has been corrupted, the corruption of an entity being verified by the corr_i algorithm returning `true` if the entity provided as an input has been corrupted, `false` otherwise. By $\mathsf{alg}^{(p)}$ we denote the execution of an algorithm alg whose execution time is bounded by the polynomial p.

Concurrent Composition Theorem. The composition theorem in the iUC framework needs a notion of polynomial run-time, that has been shown to be non-trivial in UC [HMU05]. Compared to other UC frameworks, the iUC framework, based on the IITM framework [KTR20a], provides a simpler notion of polynomial run-time execution. Let $\mathcal{P}$ be a protocol, a be the input and λ be the security parameter. We said that $\mathcal{P}$ is *environmentally bounded* if for every environment $\mathcal{E}$, the protocol $\mathcal{E}|\mathcal{P}$ has an execution time bounded by the polynomial $p(\lambda, |a|)$. Similarly, an environment $\mathcal{E}$ is *universally bounded* if for every protocol $\mathcal{P}$, the run-time execution of the system $\mathcal{E}|\mathcal{P}$ is bounded by $p(\lambda, |a|)$. We recall in Theorem 14 the concurrent composition theorem of the iUC framework. We recall the few notions required by the theorem: Let $\mathcal{P}$ and $\mathcal{Q}$ be two protocols. We said that $\mathcal{P}$ is *complete* if every subroutine role is part of $\mathcal{P}$. We denote by $\mathrm{Comb}(\mathcal{Q}, \mathcal{P})$ the set of all protocols obtained by combining $\mathcal{Q}$ and $\mathcal{P}$.

Theorem 14 (Concurrent Composition Theorem [CKKR19])**.** Let $\mathcal{P}$ and $\mathcal{F}$ be two protocols such that $\mathcal{P} \leq \mathcal{F}$. Let $\mathcal{Q}$ be another protocol such that $\mathcal{Q}$ and $\mathcal{F}$ are connectable. Let $\mathcal{R} \in \text{Comb}(\mathcal{Q}, \mathcal{P})$ and let $\mathcal{I} \in \text{Comb}(\mathcal{Q}, \mathcal{F})$ such that $\mathcal{R}$ and $\mathcal{I}$ have the same sets of public roles. If $\mathcal{R}$ is environmentally bounded and complete, then $\mathcal{R} \leq \mathcal{I}$.

5.2 Standard Attribute-based Encryption Realisation

Attribute-based Encryption allows to broadcast a message to all users, where only users having the read access with respect to an *access policy* associated to the message are able to read the message. We say that a user has a read access when it is associated to some *attribute*, say x, respecting the policy of the message, say y. This statement is represented by $x \in y$. Let recall the ABE definition introduced in Definition 12 at page 29. A standard ABE scheme is defined by four algorithms, namely $\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}$ and Dec. The Setup algorithm takes as an input the unary representation of the security parameter λ and it outputs a master key pair (msk, mpk). The encryption algorithm Enc expects as an input the master public key mpk, the access policy y and a message m, and it outputs the ciphertext c_y. To decrypt a message, one may previously ask to the authority owning the master key pair a decryption key denoted sk_x associated to some attribute x. This decryption key generation is handled by the KeyGen algorithm taking as input the master secret key msk as well as the attribute x and outputs the decryption key sk_x. This decryption key sk_x along a ciphertext c_x are provided to the decryption algorithm Dec, returning either the underlying plaintext m if and only if $x \in y$, or $\perp$ otherwise. An ABE scheme is said *correct* if for every master key pair $(msk, mpk) \leftarrow \mathsf{Setup}(1^\lambda)$, every ciphertext $c_y \leftarrow \mathsf{Enc}(mpk, y, m)$ for any message m and access policy y, every decryption key $sk_x \leftarrow \mathsf{KeyGen}(msk, x)$ for any attribute x with $x \in y$, we have $\Pr[\mathsf{Dec}(sk_x, c_y) = m] = 1 - \epsilon$ for some negligible probability ϵ. In this work, we require an IND-CCA2-secure ABE which informally states that it must be infeasible to tell if a ciphertext c_y encrypts either the message m_0 or m_1 as long as no corrupted user has a secret key sk_x allowing to decrypt c_y. The security experiment is presented in Figure 2.3 at page 30.

Standard Attribute-Based Encryption Ideal Functionality

The ideal functionality $\mathcal{F}_{\mathsf{ABE}}$, presented in Figure 5.2, proposes an instance managing several encryptors and decryptors as well as a *single* setup entity designated by the `setup` role. This setup entity, as its name suggests, deals with the setup algorithm and hence owns the master key pair, which has to remain private. Observe that using hierarchical session identifier property of iUC, an entity in $\mathcal{F}_{\mathsf{ABE}}$ is supposed to have a session identifier sid of the form $(\mathsf{pid}', \mathsf{sid}')$ with pid' the party identifier of the setup entity.

During the initialisation, the ideal functionality sends a responsive request via the `NET` interface to the simulator in order to obtain the master public key mpk as well as the encryption and decryption algorithms. In addition to this request, the ideal functionality parses the session identifier of the entity being accepted by the instance

Ideal functionality $\mathcal{F}_{\mathsf{ABE}} = (\texttt{setup}, \texttt{encryptor}, \texttt{decryptor})$:

Participating roles: `setup`, `encryptor`, `decryptor`
Corruption model: static corruption
Protocol parameters:

- A polynomial $p \in \mathbb{Z}[x]$ used to bound the runtime execution of provided algorithms.
- A deterministic length-preserving leakage function L used to compute leakages.
- A time $T \in \mathbb{N}$ delimiting phase in which decryption keys are provided, from the phase where encryption and decryption are operated. We denote by $t \in \mathbb{N}$ the current time.

$M_{\texttt{setup,encryptor,decryptor}}$:

Implemented role(s): `setup`, `encryptor`, `decryptor`
Internal state:

- $\mathsf{msgList} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ {*Set of encrypted messages*
- $\mathsf{keys} \subset (\{0,1\}^*)^3 \mapsto \{0,1\}^* \times \{0,1\}^*$
- $(mpk, \mathsf{Enc}, \mathsf{Dec}) \in (\{0,1\}^* \cup \bot)^3 = (\bot, \bot, \bot)$
- $\mathsf{pidsetup} \in \{0,1\}^* \cup \{\bot\} = \bot$
- $\mathsf{corrAttr} \subseteq \{0,1\}^* = \emptyset$

CheckID(pid, sid, role): Check that $\mathsf{sid} = (\mathsf{pid}', \mathsf{sid}')$, then accept every entity with the same SID, otherwise reject.
Corruption behavior:

- **AllowCorruption**(pid, sid, role): Returns `false` if role = `setup` or role = `decryptor` and $T < t$, otherwise returns `true`.

Initialization:

send responsively `InitABE` **to** `NET`
wait for $(\texttt{Init}, (mpk', Enc, Dec))$
$(mpk, \mathsf{Enc}, \mathsf{Dec}) \leftarrow mpk', Enc, Dec$
parse $\mathsf{sid}_{\mathsf{cur}}$ **as** (pid, sid)
$\mathsf{pidsetup} \leftarrow \mathsf{pid}$

Main:

recv $(\texttt{InitAttr}, x)$ **from** `I/O` **to** $(\_, \_, \texttt{decryptor})$ **s.t.** $\mathsf{keys}[\mathsf{entity}_{\mathsf{call}}] \neq \bot$:
  send responsively $(\texttt{InitReceiver}, x)$ **to** `NET`
  wait for $(\texttt{Init}, sk_x)$
  for $(m, y, c_y) \in \mathsf{msgList}$:
    $m' \leftarrow \mathsf{Dec}^{(p)}(sk_x, c_y)$
    if $(x \in y \wedge m' \neq L(\lambda, m)) \vee (x \notin y \wedge m' \neq \bot)$:
      send $(\texttt{Registered}, x, 0)$ **to** `NET` {*Decryption correctness failure*
  $\mathsf{keys}[\mathsf{entity}_{\mathsf{call}}] \leftarrow (x, sk_x)$
  send $(\texttt{Registered}, x, 1)$ **to** `NET`

recv $(\texttt{CorrAttr}, x)$ **from** `NET` **s.t.** $t \leq T$:
  add x **to** corrAttr

recv `PubKey?` **from** `_` **to** $(\mathsf{pidsetup}, \_, \texttt{setup})$:
  reply $(\texttt{PubKey}, mpk)$

recv $(\texttt{Encrypt}, y, m, mpk')$ **from** `I/O` **to** $(\_, \_, \texttt{encryptor})$ **s.t.** $T < t$:
  if $mpk \neq mpk' \vee \exists x \in \mathsf{corrAttr}$ **s.t.** $x \in y$:
    $c_y \leftarrow \mathsf{Enc}^{(p)}(mpk', y, m)$
    reply $(\texttt{Ciphertext}, c_y)$
  $m' \leftarrow L(\lambda, m)$
  $c_y \leftarrow \mathsf{Enc}^{(p)}(mpk', y, m')$
  for $(\_, (x, sk_x)) \in \mathsf{keys}$:
    if $(x \in y \wedge \mathsf{Dec}^{(p)}(sk_x, c_y) \neq m') \vee (x \notin y \wedge \mathsf{Dec}^{(p)}(sk_x, c_y) \neq \bot)$:
      reply $(\texttt{Ciphertext}, \bot)$ {*Encrytion correctness failure*
  add (m, y, c_y) **to** msgList
  reply $(\texttt{Ciphertext}, c_y)$

recv $(\texttt{Decrypt}, c_y)$ **from** `I/O` **to** $(\_, \_, \texttt{decryptor})$ **s.t.** $T < t \wedge \mathsf{keys}[\mathsf{entity}_{\mathsf{cur}}] \neq \bot$:
  $(x, sk_x) \leftarrow \mathsf{keys}[\mathsf{entity}_{\mathsf{cur}}]$
  if $\nexists(\_, \_, c_y) \in \mathsf{msgList}$: **reply** $(\texttt{Plaintext}, \mathsf{Dec}^{(p)}(sk_x, c_y))$
  if $\exists m, m'$ **s.t.** $(m, \_, c_y), (m', \_, c_y) \in \mathsf{msgList} \wedge m \neq m'$: **reply** $(\texttt{Plaintext}, \bot)$
  get(m, y, c_y) **from** msgList
  if $x \notin y$: **reply** $(\texttt{Plaintext}, \bot)$ {*Incompatible policy-access*
  reply $(\texttt{Plaintext}, m)$

Figure 5.2: Description of our ideal functionality $\mathcal{F}_{\mathsf{ABE}}$.

in order to recover the party identifier pidsetup of the setup entity. Note that the setup entity cannot be corrupted, thanks to the AllowCorruption definition.

Recall that when encrypting a message, the set of corrupted attributes is required to be static *i.e.,* the environment is not allowed to dynamically obtain decryption key, otherwise is able to trivial distinguish by looking for a ciphertext encrypting a leakage, so being impossible to prove secure as shown in [LW16]. Many scenarios are possible to obtain static corruption of attributes. We have chosen to divide the time in two distinct phases separated by a time $T \in \mathbb{N}$. The first phase happens when $t \leq T$ and must be seen as a setup phase, in which the environment is allowed to instantiate entities including encryptors and decryptors. Our ideal functionality allows static corruption meaning that the environment is allowed to corrupt an entity directly after its initialization only. The corruption of a decryptor allows the environment, on the behalf of the corrupted decryptor, to obtain a decryption key associated to the attribute of its choice. The corruption of an attribute is formalised in the ideal functionality using the `CorrAttr` request sent by the simulator via the `NET` interface. This notification sent by the simulator allows the ideal functionality to update the set of corrupted attributes corrAttr later used to distinguish ciphertext in which the ciphertext can be decrypted by the environment from ciphertext that does not. During the second phase, when $t > T$, the environment is no more allowed to corrupt a decryptor but also to obtain a decryption key using a previously corrupted decryptor.

During the second phase, the encryptor handles `Encrypt` requests used to encrypt a message, sent via the `I/O` interface. It expects as an input a message m to be encrypted, an access policy y as well as the master public key denoted mpk' that can be accessed by the environment via the `PublicKey?` instruction. The security of the ABE ideal functionality states that for a given valid master public key mpk, a message m and an access policy y, *if mpk is the valid master public key and if there is no corrupted attribute x such that $x \in y$*, then it must be infeasible to distinguish the real message m encrypted in the real protocol and the encryption of the leakage $L(\lambda, m)$ where L is the length-preserving deterministic leakage function. In other words, if the provided master public key is not the provided master public key or if the environment has a decryption key allowing to decrypt the ciphertext encrypting m, then the indistinguishability of the ciphertext cannot be guaranteed. This mentioned leakage function is given as a protocol parameter, and can be instantiated by a higher-protocol if required. It is clear that a winning adversary against the indistinguishability property of the ABE scheme can be used to construct a distinguisher against $\mathcal{F}_{\mathsf{ABE}}$ and a real ABE protocol. In case where the ciphertext encrypts a leakage, the ciphertext and the associated message are stored in the internal state of the instance, later used for the decryption.

Still during the second phase, entities having the `decryptor` role are allowed to receive decryption request from the environment via the `I/O` interface. The ciphertext decryption function expects as an input a ciphertext c_y. The procedure is only executed under two conditions: The function should be executed during the second phase and when the decryptor entity has been registered, meaning that the decryptor has a decryption key sk_x associated to some attribute x provided by the environment via the `InitAttr` request. Observe that this `InitAttr` function is not limited to be used only during the setup phase where $t \leq T$ but can be used at any time. Indeed, since the ideal

functionality supports static corruption and since the Main block of the ideal functionality is executed only by honest entities, then by construction a decryptor running the `InitAttr` function is necessary honest as well.

In case where the received ciphertext is stored in the internal state, along the associated plaintext then the plaintext is directly returned as a response. If the ciphertext is not stored in the internal state, then the ciphertext has been computed outside of the ideal functionality and hence no security can be proven. So we decrypt c_y using the provided decryption algorithm Dec and return the output as the plaintext response.

Standard Attribute-Based Encryption Protocol

Our attribute-based encryption protocol $\mathcal{P}_{\mathsf{ABE}}$, presented in Figure 5.3, is composed of three distinct instances, one used to manage a *single* setup, one used to manage a *single* encryptor and one used to manage a *single* decryptor. This stands in contrast with the ideal functionality whose single instance manages at the same time the setup party as well as all encryptors and decryptors related to this setup party (thanks to the entity acceptation mechanism rejecting entities having a distinct session identifier).

In the real protocol, the setup entity generates the master key pair (msk, mpk) and handles master public key access and decryption registration requests, returning respectively the master public key mpk and a freshly generated decryption key sk_x associated to an attribute x. Observe in Figure 5.3 that the registration requests are only accepted if the initiator of the request is a decryptor which makes sense since a decryption key is used only by a decryptor. Since a decryptor is allowed to have a single decryption key, we reject registration request if the decryptor having initiated the request has already obtained a decryption key. In the real protocol, the setup entity is made incorruptible thanks to the AllowCorruption specification.

An encryptor in the real protocol handles plaintext encryption request from the environment via the `I/O` interface, only during the second phase when $T < t$. It runs the encryption algorithm inputted with the master public key mpk, the plaintext m and the access policy y provided by the environment and returns the resulting ciphertext as a response. The decryptor in the real protocol handles attribute initialisation request `InitAttr`, obtaining as an input an attribute x that is sent to the setup entity for registration. If the key generation succeeds then it updates its internal state (only accessible by the current decryptor) with the received decryption key sk_x. Observe that this code is only executed by honest decryptors. Corrupted decryptors are allowed to send a registration request (and hence to get a decryption key) only during the first phase, thanks to the AllowAdvMessage blocking registration requests sent by the environment in the behalf of the corrupted decryptor during the second phase. In addition to the initialisation request, a decryptor handles the ciphertext decryption running the decryption algorithm on the previously obtained decryption key sk_x and the provided ciphertext c_y, before to return the obtained plaintext.

Theorem 15. Assuming the existence of a perfectly-correct and IND-CCA2-secure attribute-based encryption $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$, then $\mathcal{P}_{\mathsf{ABE}} \leq \mathcal{F}_{\mathsf{ABE}}$.

Protocol $\mathcal{P}_{\mathsf{ABE}} = (\texttt{setup}, \texttt{encryptor}, \texttt{decryptor})$:

Participating roles: `setup`, `encryptor`, `decryptor`
Corruption model: static corruption
Protocol parameters:
- An IND-CCA2 attribute-based encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$
- A time $T \in \mathbb{N}$ from which corruption and decryption keys are not allowed. We denote by $t \in \mathbb{N}$ the current time.

$M_{\texttt{setup}}$:

Implemented role(s): `setup`
Internal state:
- $mpk \in \{0,1\}^* \cup \{\bot\} = \bot$
- $msk \in \{0,1\}^* \cup \{\bot\} = \bot$
- $\mathsf{pidsetup} \in \{0,1\}^* \cup \{\bot\} = \bot$
- $\mathsf{keys} : (\{0,1\}^*)^3 \to \{0,1\}^*$

CheckID(pid, sid, role): Check that $\mathsf{sid} = (\mathsf{pid}', \mathsf{sid}')$. Accept a single entity.
Corruption behavior:
- **AllowCorruption**(pid, sid, role): return `false`

Initialization:
$(msk, mpk) \leftarrow \mathsf{Setup}(1^\lambda)$
parse $\mathsf{sid}_{\mathsf{cur}}$ **as** $(\mathsf{pid}, \mathsf{sid})$
$\mathsf{pidsetup} \leftarrow \mathsf{pid}$
Main:
recv `PubKey?` **from** _ **to** $(\mathsf{pidsetup}, \_, \texttt{setup})$:
reply $(\texttt{PubKey}, mpk)$

recv $(\texttt{Register}, x)$ **from** $(\_, \_, \texttt{decryptor})$ **to** $(\mathsf{pidsetup}, \_, \texttt{setup})$:
if $\mathsf{keys}[\mathsf{entity}_{\mathsf{call}}] \neq \bot$: **reply** $(\texttt{Registered}, \bot)$
$sk_x \leftarrow \mathsf{KeyGen}(msk, x)$
$\mathsf{keys}[\mathsf{entity}_{\mathsf{call}}] \leftarrow sk_x$
reply $(\texttt{Registered}, sk_x)$

$M_{\texttt{encryptor}}$:

Implemented role(s): `encryptor`
CheckID(pid, sid, role): Check that $\mathsf{sid} = (\mathsf{pid}', \mathsf{sid}')$. Accept a single entity.
Corruption behavior:
- **AllowAdvMessage**(pid, sid, role, $\mathsf{pid}_{\mathsf{recv}}$, $\mathsf{sid}_{\mathsf{recv}}$, $\mathsf{role}_{\mathsf{recv}}$, m): Check that ($\mathsf{pid} = \mathsf{pid}_{\mathsf{recv}}$). Otherwise, returns $\mathsf{role}_{\mathsf{recv}} \neq \texttt{setup}$ or m does not start with `Register`.

Main:
recv $(\texttt{Encrypt}, y, m, mpk)$ **from** `I/O` **s.t.** $T < t$:
$c_y \leftarrow \mathsf{Enc}(mpk, y, m)$
reply $(\texttt{Ciphertext}, c_y)$

$M_{\texttt{decryptor}}$:

Implemented role(s): `decryptor`
Internal state: $(x, sk_x) \in \{0,1\}^* \times \{0,1\}^* = (\bot, \bot)$ {*Decryption key*
CheckID(pid, sid, role): Check that $\mathsf{sid} = (\mathsf{pid}', \mathsf{sid}')$. Accept a single entity.
Corruption behavior:
- **AllowAdvMessage**(pid, sid, role, $\mathsf{pid}_{\mathsf{recv}}$, $\mathsf{sid}_{\mathsf{recv}}$, $\mathsf{role}_{\mathsf{recv}}$, m): If $\mathsf{role}_{\mathsf{recv}} = \texttt{setup}$ and m starts with `Register` and $T < t$, outputs `false`. Otherwise, outputs $\mathsf{pid} = \mathsf{pid}_{\mathsf{recv}}$.

Main:
recv $(\texttt{InitAttr}, x)$ **from** `I/O` **s.t.** $sk_x = \bot$:
parse $\mathsf{sid}_{\mathsf{cur}}$ **as** $(\mathsf{pid}, \mathsf{sid})$
send $(\texttt{Register}, x)$ **to** $(\mathsf{pid}, \mathsf{sid}_{\mathsf{cur}}, \texttt{setup})$
wait for $(\texttt{Registered}, sk'_x)$
if $sk'_x \neq \bot$: $sk_x \leftarrow sk'_x$

recv $(\texttt{Decrypt}, c_y)$ **from** `I/O` **s.t.** $T < t \wedge sk_x \neq \bot$:
$m \leftarrow \mathsf{Dec}(sk_x, c_y)$
reply $(\texttt{Plaintext}, m)$

Figure 5.3: Description of the protocol $\mathcal{P}_{\mathsf{ABE}}$.

Proof. Suppose a perfectly-correct IND-CCA2-secure attribute-based encryption Π. We start by giving the description of our simulator $\mathcal{S}$, used with our ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ in order to show that $\mathcal{P}_{\mathsf{ABE}} \leq \mathcal{F}_{\mathsf{ABE}}$. The simulator $\mathcal{S}$ starts the simulation by generating a new master key pair $(msk, mpk) \leftarrow \Pi.\mathsf{Setup}(1^\lambda)$, and it sends the Initialization request $(mpk, \Pi.\mathsf{Enc}, \Pi.\mathsf{Dec})$ to $\mathcal{F}_{\mathsf{ABE}}$. When a request of the form $(\texttt{InitReceiver}, x)$ is sent from $\mathcal{F}_{\mathsf{ABE}}$ to $\mathcal{S}$, a honest decryptor is initialised and hence the simulator generates the decryption key $sk_x \leftarrow \Pi.\mathsf{KeyGen}(msk, x)$ and responds with sk_x. A notification of the form $(\texttt{Registered}, x, b)$ is then received from the ideal functionality. If the bit b equals 1, then $\mathcal{S}$ registers than this decryptor has claimed the attribute x, otherwise it ignores the notification. Recall that we under static corruption, the adversary is allowed to corrupt only an entity directly after its initialisation and only at this point of the entity lifetime. In details, the static corruption is initiated by the entity, asking to the environment $\mathcal{E}$ its initial corruption status. We now specify the behaviour of our simulator acting differently depending on the current time $t \in \mathbb{N}$ with respect to the time $T \in \mathbb{N}$:

- Case $t \leq T$: In case where the adversary corrupts (directly after the initialisation) a decryptor and asks in the decryptor's name to obtain a decryption key to the `setup` entity using a request of the form $(\texttt{Register}, x)$ in the simulated real protocol, then the simulator executes honestly the decryption key generation code, but also notifies the ideal functionality of the corruption of x with the `CorrAttr` request.
- Case $T < t$: By construction of our real protocol, every `Register` request is blocked and hence no more decryption key is provided to the environment.

Since we assume static corruption, every corruption request sent by the environment to initialise entities is blocked by the simulator, preventing dynamic corruption (under which no security can be proven). To be more clear, we suppose without loss of security that $\mathcal{E}$ always use the valid master public encryption key to an `encryptor`, since it does not provide any advantage for $\mathcal{E}$ to break the security of Π.

Game 0. this game corresponds to the execution of the ideal protocol $\mathcal{S}|\mathcal{F}_{\mathsf{ABE}}$ with the environment $\mathcal{E}$.

Game 1. In this game, we replace $\mathcal{S}|\mathcal{F}_{\mathsf{ABE}}$ with $\mathcal{S}'|\mathcal{F}_{\mathsf{Fwd}}$ where $\mathcal{S}'$ simulating $\mathcal{S}|\mathcal{F}_{\mathsf{ABE}}$ and where $\mathcal{F}_{\mathsf{Fwd}}$ is a forwarding IITM, transferring every request from $\mathcal{S}'$ to $\mathcal{E}$ and $\mathcal{E}$ to $\mathcal{S}'$. Requests coming from the network interface of $\mathcal{S}'$ are directly transferred to $\mathcal{S}$. Since we do not have perform any modification, we have perfect indistinguishability: $\Pr\left[(\mathcal{E}|\mathcal{S}|\mathcal{F}_{\mathsf{ABE}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right]$. For more clarity, we index all the simulators by the index of the current game, hence $\mathcal{S}'$ is referred as $\mathcal{S}'_1$.

Game 2. Observe that by the perfect correctness of the attribute-based encryption, the correctness issues occurring during both the register procedure (ensuring valid generation of decryption keys) and during encryption cannot occurs. Hence in this game, we modify $\mathcal{S}'_1$ to construct $\mathcal{S}'_2$ in which we remove these correctness validation procedures in this game without impacting the view of $\mathcal{E}$. Hence, $\Pr\left[(\mathcal{E}|\mathcal{S}'_1|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}'_2|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right]$.

Game $i \in [\![3, n+3]\!]$. In this game, we focus on the i-th encryption request sent to the simulator $\mathcal{S}'_i$. In this game, we replace the encryption of the leakage $L(\lambda, m_i)$ by the encryption of the message m_i. Suppose that $\mathcal{E}$ is able to distinguish with a non-negligible probability between $(\mathcal{E}|\mathcal{S}'_i|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda)$ and $(\mathcal{E}|\mathcal{S}'_{i-1}|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda)$. We construct an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the security of Π, simulating $\mathcal{E}$ and whose role is

create a perfect simulation of $\mathcal{S}'_i|\mathcal{F}_{\mathsf{Fwd}}$ without having access to the master secret key msk owned by the challenger of the IND-CCA2 game. In details, $\mathcal{A}_1$ is running *before* obtaining the challenge ciphertext, and $\mathcal{A}_2$ continues the run of the simulation of $\mathcal{E}$ *after* that the challenge ciphertext was obtained. The adversary $\mathcal{A}_1$ obtains as the input the encryption key mpk, and has access to the key generation oracle $\mathsf{OKeyGen}$ and the decryption oracle ODec. The adversary $\mathcal{A}_1$ works as follows:

- When receiving a `PublicKey?` request, it responds $(\texttt{PublicKey}, mpk)$.
- When receiving a `InitAttr` request, it simulates a `Register` request with the same provided parameters, described below.
- When $\mathcal{E}$ sends a key generation request of the form $(\texttt{Register}, x)$ from a `decryptor` entity: If there is no record $(\mathsf{entity}, \_, \_)$ yet, then it calls the $\mathsf{OKeyGen}$ oracle to generate a secret decryption key sk_x associated to the provided attribute x, it registers $(\mathsf{entity}, sk_x, x)$ and returns sk_x. Otherwise, it returns an error.
- When $\mathcal{E}$ sends a request of the form $(\texttt{Encrypt}, mpk, y_j, m_j)$ for some $j < i$, then $\mathcal{A}_1$ computes $c_j \leftarrow \mathsf{Enc}(mpk, y_j, m_j)$ and responds with $(\texttt{Ciphertext}, c_j)$. When $j = i$, then $\mathcal{A}_1$ computes the leakage $\bar{m}_j \leftarrow L(\lambda, m_j)$ where L is a length-preserving leakage function, and encodes its all internal state in the st variable, and sends to the challenger the challenge response $((y_j, m_j, \bar{m}_j), \mathsf{st})$.
- When $\mathcal{E}$ sends a decryption request of the form $(\texttt{Dec}, c_j)$ from entity for $j < i$, it checks that the entity has already asked for decryption key sk_x by checking if there is a record $(\mathsf{entity}, sk_x, x)$. If there is no match, aborts with $(\texttt{Plaintext}, \perp)$. Otherwise, computes $m_j \leftarrow \Pi.\mathsf{Dec}(sk_x, c_j)$ and returns $(\texttt{Plaintext}, m_j)$.

We now describe our second adversary $\mathcal{A}_2$ taking as an input the challenge ciphertext c_i and the state st constructed by $\mathcal{A}_1$ used to continue the simulation of $\mathcal{E}$. The adversary $\mathcal{A}_2$ works as follows:

- The adversary $\mathcal{A}_2$ begins its simulation by sending the ciphertext c_i to $\mathcal{E}$, that is supposed to encrypt either m_i or $\bar{m}_i$. Observe that when m_i is encrypted, then $\mathcal{A}$ is simulating $(\mathcal{E}|\mathcal{S}'_i)$ or $(\mathcal{E}|\mathcal{S}'_{i-1})$ otherwise.
- The `InitAttr` and `Register` requests are the same as defined for $\mathcal{A}_1$.
- When $\mathcal{E}$ sends a request of the form $(\texttt{Encrypt}, mpk, y_j, m_j)$ for some $j > i$, then it computes $c_j \leftarrow \mathsf{Enc}(mpk, y_j, L(\lambda, m_j))$ and records (c_j, y, m) and finally responds with $(\texttt{Ciphertext}, c_j)$.
- When $\mathcal{E}$ sends a decryption request of the form $(\texttt{Dec}, c_j)$ from entity for $j > i$: If there is no record $(\mathsf{entity}, sk_x, x)$ or no record (c_j, y, m_j) then it responds with a failure. The case where there is several records for the same ciphertext is not considered since prevented by the attribute-based encryption scheme Π. If $x \in y$, then it responds with $(\texttt{Plaintext}, m)$, otherwise it responds with $(\texttt{Plaintext}, \perp)$.
- When $\mathcal{E}$ stops the simulation, $\mathcal{A}_2$ outputs 1 if $\mathcal{E}$ outputs 1. Otherwise, $\mathcal{A}_2$ outputs 0.

It is clear that our adversary $\mathcal{A}$ is polynomial-time. Since $\mathcal{E}$ is universally bounded, hence $\mathcal{A}$ constitutes a valid adversary for our IND-CCA2 experiment. Hence, we have:

$$\begin{aligned}\mathsf{Adv}^{\text{IND-CCA2}}_{\mathcal{A},\Pi} &= \left|\frac{1}{2}\cdot\Pr\left[b'=0|b=0\right]+\frac{1}{2}\cdot\Pr\left[b'=1|b=1\right]-\frac{1}{2}\right| \\ &= \left|\frac{1}{2}\cdot\left(\Pr\left[b'=0|b=0\right]-\Pr\left[b'=1|b=0\right]\right)\right| \\ &= \frac{1}{2}\cdot\left|\Pr\left[(\mathcal{E}|\mathcal{S}'_i|\mathcal{F}_{\mathsf{Fwd}})(\lambda)\to 1\right]-\Pr\left[(\mathcal{E}|\mathcal{S}'_{i-1}|\mathcal{F}_{\mathsf{Fwd}})(\lambda)\to 0\right]\right|\end{aligned}$$

Therefore, $\mathcal{E}|\mathcal{S}'_3|\mathcal{F}_{\mathsf{Fwd}}$ and and $\mathcal{E}|\mathcal{S}'_{n+3}|\mathcal{F}_{\mathsf{Fwd}}$ are indistinguishable:

$$\begin{aligned}&\left|\Pr\left[(\mathcal{E}|\mathcal{S}'_3|\mathcal{F}_{\mathsf{Fwd}})(\lambda)\to 1\right]-\Pr\left[(\mathcal{E}|\mathcal{S}'_{n+3}|\mathcal{F}_{\mathsf{Fwd}})(\lambda)\to 0\right]\right| \\ &\leq n\cdot\sum_{i=3}^{n+3}\left|\Pr\left[(\mathcal{E}|\mathcal{S}'_i|\mathcal{F}_{\mathsf{Fwd}})(\lambda)\to 1\right]-\Pr\left[(\mathcal{E}|\mathcal{S}'_{i-1}|\mathcal{F}_{\mathsf{Fwd}})(\lambda)\to 0\right]\right| \\ &\leq 2n\cdot\mathsf{Adv}^{\text{IND-CCA2}}_{\mathcal{A},\Pi}\end{aligned}$$

Game $n+4$. Observe that at this point, every ciphertext is encrypting the real message. Hence, instead of performing a plaintext recovery from the internal state of our simulator, our modified simulator $\mathcal{S}'_{n+4}$ ignores the ciphertext register and directly performs the decryption. As a consequence, we do not perform the attribute validation check $x \in y$, that we remove from the (simulated) ideal functionality $\mathcal{F}_{\mathsf{ABE}}$. By correctness of the attribute-based encryption, we have $\Pr\left[(\mathcal{E}|\mathcal{S}'_{n+3}|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda)\to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}'_{n+4}|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda)\to 1\right]$.

Game $n+5$. This game works exactly as the previous game except that we do not share the master public key in the simulated $\mathcal{F}_{\mathsf{ABE}}$. Instead, the simulated $\mathcal{F}_{\mathsf{ABE}}$ asks the simulator $\mathcal{S}'_{n+5}$ to obtain the master public key and is returned back to the environment. Since the master public key initially stored in the simulated $\mathcal{F}_{\mathsf{ABE}}$ is already the master public key generated by the simulator, then the view of $\mathcal{E}$ is not changed. Similarly, the encryption and the decryption procedures done in the simulated ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ are delegated to the simulators, forwarding for instance the request $(\texttt{Encrypt}, y, m, mpk)$ to the same encryptor in the simulated $\mathcal{P}_{\mathsf{ABE}}$. We follow the same approach for decryption requests. The response produced by the encryptor or the decryptor in the simulated protocol $\mathcal{P}_{\mathsf{ABE}}$ is returned back to the simulated ideal functionality $\mathcal{F}_{\mathsf{ABE}}$, forwarding the response to the environment $\mathcal{E}$. Finally, since the simulated $\mathcal{F}_{\mathsf{ABE}}$ does note use its internal state anymore, we remove it. Observe that all these modifications does not affect the view of $\mathcal{E}$ since the simulated ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ was not performing any check or internal state access. Hence, we have $\Pr\left[(\mathcal{E}|\mathcal{S}'_{n+4}|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda)\to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}'_{n+5}|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda)\to 1\right]$.

At this point, our simulator $\mathcal{S}'_{n+5}$ constitutes the most interesting part of the protocol, encrypting, decrypting and generating keys for entities, without performing any attribute validation (as done in our original $\mathcal{F}_{\mathsf{ABE}}$ *i.e.,* $x \in y$), and does not consider any internal state between the simulated entities. In other words, $\mathcal{S}'_{n+5}$ is our real protocol $\mathcal{P}_{\mathsf{ABE}}$. Even more, the simulated ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ is now limited to forward the machine. As result, our simulator $\mathcal{S}'_{n+5}$ is now connected to the environment via the intermediate of two forwards machines. By removing these two forward machines $\mathcal{F}_{\mathsf{Fwd}}$ from $\mathcal{S}'|\mathcal{F}_{\mathsf{Fwd}}|\mathcal{F}_{\mathsf{Fwd}}$ and connect all wires from the environment via the $\texttt{I/O}$ inter-

face directly to $\mathcal{S}'_{n+5}$, all these modifications being structural, we are ensured to have a perfect indistinguishability. Since all parties are simulated by $\mathcal{F}_{\mathsf{ABE}}$ and each party follows the instruction of the real protocol $\mathcal{P}_{\mathsf{ABE}}$ without having access to any shared register between entities, we have $(\mathcal{E}|\mathcal{S}'_{n+5}) = (\mathcal{E}|\mathcal{P}_{\mathsf{ABE}})$. By our sequence of games, we have shown that $(\mathsf{Enc}|\mathcal{S}|\mathcal{F}_{\mathsf{ABE}}) \equiv (\mathsf{Enc}|\mathcal{P}_{\mathsf{ABE}})$, thus $\mathcal{P}_{\mathsf{ABE}} \leq \mathcal{F}_{\mathsf{ABE}}$. Since the protocol is environmentally bounded and complete, then the Theorem 15 holds.

□

5.3 Authenticated Attribute-based File Transfer

5.3.1 Description of our Ideal Functionality $\mathcal{F}_{\mathsf{AAFT}}$

Our authenticated attribute-based file transfer ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ depicted in Figure 5.4, has been designed to allow a higher-protocol to easily rely on authenticated attribute-based file transfer. Each entity managed by the instance of $\mathcal{F}_{\mathsf{AAFT}}$ is associated with one of two following roles: A role `sender` representing an entity sending a file and a role `receiver` receiving a file. The ideal functionality maintains three distinct internal states attr, sentFiles and receivedFiles used respectively to remember inputted and corrupted attributes, to authenticate files sent by honest senders and finally to store valid received files, eventually shared with the environment via the `Collect` request.

A sender handles only `Send` requests coming from the `I/O` interface, expecting as a parameter the input data file f as well as the access policy y. A short tag r is uniformly sampled from $\{0,1\}^\lambda$ and stored along the file f and the access policy y into the sentFiles internal states, shared between all entities (sharing the same session identifier). This set consists of all authenticated files. To send the file, the ideal functionality shares the file to the simulator $\mathcal{S}$ using one of two manners depending on the corruption of attributes: If the environment has an attribute $x \in y$ then confidentiality of the file f cannot be ensured, hence shared with $\mathcal{S}$. On the other hand, if the environment does not have an attribute $x \in y$, the file is not shared with the simulator $\mathcal{S}$. It models confidentiality in the sense that it remains safely in the ideal functionality, following the standard encryption in UC such as [KTR20b] producing ciphertext encrypting a leakage instead of the real plaintext. In contrast with the `Send` request handling requests from the higher-protocol, the file reception modelled by the `Receive` request is received from the `NET` interface *i.e.,* from the simulator. This is motivated by the real-life mail system in which the server receives files from the network, awaiting the user to connect in order to collect messages. It is up to the simulator to correctly simulates the protocol and notifies the ideal functionality if a receiver receives a file. Observe that the code for `Receive` ensures, in case of honest sender, authentication and file access depending on the attribute x owned by the current receiver by checking if $x \in y$. If the sender is corrupted, then authentication and file access is delegated to the simulator.

Observe that the `Send` and `CorrAttr` functions are accessible only if the current time $t \in \mathbb{N}$ is strictly greater than a constant time $T \in \mathbb{N}$ defined as a parameter of the protocol, a crucial restriction to include the ABE ideal functionality in our hybrid protocol. This restriction leads us to separate the time in two distinct phases. During the first phase, we allow the environment to instantiate any entities but also to *statically*

Ideal functionality $\mathcal{F}_{\mathsf{AAFT}} = (\texttt{sender}, \texttt{receiver})$:

Participating roles: $\texttt{sender}, \texttt{receiver}$
Corruption model: static corruption
Protocol parameters:
- A time $T \in \mathbb{N}$ delimiting phase in which file sending is not accessible. We denote by $t \in \mathbb{N}$ the current time.

$M_{\texttt{sender},\texttt{receiver}}$:

CheckID(pid, sid, role): Accept all entities with the same SID.
Corruption behavior:
- **AllowCorruption**(pid, sid, role): Returns role $\neq$ `decryptor` or $t < T$.

Internal state:
- attr $\subseteq (\{0,1\}^*)^3 \times \{0,1\}^* \times \{0,1\} = \emptyset$
- sentFiles $\subseteq (\{0,1\}^*)^3 \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* = \emptyset$
- receivedFiles $\subseteq (\{0,1\}^*)^3 \times \{0,1\}^* = \emptyset$

Main:

recv $(\texttt{InitAttr}, x)$ **from** `I/O` **to** $(\_, \_, \texttt{receiver})$ **s.t.** $\nexists(\mathsf{entity}_{\mathsf{cur}}, \cdot, \cdot) \in \mathsf{attr}$:
  add $(\mathsf{entity}_{\mathsf{cur}}, x, 0)$ **to** attr
  send $(\texttt{Registered}, x)$ **to** `NET`

recv $(\texttt{CorrAttr}, \mathit{receiver}, x)$ **from** `NET` **s.t.** $t \leq T$:
  get $(\mathit{receiver}, x, b)$ **from** attr
  $b \leftarrow 1$

recv $(\texttt{Send}, f, y)$ **from** `I/O` **to** $(\_, \_, \texttt{sender})$ **s.t.** $T < t$:
  $r \xleftarrow{\$} \{0,1\}^\lambda$
  add $(\mathsf{entity}_{\mathsf{cur}}, y, r, f)$ **to** sentFiles
  if $\exists\ (\cdot, x, 1)$ **s.t.** $x \in y$:
    send $(\texttt{SendCorrupted}, y, r, f)$ **to** `NET`
  else:
    send $(\texttt{SendHonest}, y, r, |f|)$ **to** `NET`

recv $(\texttt{Receive}, y, r, \mathit{sender})$ **from** `NET` **to** $(\_, \_, \texttt{receiver})$:
  if $\nexists\ (\mathit{sender}, \cdot, r, \cdot) \in$ sentFiles:
    send responsively $(\texttt{WaitFile}, y, r, \mathit{sender})$ **to** `NET`
    wait for $(\texttt{ProvideFile}, b, f)$
    if $b = 1$:
      add $(\mathsf{entity}_{\mathsf{cur}}, f)$ **to** receivedFiles
  else:
    get $(\mathit{sender}, y, r, f)$ **from** sentFiles
    if $\mathsf{attr}[\mathsf{entity}_{\mathsf{cur}}] \in y$:
      add $(\mathsf{entity}_{\mathsf{cur}}, f)$

recv `Collect` **from** `I/O` **to** $(\_, \_, \texttt{receiver})$:
  reply $\mathcal{F} = \{f : (\mathsf{entity}_{\mathsf{cur}}, f) \in \text{receivedFiles}\}$

Figure 5.4: Description of our ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$.

corrupt any entity of its choice, including receivers and hence to obtain attributes. During the second phase, we prevent the environment to corrupt a receiver, and allow the environment to send a file.

5.3.2 Our Hybrid Protocol $\mathcal{P}_{\mathsf{AAFT}}$

Depicted in Figure 5.5, our protocol $\mathcal{P}_{\mathsf{AAFT}}$ is proved to realise our ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$. Our protocol relies at the same time on the real subroutine IPFS (being part of our contribution) but also on idealised subroutines including a certificate authority, the random oracle, digital signatures. Observe that following the UC terminology, since

our protocol $\mathcal{P}_{\mathsf{AAFT}}$ is built above real and idealised subroutines, $\mathcal{P}_{\mathsf{AAFT}}$ is said *hybrid*. We first present all these subroutines before introducing our hybrid protocol $\mathcal{P}_{\mathsf{AAFT}}$.

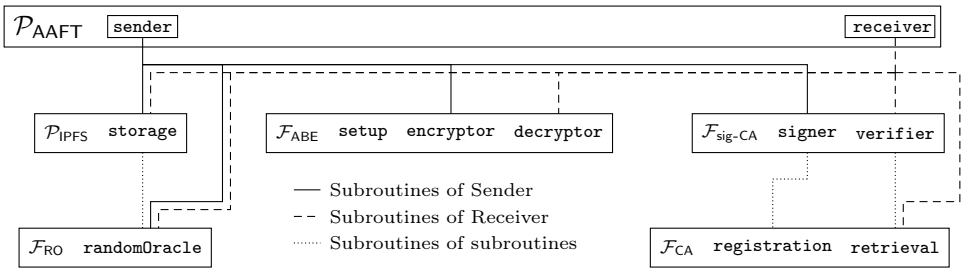


Figure 5.5: Graphical representation of our protocol $\mathcal{P}_{\mathsf{AAFT}}$. The random oracle ideal functionality $\mathcal{F}_{\mathsf{RO}}$ comes from [KTR20b], whereas the ideal functionalities for digital signature $\mathcal{F}_{\mathsf{sig\text{-}CA}}$ and certificate authority $\mathcal{F}_{\mathsf{CA}}$ comes from [CKKR19]. The link between roles A and B means that A depends on B, and must be read from top to bottom.

Certificate Authority Ideal Functionality

The Certificate Authority (CA) allows to register public keys and certifying that a given public key corresponds to some user. The modelisation of $\mathcal{F}_{\mathsf{CA}}$ presented in Figure 5.6 is taken from [CKKR19], consisting of two roles `registration` and `retrieval` . The `registration` role allows a calling entity to register a public key pk. The registration is modelled using the internal state of the (unique) instance associating the public key pk to the identification pair $(\mathsf{pid}, \mathsf{sid})$. Later, the public key is returned thanks the `retrieval` returning the public key pk associated to the provided identification pair $(\mathsf{pid}, \mathsf{sid})$. This ideal functionality enforces that the unique instance of $\mathcal{F}_{\mathsf{CA}}$ cannot be corrupted by the adversary which is standard hypothesis in numerous protocols.

Ideal functionality $\mathcal{F}_{\mathsf{CA}} = (\texttt{registration}, \texttt{retrieval})$:

Participating roles: `registration`, `retrieval`
Corruption model: incorruptible

$M_{\texttt{registration,retrieval}}$:

Implemented role(s): {`registration`, `retrieval`}
Internal state:
- $\mathsf{keys} : (\{0,1\}^*)^2 \to \{0,1\}^* \cup \{\bot\}$

CheckID(pid, sid, role): Accept all entities.
Main:

recv (`RegisterKey`, *key*) **from** `I/O` **to** (_, _, `registration`) :
  if $\mathsf{keys}[\mathsf{pid}_{\mathsf{call}}, \mathsf{sid}_{\mathsf{call}}] \neq \bot$:
    reply (`RegisteredKey`, `false`)
  else:
    $\mathsf{keys}[\mathsf{pid}_{\mathsf{call}}, \mathsf{sid}_{\mathsf{call}}] \leftarrow key$
    reply (`RegisteredKey`, `true`)

recv (`RetrieveKey`, (*pid*, *sid*)) **from** _ **to** (_, _, `retrieval`) :
  reply (`RetrievedKey`, $\mathsf{keys}[pid, sid]$)

Figure 5.6: Ideal functionality $\mathcal{F}_{\mathsf{CA}}$ [CKKR19].

Random Oracle Ideal Functionality

The ideal functionality $\mathcal{F}_{\mathsf{RO}}$ introduced in [KTR20b] and presented in Figure 5.7 exposes a single role denoted `randomOracle`. This ideal functionality handles `Hash` requests which given an arbitrary-sized bitstring m, associates a λ-sized random bitstring h. In particular, if m was never queried before, then $\mathcal{F}_{\mathsf{RO}}$ generates a random bitstring h, stores the couple (m, h) and returns h. Otherwise (m has already been queried), and thus $\mathcal{F}_{\mathsf{RO}}$ returns the h associated to m. Similarly to $\mathcal{F}_{\mathsf{CA}}$, we assume that the ideal functionality $\mathcal{F}_{\mathsf{RO}}$ manages all entities, meaning that there is a single instance of $\mathcal{F}_{\mathsf{RO}}$ in the protocol.

Ideal functionality $\mathcal{F}_{\mathsf{RO}}$ = (`randomOracle`):

Participating roles: `randomOracle`
Corruption model: incorruptible

$M_{\texttt{randomOracle}}$:

Implemented role(s): `randomOracle`
Internal state:
- $\mathsf{H} \subseteq \{0,1\}^* \times \{0,1\}^\lambda = \emptyset$

CheckID(pid, sid, role): Accept all entities.
Main:
recv $(\texttt{Hash}, m)$ **from** _ :
  if $\exists (m, h) \in \mathsf{H}$:
    reply $(\texttt{Hashed}, h)$
  else:
    $h \xleftarrow{\$} \{0,1\}^\lambda$
    add (m, h) **to** H
    reply $(\texttt{Hashed}, h)$

Figure 5.7: Ideal functionality $\mathcal{F}_{\mathsf{RO}}$ [KTR20b].

Inter-Planetary File System Protocol

An Inter-Planetary File System (IPFS) network consists of connected servers. Every server in the network maintains an internal state associating to a file f a link l where l is the hash of f computed using the cryptographic hash function. IPFS plays a central role in the efficiency of our construction by allowing a potentially large data to be transferred over a distributed storage network rather than to be sent between each user directly. We have modelled a single storage server in iUC as a real protocol denoted $\mathcal{P}_{\mathsf{IPFS}}$ and depicted in Figure 5.8. A storage server having the role `storage`, relies on the random oracle $\mathcal{F}_{\mathsf{RO}}$ used to hash files. It is equipped of the three following functions: `Upload` used to store files, `Links` returning all saved links, and `Download` which given a link l returns the file f associated with l. A storage server is not intended to provide more than the efficiency in our construction. Hence it can be dynamically corrupted without erasure *i.e.,* corruption occurs at any time, leaving the full control of the corrupted server to the adversary, all its internal state being leaked.

Digital Signature Ideal Functionality

The ideal functionality for digital signature $\mathcal{F}_{\mathsf{sig\text{-}CA}}$, introduced in [CKKR19] and recalled in Figure 5.9, is composed of the two roles `signer` and `verifier`, allowing respectively

Protocol $\mathcal{P}_{\mathsf{IPFS}} = (\texttt{storage})$:

Participating roles: `storage`
Corruption model: dynamic corruption without erasure

$M_{\texttt{storage}}$:

Subroutines: $\mathcal{F}_{\mathsf{RO}}$: `randomOracle`
Implemented role(s): `storage`
Internal state:
- files : $\{0,1\}^* \to \{0,1\}^* = \emptyset$ {*Stored files*

CheckID(pid, sid, role): Accept a single entity.
Main:
  recv $(\texttt{Upload}, f)$ **from** _ :
    send $(\texttt{Hash}, f)$ **to** $(\mathsf{pid}_{\mathsf{cur}}, \mathsf{sid}_{\mathsf{cur}}, \mathcal{F}_{\mathsf{RO}} : \texttt{randomOracle})$
    wait for $(\texttt{Hashed}, l)$
    $\mathsf{files}[l] \leftarrow f$
    reply $(\texttt{Uploaded}, l)$
  recv `Links` **from** _ :
    reply $\{l : \forall l \mapsto f \in \mathsf{files}\}$
  recv $(\texttt{Download}, l)$ **from** _ :
    reply $(\texttt{Downloaded}, \mathsf{files}[l])$

Figure 5.8: Protocol $\mathcal{P}_{\mathsf{IPFS}}$ for a storage server in the IPFS network.

to create a signature of a given message and to verify a signature. During the instance initialisation, the party identifier pid' of the signer is obtained from the session identifier having the form $\mathsf{sid} = (\mathsf{pid}', \mathsf{sid}')$. Before providing any signature, the signer expects from the higher protocol an initialisation request allowing the signer to register the verification key pk to the certificate authority, modelled via the $\mathcal{F}_{\mathsf{CA}}$ ideal functionality. After this initialisation step, the signer is allowed to sign any message. Note that before returning the signature, the signer stores the signed message m in a set of authenticated messages managed by the instance, used later for the signature verification. To verify a signature, an entity having the `verifier` role expects as an input a message m, a signature σ_m and a public verification key pk. The security of a digital signature is modelled by the ideal functionality by always rejecting every valid signature σ_m coming from an uncorrupted signer whose message m does not belong to the set of authenticated messages, and whose provided verification key pk is the valid one (*i.e.,* the public verification key provided by the adversary in the ideal functionality).

Hybrid Authenticated Attribute-based File Transfer Protocol

We are now ready to introduce our hybrid authenticated attribute-based file transfer protocol presented in Figure 5.10 and Figure 5.11 and denoted $\mathcal{P}_{\mathsf{AAFT}}$. It is based on the real protocol $\mathcal{P}_{\mathsf{IPFS}}$ modelling IPFS but also on the attribute-based encryption, digital signatures, certificate authority and random oracle ideal functionalities. The protocol is composed of the two roles `sender` and `receiver` modelling respectively the file sender and the file receiver.

Observe in Figure 5.10 and Figure 5.11 that our protocol defines the two protocol parameters being the time T used to separate the setup phase and the execution phase,

Ideal functionality $\mathcal{F}_{\text{sig-CA}} = (\texttt{signer}, \texttt{verifier})$:

Participating roles: `signer`, `verifier`
Corruption model: static corruption
Protocol parameters:
- A polynomial $p \in \mathbb{Z}[x]$ used to bound the runtime execution of provided algorithms.

$M_{\texttt{signer},\texttt{verifier}}$:

Implemented role(s): `signer`, `verifier`
Subroutines: $\mathcal{F}_{\text{CA}} : \texttt{registration}$
Internal state:
- $(\mathsf{Sign}, \mathsf{Verif}, pk, sk) \in (\{0,1\}^* \cup \{\bot\})^4 = (\bot, \bot, \bot, \bot)$
- $\mathsf{pidowner} \in \{0,1\}^* \cup \{\bot\} = \bot$
- $\mathsf{msgList} \subseteq \{0,1\}^* = \emptyset$
- $\mathsf{KeysGenerated} \in \{\mathsf{ready}, \bot\} = \bot$

CheckID(pid, sid, role): Check that sid has a $(\mathsf{pid}', \mathsf{sid}')$ format. If the check fails, return `false`, otherwise accept all entities with the same SID.

Corruption behavior:
- **LeakedData**(pid, sid, role): If called while $(\mathsf{pid}, \mathsf{sid}, \mathsf{role})$ determines its initial corruption status, use the default behavior of **LeakedData**. That is, output the initially received message and the sender of that message. Otherwise, if $\mathsf{role} = \texttt{signer}$ and $\mathsf{pid} = \mathsf{pidowner}$, return KeysGenerated. In all other cases, return $\bot$.
- **AllowAdvMessage**(pid, sid, role, $\mathsf{pid}_{\mathsf{recv}}$, $\mathsf{sid}_{\mathsf{recv}}$, $\mathsf{role}_{\mathsf{recv}}$, m): Check that $(\mathsf{pid} = \mathsf{pid}_{\mathsf{recv}})$. If $\mathsf{role}_{\mathsf{recv}} = \mathcal{F}_{\text{CA}} : \texttt{registration}$, also check that $\mathsf{role} = \texttt{signer}$ and $\mathsf{sid} = (\mathsf{pid}, \mathsf{sid}')$. If all checks succeed, output `true`, otherwise output `false`.

Initialization:
send responsively InitMe **to** `NET`
wait for $(\texttt{Init}, (Sign, Verif, pk, sk))$
$(\mathsf{Sign}, \mathsf{Verif}, pk, sk) \leftarrow (Sign, Verif, pk, sk)$
parse $\mathsf{sid}_{\mathsf{cur}}$ **as** $(\mathsf{pid}, \mathsf{sid})$
$\mathsf{pidowner} \leftarrow \mathsf{pid}$

Main:
recv `InitSign` **from** `I/O` **to** $(\mathsf{pidowner}, \_, \texttt{signer})$:
  send $(\texttt{RegisterKey}, pk)$ **to** $(\mathsf{pid}_{\mathsf{cur}}, \epsilon, \mathcal{F}_{\text{CA}} : \texttt{registration})$
  wait for $(\texttt{RegisterKey}, \_)$
  $\mathsf{KeysGenerated} \leftarrow \texttt{true}$
  reply $(\texttt{InitSign}, 1)$

recv $(\texttt{Sign}, m)$ **from** `I/O` **to** $(\mathsf{pidowner}, \_, \texttt{signer})$ **s.t.** $\mathsf{KeysGenerated} = \texttt{true}$:
  $\sigma_m \leftarrow \mathsf{Sign}^{(p)}(m, sk)$
  $b \leftarrow \mathsf{Verif}^{(p)}(m, \sigma_m, pk)$
  if $\sigma_m = \bot \vee b \neq 1$:
    reply $(\texttt{Signature}, \bot)$
  else:
    add m **to** msgList
    reply $(\texttt{Signature}, \sigma_m)$

recv $(\texttt{Verify}, m, \sigma_m, pk)$ **from** `I/O` **to** $(\_, \_, \texttt{verifier})$:
  $b \leftarrow \mathsf{Verif}^{(p)}(m, \sigma_m, pk)$
  if $pk = pk \wedge b = 1 \wedge m \notin \mathsf{msgList} \wedge (\mathsf{pidowner}, \mathsf{sid}_{\mathsf{cur}}, \texttt{signer}) \notin \mathsf{CorruptionSet}$:
    reply $(\texttt{VerResult}, \texttt{false})$ {*Prevents the signature forgeries*
  reply $(\texttt{VerResult}, b)$

Figure 5.9: Description of the ideal functionality $\mathcal{F}_{\text{sig-CA}}$ [CKKR19].

as well as the pidsetup parameter defining the party identifier of entity in charge of generating the master key pair of the ideal functionality.

Protocol $\mathcal{P}_{\mathsf{AAFT}} = (\texttt{sender}, \texttt{receiver})$:

Participating roles: `sender`, `receiver`
Subroutines: $\mathcal{F}_{\mathsf{ABE}}, \mathcal{F}_{\mathsf{sig\text{-}CA}}, \mathcal{F}_{\mathsf{CA}} : \texttt{retrieval}, \mathcal{F}_{\mathsf{RO}}, \mathcal{P}_{\mathsf{IPFS}}$
Corruption model: static corruption
Protocol parameters:

- The party identifier pidsetup, identifying the entity of the ABE scheme handling the master keys.
- A time $T \in \mathbb{N}$ delimiting phase in which decryption keys are provided, from the phase where encryption and decryption are operated. We denote by $t \in \mathbb{N}$ the current time.

$M_{\texttt{sender}}$:

Implemented role(s): `sender`
CheckID(pid, sid, role): Accept a single entity.
Internal state:

- $mpk \in \{0,1\}^* \cup \{\bot\} = \bot$

Corruption behavior:

- **DetermineCorrStatus**(pid, sid, role): return $\mathsf{corr}_i(\mathsf{pid}, (\mathsf{pid}, \epsilon), \texttt{signer})$ or $\mathsf{corr}_i(\mathsf{pid}, (\mathsf{pidsetup}, \mathsf{sid}), \texttt{encryptor})$.
- **AllowAdvMessage**(pid, sid, role, $\mathsf{pid}_{\mathsf{recv}}$, $\mathsf{sid}_{\mathsf{recv}}$, $\mathsf{role}_{\mathsf{recv}}$, m): Check that $(\mathsf{pid} = \mathsf{pid}_{\mathsf{recv}})$.

Initialization:
  send `PubKey?` **to** $(\mathsf{pidsetup}, (\mathsf{pidsetup}, \mathsf{sid}_{\mathsf{cur}}), \mathcal{F}_{\mathsf{ABE}} : \texttt{setup})$
  wait for mpk'
  $mpk \leftarrow mpk'$
  send `InitSign` **to** $(\mathsf{pid}_{\mathsf{cur}}, (\mathsf{pid}_{\mathsf{cur}}, \epsilon)), \mathcal{F}_{\mathsf{sig\text{-}CA}} : \texttt{signer})$
  wait for _
Main:
  recv $(\texttt{Send}, y, f)$ **from** `I/O` **s.t.** $T < t$:
    send responsively `Storage?` **to** `NET`
    wait for $(\texttt{Storage}, \mathit{storage})$
    parse $\mathit{storage}$ **as** $(\_, \_, \mathcal{P}_{\mathsf{IPFS}} : \texttt{storage})$
    send $(\texttt{Encrypt}, y, f, mpk)$ **to** $(\mathsf{pid}_{\mathsf{cur}}, \mathsf{sid}_{\mathsf{cur}}, \mathcal{F}_{\mathsf{ABE}} : \texttt{encryptor})$
    wait for $(\texttt{Ciphertext}, c_y)$
    send $(\texttt{Upload}, (y, c_y))$ **to** $\mathit{storage}$
    wait for _
    send $(\texttt{Hash}, (y, c_y))$ **to** $(\mathsf{pid}_{\mathsf{cur}}, \mathsf{sid}_{\mathsf{cur}}, \mathcal{F}_{\mathsf{RO}} : \texttt{randomOracle})$
    wait for $(\texttt{Hashed}, l)$
    send $(\texttt{Sign}, l)$ **to** $(\mathsf{pid}_{\mathsf{cur}}, (\mathsf{pid}_{\mathsf{cur}}, \epsilon), \mathcal{F}_{\mathsf{sig\text{-}CA}} : \texttt{signer})$
    wait for $(\texttt{Signature}, \sigma_l)$
    send $(\texttt{Sent}, l, \sigma_l)$ **to** `NET`

Figure 5.10: Description of our hybrid protocol $\mathcal{P}_{\mathsf{AAFT}}$ (Part 1).

In our protocol, an entity having the `sender` role is managed by a single instance, as specified by the CheckID algorithm. As we will see in a moment, a sender relies on the signature verification key. Within the iUC framework, the environment is allowed to corrupt any entity of its choice while the corruption model is respected. Thanks to the composability feature, an entity of an ideal functionality may have another entity coming from another ideal functionality as a subroutine, which might be corrupted by the environment. This subroutine corruption is particularly interesting to model protocols whose security must hold even in case where a (part of a) secret owned by a party is revealed, for instance with post-compromise security in secure messaging where the protocol regains security even after the leakage of a key [Rob22]. In $\mathcal{P}_{\mathsf{AAFT}}$, the

$M_{\texttt{receiver}}$:

Implemented role(s): `receiver`
Internal state:
- initiated $\in \{0,1\} = 0$
- files $\subseteq \{0,1\}^* = \emptyset$

CheckID(pid, sid, role): Check that sid = (pid′, sid′). Accept a single entity.
Corruption behavior:
- **AllowCorruption**(pid, sid, role): Returns $t < T$.
- **DetermineCorrStatus**(pid, sid, role): return $\mathsf{corr}_i(\mathsf{pid}, (\mathsf{pidsetup}, \mathsf{sid}), \texttt{decryptor})$.
- **AllowAdvMessage**(pid, sid, role, $\mathsf{pid}_{\mathsf{recv}}$, $\mathsf{sid}_{\mathsf{recv}}$, $\mathsf{role}_{\mathsf{recv}}$, m): Check that (pid = $\mathsf{pid}_{\mathsf{recv}}$).

Main:

recv $(\texttt{InitAttr}, x)$ **from** `I/O` **s.t.** initiated $= 0$:
  initiated $\leftarrow 1$
  send $(\texttt{InitAttr}, x)$ **to** ($\mathsf{pid}_{\mathsf{cur}}$, $\mathsf{sid}_{\mathsf{cur}}$, $\mathcal{F}_{\mathsf{ABE}}$: `decryptor`)

recv $(\texttt{Receive}, \mathit{sender}, \mathit{storage}, l, \sigma_l)$ **from** `NET` **s.t.** $T < t \wedge$ initiated $= 1$:
  parse *storage* **as** $(\_, \_, \mathcal{P}_{\mathsf{IPFS}} : \texttt{storage})$, *sender* **as** (pid, _, `sender`)
  send $(\texttt{Download}, l)$ **to** *storage*
  wait for $(\texttt{Downloaded}, (y, c_y))$
  send $(\texttt{RetreiveKey}, (\mathsf{pid}, \epsilon))$ **to** $(\mathsf{pid}_{\mathsf{cur}}, \mathsf{sid}_{\mathsf{cur}}, \mathcal{F}_{\mathsf{CA}} : \texttt{retrieval})$
  wait for $(\texttt{RetreivedKey}, pk)$
  send $(\texttt{Verify}, l, \sigma_l, pk)$ **to** $(\mathsf{pid}_{\mathsf{cur}}, (\mathsf{pid}, \epsilon), \mathcal{F}_{\mathsf{sig\text{-}CA}} : \texttt{verifier})$
  wait for $(\texttt{VerResult}, b)$
  send $(\texttt{Hash}, (y, c_y))$ **to** $(\mathsf{pid}_{\mathsf{cur}}, \mathsf{sid}_{\mathsf{cur}}, \texttt{randomOracle})$
  wait for $(\texttt{Hashed}, l')$
  send $(\texttt{Decrypt}, c_y)$ **to** $(\mathsf{pid}_{\mathsf{cur}}, (\mathsf{pidsetup}, \mathsf{sid}_{\mathsf{cur}}), \mathcal{F}_{\mathsf{ABE}} : \texttt{decryptor})$
  wait for $(\texttt{Plaintext}, f)$
  if $l = l' \wedge b = 1 \wedge f \neq \perp$: **add** f **to** files

recv `Collect` **from** `I/O` :
  reply files

Figure 5.11: Description of our hybrid protocol $\mathcal{P}_{\mathsf{AAFT}}$ (Part 2).

corruption of a party leads to the corruption of all its subroutines, and conversely. Hence, if the signature key of the sender is corrupted, the sender is corrupted as well. This corruption relation is modelled within the iUC framework using the DetermineCorrStatus. Observe that in addition to the corruption relation between the sender and the signature key, we define a corruption relation between the sender and the encryptor used by the sender to encrypt a file.

As an initialisation, the sender entity asks the attribute-based ideal functionality for the master public key used later to encrypt file. In addition to this master public key access request, the sender notifies the signature ideal functionality for initialisation, in which the instance of the digital signature ideal functionality obtains a signature key pair and registers the verification public key to the certificate authority $\mathcal{F}_{\mathsf{CA}}$.

The file sending, supported only during the second phase where $T < t$, obtains as an input a file f as well as an access policy y. The sender starts the file sending procedure with a responsive request to the environment to determine the storage server to use in the IPFS network. Then, the sender encrypts f using the ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ to obtain the ciphertext c_y. This ciphertext is hashed to obtain the link l and is sent to

the storage (chosen by environment $\mathcal{E}$). After having obtained the signature σ_l for l, the sender shares the link l along the signature σ_l with the environment $\mathcal{E}$.

The file reception is handled by an entity having the `receiver` role. Before accepting files, a receiver has to be initialised with an attribute x that is used to initialise the decryptor of the ABE ideal functionality $\mathcal{F}_{\mathsf{ABE}}$. In contrast with the `Send` function handling file sending request from the environment via the `I/O` interface, `Receive` requests are sent by the environment via the `NET` interface. Remark that the received files are not intended to be directly forwarded to the higher-protocol but to be collected using the dedicated `Collect` function. The file reception function receives as an input the sender and storage identities along a link l and a signature σ_l. The file reception downloads the ciphertext for the designated storage server in the IPFS network, retrieves the signature public verification key of the designated sender and asks the signature ideal functionality $\mathcal{F}_{\mathsf{sig\text{-}CA}}$ to verify the received signature σ_l. In addition, the file reception function obtains the hash l' of the previously received ciphertext c_y and compare the computed hash l with the link l' obtained as an input. Finally, the ciphertext c_y is decrypted recovering the file f. Under the condition that both links l and l' are identical, that the link verification succeeds and that the returned file f is valid, then f is stored in the set of received files *files* of the receiver.

Theorem 16. Assuming the real protocol $\mathcal{P}_{\mathsf{IPFS}}$ for the IPFS network, assuming the ideal functionalities for certificate authority $\mathcal{F}_{\mathsf{CA}}$, for digital signature $\mathcal{F}_{\mathsf{sig\text{-}CA}}$ and for the (standard) attribute-based encryption $\mathcal{F}_{\mathsf{ABE}}$, then we have:

$$(\texttt{sender}, \texttt{receiver} | \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{ABE}}, \mathcal{F}_{\mathsf{sig\text{-}CA}}, \mathcal{F}_{\mathsf{CA}}, \mathcal{P}_{\mathsf{IPFS}}) \leq \mathcal{F}_{\mathsf{AAFT}}$$

Proof. We start this proof by giving a description of our simulator $\mathcal{S}$, used with the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ to show that $\mathcal{P}_{\mathsf{AAFT}} \leq \mathcal{S} | \mathcal{F}_{\mathsf{AAFT}}$. In a nutshell, $\mathcal{S}$ runs a simulation of the real protocol and handles request coming from both the environment and the ideal functionality. During the simulation, without loss of security and functionality, we require the leakage function $L(\lambda, m)$ used by the ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ to return a zero-string $0^{|m|}$. We now provide the description of $\mathcal{S}$:

- When receiving a notification request of the form $(\texttt{Registered}, x)$ from the ideal functionality (simulating a receiver), attesting the access to some attribute x. In such case, inputs the same receiver in the simulated real protocol with the $(\texttt{InitAttr}, x)$ request.
- When receiving a corruption request from the environment to corrupt the entity entity defined by the triplet $(\mathsf{pid}, \mathsf{sid}, \mathsf{role})$. If the current time $t < T$, then it accepts the corruption request. Otherwise, ignore the corruption request. Each time a corruption request leads to the corruption either of a sender or a receiver, the simulator notifies the ideal functionality as well, leading to a synchronization of senders and receivers corruption between the simulated real protocol and the ideal functionality. In case where the entity in charge of handling the master secret keys receives a decryption key, assuming $t < T$, then the decryption key sk_x is computed and sent back to the corrupted decryptor. Observe that by construction, a corrupted decryptor equals a corrupted receiver. The simulator

$\mathcal{S}$ notifies the ideal functionality that the environment is allowed to decrypt any ciphertext associated with a policy access y whose $x \in y$ by sending the request $(\texttt{CorrAttr}, x)$ to $\mathcal{F}_{\mathsf{AAFT}}$.

- When receiving a request of the form $(\texttt{SendHonest}, y, r, |f|)$ or the request of the form $(\texttt{SendCorrupted}, y, r, f)$ from the initial functionality, the simulator inputs the simulated sender of $\mathcal{P}_{\mathsf{AAFT}}$ with the access policy y and the real file f in the case of `SendCorrupted`, or the zero-string $0^{|f|}$ of the case of `SendHonest`. In addition, in case of `SendHonest`, the simulator records the pair (r, c_y) where c_y is the ciphertext obtained via the simulated `encryptor` from the ideal functionality $\mathcal{F}_{\mathsf{ABE}}$. This record is used later to provide the random r to the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$.
- When receiving a request of the form $(\texttt{Receiver}, \mathit{sender}, \mathit{storage}, l, \sigma_l)$ from the environment $\mathcal{E}$ via the network interface, then it inputs the simulated real receiver, running all sanity checks including decryption, signature verification and link validation. Let c_y be the ciphertext obtained during the execution of the simulated receiver in $\mathcal{P}_{\mathsf{AAFT}}$. If all checks succeed, then if the simulator recovers the pair (r, c_y) and sends $(\texttt{Receive}, y, r, \mathit{sender})$ to the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$, and no response is expected later. Otherwise, there is no records (r, c_y) and hence it sends $(\texttt{Receive}, y, \perp, \mathit{sender})$ to the ideal functionality. By construction, the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ responds with a responsive request $(\texttt{WaitFile}, y, r, \mathit{sender})$ where r equals $\perp$. At this point, the ideal functionality expects a file f to register. Since there is no records by the simulator but the decryptor file f being authenticated, the simulator responds to this responsive request with $(\texttt{ProvideFile}, 1, f)$.

It is clear that our simulator $\mathcal{S}$ is polynomial-time. We are now ready to initiate our sequence of games, where our first game consists of the $(\mathcal{S}|\mathcal{F}_{\mathsf{AAFT}})$ ideal protocol, and our last game is our hybrid protocol $\mathcal{P}_{\mathsf{AAFT}}$:

Game 0. This game is the execution of the ideal protocol $(\mathcal{S}|\mathcal{F}_{\mathsf{AAFT}})$ with the environment $\mathcal{E}$, connected respectively to the `NET` interface of $\mathcal{S}$ and to the `I/O` interface of $\mathcal{F}_{\mathsf{AAFT}}$, with $\mathcal{S}$ being connected to the `NET` interface of $\mathcal{F}_{\mathsf{AAFT}}$.

Game 1. This game works as the previous game, except that we now execute the protocol $(\mathcal{S}'|\mathcal{F}_{\mathsf{Fwd}})$ where $\mathcal{S}'$ runs the simulated ideal protocol $(\mathcal{S}|\mathcal{F}_{\mathsf{AAFT}})$ and where $\mathcal{F}_{\mathsf{Fwd}}$ simply forwards every request from the environment $\mathcal{E}$ to $\mathcal{S}'$ and conversely. Since this modification is only structural without any modification on the ideal protocol behaviour, then we have a perfect indistinguishability between these two games. In the following, for a better clarity, we index each simulator $\mathcal{S}'$ with the index of the current game, hence: $\Pr\left[(\mathcal{E}|\mathcal{S}'_1|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}|\mathcal{F}_{\mathsf{AAFT}})(1^\lambda) \to 1\right]$.

Game 2. This game works as the previous game, except that we introduce a new simulator $\mathcal{S}'_2$ consisting of $\mathcal{S}'_1$ where we delegate the all attribution verification both during the `Send` and `Receive` requests to the simulator $\mathcal{S}$. First, let focus on the `Send` part of the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$, currently simulated by our simulator $\mathcal{S}'_2$. We rewrite the code of `Send` to remove the random r as well as the corrupted attribute condition, all of these lines being replaced only by a request of the form $(\texttt{Send}, y, f)$. The code still stores files being sent by an honest sender, but omits the random r *i.e.,* it records a tuple of the form (sender, y, f). The file reception request in the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$, handling requests of the form $(\texttt{Receive}, y, r, \mathit{sender})$, now

handles requests of the form $(\texttt{Receive}, y, f, \mathit{sender})$. The condition verifying of there is no tuple of the form $(\mathit{sender}, \cdot, r, \cdot)$ is still performed but with the tuple of the form $(\mathit{sender}, \cdot, f)$. The random r is replaced by $\perp$ in the code which is executed when the condition is checked. The executed code when the condition fails, including the attribute verification $(x \in y)$, is replaced by the insertion of the file f to the received file register. The simulator, on its side, does not register the pair (r, c_y) anymore.

Observe that all these modifications are hidden to the environment, and we claim that the view of the environment remains unchanged. This can be easily deduced since the at this point, the ideal functionality does not perform the attribute verification by itself but rather delegate this task to the ideal functionality $\mathcal{F}_{\mathsf{ABE}}$, which is secure and correct by design. However, compared to the previous game, the received file is now always f instead of the zero-string. In case where the environment does not have a valid decryption key to decrypt c_y, it encrypts the leakage $L(\lambda, f)$ in this game, instead of $L(\lambda, 0^{|f|})$ in the previous one. Thanks to our specification of L, always outputting a zero-string, both of these leakages are the same. Therefore, by construction of the ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ and by the leakage function L, the environment cannot distinguish, otherwise breaking the security of $\mathcal{F}_{\mathsf{ABE}}$. Hence, $\Pr\left[(\mathcal{E}|\mathcal{S}_2'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}_1'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right]$.

Game 3. At this point, the (simulated) ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ still maintain a set of sent files, essentially used to provide authentication of the files for honest senders. An honest receiver, on its side, verifies that a received file f belongs to the set of sent files (with respect to the provided sender) and responds to the simulator via the network interface if the file is not found. Observe that in this case, the simulator already activates the (honest) receiver with a request `Receive` if and only if the provided signature σ_l authenticates the link l, which corresponds to the hash of the ABE ciphertext c_y. Hence, by construction, authentication of the file with respect to the provided sender is already ensured by the signature ideal functionality $\mathcal{F}_{\mathsf{sig\text{-}CA}}$. Hence, the condition in the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ is no more necessary and all the code handling `Receive` requests is now limited to add the received file f (added in the previous game) to the set of received files. This constitutes our new simulator $\mathcal{S}_3'$. Since the authentication in the ideal functionality $\mathcal{F}_{\mathsf{sig\text{-}CA}}$ is correct and secure by definition, this modification does not impact the view of the environment $\mathcal{E}$ and hence $\Pr\left[(\mathcal{E}|\mathcal{S}_3'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}_2'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right]$.

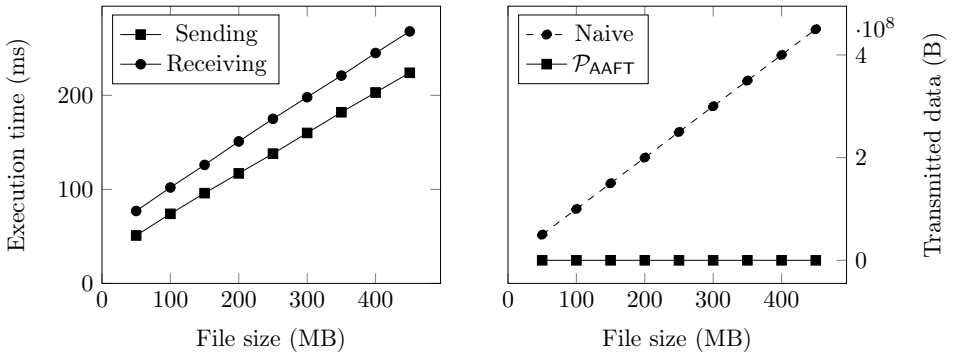
Game 4. This game works as the previous game, except that we remove the attribute state attr that are not used anymore in this game. Additionally, we replace the internal state $\mathsf{receivedFiles}$ by a local internal state specific to each receiver. This last modification clearly does not affect the view of the the environment $\mathcal{E}$, and hence we have $\Pr\left[(\mathcal{E}|\mathcal{S}_4'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{S}_3'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right]$.

Observe that in our last game, the ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ simulated in our last simulator $\mathcal{S}_4'$ never rely on internal state and essentially constitutes a forward machine between the simulated game protocol and the `I/O` interface where the environment is connected. As a result, we claim that $\Pr\left[(\mathcal{E}|\mathcal{S}_4'|\mathcal{F}_{\mathsf{Fwd}})(1^\lambda) \to 1\right] = \Pr\left[(\mathcal{E}|\mathcal{P}_{\mathsf{AAFT}})(1^\lambda) \to 1\right]$. As a result, we have $\mathcal{P}_{\mathsf{AAFT}} \leq \mathcal{F}_{\mathsf{AAFT}}$.

□

5.3.3 Implementation of $\mathcal{P}_{\mathsf{AAFT}}$

Our open-source proof-of-concept [DAF24] written in Rust confirms the practicality of our protocol. We have chosen the Schnorr signature over the Curve25519 [Ber06] curve as our EUF-CMA digital signature. Since our cryptographic hash function handles potentially large files, we have chosen to construct a parallelized Merkle-tree-based hash function using the standard SHA-256 [Dan15] cryptographic hash function as the underlying building block. To construct our IND-CCA2-secure ABE, we have applied the Fujisaki-Okamoto transform [FO13] on the Agrawal-Chase IND-CPA Ciphertext-Policy ABE scheme [AC17] following the idea of Green, Hohenberger and Waters [GHW11]. We have used AES-256-CTR as the secret-key encryption within the transform. Benchmarks have been performed on an Ubuntu, embedding a 64 bits Intel Core i5-6500 processor cadenced at 3.20GHz including four cores, and embedding 16Gb of memory. In Figure 5.12a is depicted the execution time of the sending and receiving procedures depending on the input file size. We directly observe that both procedures are mlinear. For a 450 megabytes file the sending procedure requires approximately 216 milliseconds, whereas the file receiving procedure expects 258 milliseconds. The difference between the sending and reception is explained by the Fujisaki-Okamoto transform, executing during the reception the encryption algorithm, used to reject malformed ciphertext. In Figure 5.12b, we compare the amount of data sent by a sender to a receiver through the low-rate medium between the *naive* corresponding to the situation where the encrypted file is directly sent to the receiver (for example with OpenPGP [FDC+07]), and the approach motivated in the $\mathcal{P}_{\mathsf{AAFT}}$ protocol. Compared to the naive approach, our solution provides a *constant* amount of transferred data of 96 bytes corresponding to the link (*i.e.,* the hash of the encrypted file) and the signature σ_l. This constant communication size is explained by the encrypted file being sent over the distributed storage network instead of being directly transferred, still with the guarantee to have confidentiality and integrity of the file but also authentication of the sender.



(a) Sending and receiving execution time.

(b) Exchanged data size between users.

Figure 5.12: Evaluation of the execution time and the communicated data size.

5.4 Conclusion

Facing the lack of ideal functionality tailored for standard attribute-based encryption, we have propose the first ideal functionality $\mathcal{F}_{\mathsf{ABE}}$ whose realisation called $\mathcal{P}_{\mathsf{ABE}}$ is modelled using standard attribute-based encryption under the iUC framework.

To emphasise the usability of our $\mathcal{F}_{\mathsf{ABE}}$ ideal functionality, we have proposed a study case focusing the Authenticated Attribute-based File Transfer (AAFT) allowing to transfer a file from a sender to a receiver following an access policy y, while ensuring authenticity and integrity of the transferred file. The ideal functionality $\mathcal{F}_{\mathsf{AAFT}}$ implementing all these properties ensures that a receiver having an attribute x such that $x \in y$ has access to the file. All other receivers where $x \notin y$ does not have access to the given file, ensuring confidentiality. In addition, the file is authenticated with respect to some authentication proof produced by the sender, authenticated at the same time the file integrity and the sender.

We have proposed $\mathcal{P}_{\mathsf{AAFT}}$, proved to realize $\mathcal{F}_{\mathsf{AAFT}}$, based on attribute-based encryption ideal functionality $\mathcal{F}_{\mathsf{ABE}}$, digitial signature ideal functionality $\mathcal{F}_{\mathsf{sig\text{-}CA}}$, certificate authority ideal functionality $\mathcal{F}_{\mathsf{CA}}$ and the random oracle ideal functionality $\mathcal{F}_{\mathsf{CA}}$. To improve efficiency for the file transfer, $\mathcal{P}_{\mathsf{AAFT}}$ is also based on the hash-based distributed storage system called IPFS for which we have proposed a protocol within the iUC framework.

From a performance standpoint, the praticality of our protocol has been confirmed by our implementation of $\mathcal{P}_{\mathsf{AAFT}}$ fully written in Rust. Indeed, the observed performance shows that only 216 milliseconds are necessary to send 450 megabytes of data and 258 milliseconds to receive the same amount of data, for a total of 474 milliseconds.

Chapter 6

Conclusion & Perspectives

Three contributions have been introduced in this manuscript, focusing on three distinct problems dealing with multiple users. As with any research work, the presented contributions are designed to tackle precise problems. We draw the future of these contributions and identify possible perspectives.

Secure Federated Multi-Armed Bandits

The two protocols Tango and Salsa presented in this manuscript are proven to securely solve the reward maximisation problem in the federated multi-armed bandits setting, in which the controller (also referred to as the federation server in the literature) acts as a learning agent trying to maximise its rewards by choosing the bandit associated with the highest reward probability. In the federated learning setting, a bandit is modelled as a data owner, holding the unknown reward distribution and interacting only with the controller. A secure federated multi-armed bandits protocol should prevent the controller from learning any information about the bandit having the best reward distribution but also on the generated rewards.

This problem has already been focused on by our recently published protocol called Samba [CLMS22], based on standard cryptographic primitives and working with two controllers. However, we have shown that Samba suffers from correctness and security issues. Our first protocol Tango follows the initial blueprint of Samba, fixing the protocol using fully homomorphic encryption and extending the set of provided properties by allowing a data owner either to join or to leave the protocol dynamically. Due to the usage of fully homomorphic encryption, Tango suffers from a significant computation overhead, around 2.3 seconds to identify the best among 9 bandits.

Motivated to overcome this computation overhead, we have introduced a more scalable protocol called Salsa, moving away from the blueprint of Samba and Tango. Indeed, Salsa works using three distinct controllers whose two servers are involved in secure two-party computations, allowing one to delegate the computation of an arbitrary function over private input without revealing any information on the input, the output, and any intermediate data. This approach has the advantage of being significantly faster, requiring only 0.15 milliseconds to select the best among 9 bandits, allowing Salsa to handle even more bandits at the same time and hence to be scalable.

Despite all these interesting properties, notably from a performance and security standpoint, both Tango and Salsa appear to be more limited regarding the generality property enjoyed by Samba: All of these protocols aim to be generic in the sense that any multi-armed bandits algorithm respecting a precise definition can be plugged into the protocol. It turns out that Samba has a more general definition, allowing

to plug in five multi-armed bandits algorithms whereas Tango and Salsa allow only three multi-armed bandits algorithms to be plugged in. Briefly, Tango and Salsa support multi-armed bandits algorithms whose arm selection depends only on the arg max function. This restriction is due to our security definition considering arg max-based multi-armed bandits algorithm. An interesting future work may consist of designing a more generalised security definition supporting a larger class of algorithms. Given this generalised security definition, one may easily convert Tango or Salsa to support this larger class of multi-armed bandits algorithms.

As an orthogonal future work, we are interested in the linear multi-armed bandits version where a data owner now runs an multi-armed bandits algorithm, locally updating its own local model. Regularly at some time step, the federation server receives the local models from the data owners before to perform the aggregation, leading to an aggregated model replacing the data owner's local model. Adapting our protocols for this setting is an interesting question that we delegate as a future work.

Anonymous, Transferable, Auditable Tickets

Currently deployed ticketing systems offer standard functionalities, namely ticket purchasing, ticket refunding, and ticket validation but suffer from vulnerabilities, including the ability for a user to sell the same ticket several times or to use nominative tickets, preventing a user from using the system anonymously.

Facing these issues, we have developed two ticketing systems called respectively Applause and Spotlight, whose design should provide all standard features, namely ticket purchasing, ticket refunding, and ticket validation. In cases where a user cannot attend an event for which it owns a ticket, Applause and Spotlight provide the ticket transferability feature, allowing the user to transfer its ticket to another user. In contrast with currently deployed ticketing systems relying on nominative tickets, Applause and Spotlight allow a user to use the system without having to reveal its identity.

Despite all these similarities, an important difference remains between Applause and Spotlight. In Applause, the anonymity of users is guaranteed at every time against both the system and all other users. In contrast, Spotlight brings an additional third-party, referred to as the judge, able to reveal the identity of a user attending an event, hence does not ensure anonymity of users against the judge. Despite this additional third-party, Spotlight still guarantees the anonymity of users against the system and other users. As we have observed, this auditability feature implies a slightly longer ticket validation due to the usage of the more complex cryptographic building block Protego [CDLPK22], increasing execution time from 45 for Applause to 165 milliseconds for Spotlight, which is still practical. Other interactions including ticket purchasing, ticket refunding, and ticket validation achieve great performance of 17 milliseconds to purchase a ticket, 68 milliseconds to transfer a ticket, and 25 milliseconds to refund a ticket.

The presented execution times take into account only the time necessary to perform the cryptographic operations. However, during each of these interactions such as ticket purchasing, a few communications occur between a user and the system. One may be

interested in the impact of the network latency on the protocol efficiency, particularly during the ticket validation stage where performance is crucial. Hence, one possible perspective for Applause and Spotlight would be to initiate performance on a more real case.

To purchase a ticket while guaranteeing the anonymity of users, both Applause and Spotlight rely on a generic anonymous payment scheme. However, as we have mentioned, unlinkability of the payments, an important property to ensure unlinkability of interactions between a user and the system, is particularly hard to obtain using well-deployed, commonly used payment systems such as the EMV protocol [EMV22]. Trivial anonymous payment schemes such as anonymous currency like cash or prepaid cards are possible candidates but lack practicality compared to online payment using credit cards. The design of a completely anonymous payment scheme enjoying friction-less interaction for the user is a very interesting research topic that deserves to be explored in future work.

UC-Secure Distributed File Transfer

In Chapter 5, we have focused our attention on the design of a file transfer protocol based on three distinct building blocks: Attribute-based encryption, digital signature and a distributed storage network. In our protocol, the attribute-based encryption allows a user to broadcast a short piece of data, referred later as a link, to an *a priory* known group of users. The so-called link refers to an address on the distributed storage system which is associated to an encrypted of the transferred file. Thanks to the attribute-based encryption, only users having an appropriate attribute are able to recover the transferred file. Along the link, the file sender provides a signature of the link using its signature key of the digital signature scheme, used by a recipient to verify the authenticity of the link and hence the authenticity of the file.

This distributed file transfer protocol has been proven secure under the Universal Composability (UC) paradigm and more particularly under the iUC framework, whose the design has required an ideal functionality for a standard attribute-based encryption, being part of the contribution. From a performance standpoint, the protocol has shown a great performance with approximately 216 milliseconds to send a 450 megabytes file, whereas the file receiving procedure expects 258 milliseconds.

The introduced protocol supports file confidentiality as well as file authenticity and integrity. These three properties are interesting for most use-cases but are not exhaustive. Indeed, it is possible to integrate more functionalities within the protocol, such as anonymity of the sender using group signatures [CvH91, KTY04] or ring signatures [BKM06, FS07], which in a nutshell allows one to sign a message in the behalf of a group. Extending this work constitutes a very exciting direction of improvement.

Bibliography

[AA22] Masayuki Abe and Miguel Ambrona. Blind key-generation attribute-based encryption for general predicates. *Designs, Codes and Cryptography*, 90, 08 2022.

[ABC+15] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2015:1192, 2015.

[AC17] Shashank Agrawal and Melissa Chase. FAME: fast attribute-based message encryption. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 665–682. ACM, 2017.

[AEAKH10] Diaa Salama Abd Elminaam, Hatem Mohamed Abdual-Kader, and Mohiy Mohamed Hadhoud. Evaluating the performance of symmetric encryption algorithms. *Int. J. Netw. Secur.*, 10(3):216–222, 2010.

[AES07] Recommendation for BlockCipher Modes of Operation:Galois/Counter Mode (GCM) and GMAC. `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf`, 2007. NIST Special Publication 800-38D.

[AIRC13] Muhammad Rizwan Asghar, Mihaela Ion, Giovanni Russello, and Bruno Crispo. Espoon erbac: Enforcing security policies in outsourced environments. Cryptology ePrint Archive, Paper 2013/587, 2013. `https://eprint.iacr.org/2013/587`.

[AJJM20] Prabhanjan Vijendra Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *TCC*, 2020.

[AKO24] Yavuz Akın, Jakub Klemsa, and Melek Önen. A practical tfhe-based multi-key homomorphic encryption with linear complexity and low noise growth. In *Computer Security – ESORICS 2023: 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25–29, 2023, Proceedings, Part I*, page 3–23, Berlin, Heidelberg, 2024. Springer-Verlag.

[Ave16] Aventus. `https://aventus.io/`, 2016.

[BCFK15] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable e-cash. In *International Workshop on Public Key Cryptography*. Springer, 2015.

[BDF+20] Johannes Buchmann, Ghada Dessouky, Tommaso Frassetto, Ágnes Kiss, Ahmad-Reza Sadeghi, Thomas Schneider, Giulia Traverso, and Shaza Zeitouni. Safe: A secure and efficient long-term distributed storage system. Cryptology ePrint Archive, Paper 2020/690, 2020. `https://eprint.iacr.org/2020/690`.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BFQ21] Balthazar Bauer, Georg Fuchsbauer, and Chen Qian. Transferable e-cash: A cleaner model and the first practical instantiation. In *IACR International Conference on Public-Key Cryptography*, pages 559–590. Springer, 2021.

[BGGJ19] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar P. Jetchev. Simulating homomorphic evaluation of deep learning predictions. In Dolev S.Hendler D.Lodha S.Yung M., editor, *3rd International Symposium on Cyber Security Cryptography and Machine Learning, CSCML 2019*, volume 11527, pages 212–230, Beer-Sheva, France, 2019.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[BK19] Osman Bicer and Alptekin Kupcu. Versatile abs: Usage limited, revocable, threshold traceable, authority hiding, decentralized attribute based signatures. Cryptology ePrint Archive, Paper 2019/203, 2019. `https://eprint.iacr.org/2019/203`.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. *Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles*, page 60–79. Springer Berlin Heidelberg, 2006.

[BM17] Elaine Barker and Nicky Mouha. *Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher*. November 2017.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume

10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer, 2018.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, page 62–73, New York, NY, USA, 1993. Association for Computing Machinery.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.

[Cas18] CashApp. CashApp. `https://www.cash.app/`, 2018.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from tfhe. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 446–472, Cham, 2019. Springer International Publishing.

[CDEN12] Jan Camenisch, Maria Dubovitskaya, Robert R Enderlein, and Gregory Neven. Oblivious transfer with hidden access control from attribute-based encryption. In *Security and Cryptography for Networks: 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings 8*, pages 559–579. Springer, 2012.

[CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 395–412, New York, NY, USA, 2019. Association for Computing Machinery.

[CDLPK22] Aisling Connolly, Jérôme Deschamps, Pascal Lafourcade, and Octavio Perez Kempner. Protego: Efficient, revocable and auditable anonymous credentials with applications to hyperledger fabric. In *International Conference on Cryptology in India*, pages 249–271. Springer, 2022.

[CDŞ14] Yao Cui, Izak Duenyas, and Özge Şahin. Should event organizers prevent resale of tickets? *Management Science*, 60(9):2160–2179, 2014.

[CES+05] Liqun Chen, Matthias Enzmann, Ahmad-Reza Sadeghi, Markus Schneider, and Michael Steiner. A privacy-protecting coupon system. In *Financial Cryptography and Data Security: 9th International Conference, FC*

2005, Roseau, The Commonwealth Of Dominica, February 28–March 3, 2005. Revised Papers 9, pages 93–108. Springer, 2005.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

[CGGI19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33, 04 2019.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06. Association for Computing Machinery, 2006.

[CHY23] Younghwan Choi, Yong-Geun Hong, and Joo-Sang Youn. Transmission of IPv6 Packets over Near Field Communication. RFC 9428, July 2023.

[CKK20] J. H. Cheon, D. Kim, and D. Kim. Efficient Homomorphic Comparison Methods with Optimal Complexity. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 221–256, 2020.

[CKKR19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iuc: Flexible universal composability made simple. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 191–221. Springer, 2019.

[CLLS19] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare. Secure Best Arm Identification in Multi-Armed Bandits. In *International Conference on Information Security Practice and Experience (ISPEC)*, pages 152–171, 2019.

[CLLS20] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare. Secure Outsourcing of Multi-Armed Bandits. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 202–209, 2020.

[CLMS22] Radu Ciucanu, Pascal Lafourcade, Gael Marcadet, and Marta Soare. SAMBA: A generic framework for secure federated multi-armed bandits. *J. Artif. Intell. Res.*, 73:737–765, 2022.

[CMÖ11] Leucio Antonio Cutillo, Refik Molva, and Melek Önen. Safebook: A distributed privacy preserving online social network. In *12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WOWMOM 2011, Lucca, Italy, 20-24 June, 2011*, pages 1–3. IEEE Computer Society, 2011.

[CMÖ12] Leucio Antonio Cutillo, Refik Molva, and Melek Önen. Privacy preserving picture sharing: enforcing usage control in distributed on-line social networks. In Eiko Yoneki, Davide Frey, and Ian Brown, editors, *Proceedings of the Fifth Workshop on Social Network Systems, Bern, Switzerland, April 10, 2012*, page 6. ACM, 2012.

[CvH91] David Chaum and Eugène van Heyst. *Group Signatures*, page 257–265. Springer Berlin Heidelberg, 1991.

[CZW17] Long Chen, Zhenfeng Zhang, and Xueqing Wang. Batched multi-hop multi-key fhe from ring-lwe with compact ciphertext extension. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 597–627, Cham, 2017. Springer International Publishing.

[DAF24] Daft: Proof-of-concept. `https://anonymous.4open.science/r/DAFT/`, 2024.

[Dan15] Quynh H. Dang. *Secure Hash Standard*. July 2015.

[DBN+01] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001.

[DGGB22] Yuhao Dong, Ian Goldberg, Sergey Gorbunov, and Raouf Boutaba. Astrape: Anonymous payment channels with boring cryptography. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 748–768. Springer, 2022.

[DP20] A. Dubey and A. Pentland. Differentially-Private Federated Linear Bandits. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[DR14] C. Dwork and A. Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[DSZ15] Daniel Demmler, T. Schneider, and Michael Zohner. Aby - a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security Symposium*, 2015.

[dV18] Alex de Vries. Bitcoin's growing energy problem. 2:801–805, 05 2018.

[Elg85a] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[ElG85b] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 1985.

[EMV11] EMVCo. Book 1: Application independent icc to terminal interface requirements. 2011.

[EMV22] LLC EMVCo. Emv payment tokenisation specification technical framework v2.3. *EMVCo: Foster City, CA, USA*, 2022.

[EY80] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Technical report, Computer Science Department, Technion, 1980.

[FB23] Jörg Schwenk Fabian Bäumer, Marcus Brinkmann, 2023.

[FDC+07] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and Daphne Shaw. OpenPGP Message Format. RFC 4880, November 2007.

[FFW13] Anna Lisa Ferrara, George Fuchsbauer, and Bogdan Warinschi. Cryptographically enforced rbac. Cryptology ePrint Archive, Paper 2013/492, 2013. `https://eprint.iacr.org/2013/492`.

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 2019.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptol.*, 26(1):80–101, 2013.

[FPP+02] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. drbac: distributed role-based access control for dynamic coalition environments. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 411–420, 2002.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. *Traceable Ring Signature*, page 181–200. Springer Berlin Heidelberg, 2007.

[Gao18] Shuhong Gao. Efficient fully homomorphic encryption scheme. Cryptology ePrint Archive, Paper 2018/637, 2018. `https://eprint.iacr.org/2018/637`.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[GGJR98] Juan A. Garay, Rosario Gennaro, Charanjit Jutla, and Tal Rabin. Secure distributed storage and retrieval. Cryptology ePrint Archive, Paper 1998/025, 1998. https://eprint.iacr.org/1998/025.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts. In *20th USENIX Security Symposium (USENIX Security 11)*, San Francisco, CA, August 2011. USENIX Association.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[GSK14] Ivan Gudymenko, Felipe Sousa, and Stefan Kopsell. A simple and secure e-ticketing system for intelligent public transportation based on NFC. In *The First International Conference on IoT in Urban Space, Urb-IoT*. ICST, 2014.

[Gud13] Ivan Gudymenko. On protection of the user's privacy in ubiquitous e-ticketing systems based on RFID and NFC technologies. In *PECCS 2013 - Proceedings of the 3rd International Conference on Pervasive Embedded Computing and Communication Systems*. SciTePress, 2013.

[HAY12] N. Abdul Hamid, M. F. Al A'zhim, and M. L. Yap. e-ticketing system for football events in malaysia. In *7th International Conference for Internet Technology and Secured Transactions, ICITST*. IEEE, 2012.

[HCE05] Wan Huzaini Wan Hussin, Paul Coulton, and Reuben Edwards. Mobile ticketing system employing trustzone technology. In *International Conference on Mobile Business*. IEEE Computer Society, 2005.

[HCS+21] Jinguang Han, Liqun Chen, Steve Schneider, Helen Treharne, and Stephan Wesemeyer. Privacy-preserving electronic ticket scheme with attribute-based credentials. *IEEE Trans. Dependable Secur. Comput.*, 2021.

[HK16] F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19:1–19:19, 2016.

[HKN05] Shai Halevi, Paul A. Karger, and Dalit Naor. Enforcing confinement in distributed storage and a cryptographic model for access control. *IACR Cryptol. ePrint Arch.*, page 169, 2005.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, page 511–542, Berlin, Heidelberg, 2023. Springer-Verlag.

[HMU05] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Polynomial runtime in simulatability definitions. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence, France*, pages 156–169. IEEE Computer Society, 2005.

[HWYS21] R. Huang, W. Wu, J. Yang, and C. Shen. Federated Linear Contextual Bandits. In *Conference on Neural Information Processing Systems (NIPS)*, 2021.

[Int06] International Organization for Standardization. Information technology — automatic identification and data capture techniques — qr code 2005 bar code symbology specification. ISO/IEC 18004:2006, 2006.

[KLG13] Florian Kerschbaum, Hoon Wei Lim, and Ivan Gudymenko. Privacy-preserving billing for e-ticketing systems in public transportation. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*. ACM, 2013.

[KLSW24] Hyesun Kwak, Dongwon Lee, Yongsoo Song, and Sameer Wagh. A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 403–430, Cham, 2024. Springer Nature Switzerland.

[KMea21] P. Kairouz, H. B. McMahan, and et al. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[KMS22] Hyesun Kwak, Seonhong Min, and Yongsoo Song. Towards practical multi-key tfhe: Parallelizable, key-compatible, quasi-linear complexity. *IACR Cryptol. ePrint Arch.*, 2022:1460, 2022.

[KP14] V. Kuleshov and D. Precup. Algorithms for Multi-Armed Bandit Problems. *CoRR*, abs/1402.6028, 2014.

[KS06] L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning (ECML)*, 2006.

[KTR20a] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: A simple and expressive model for universal composability. *J. Cryptol.*, 33(4):1461–1584, 2020.

[KTR20b] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. Joint state composition theorems for public-key encryption and digital signature functionalities with local computation. *J. Cryptol.*, 33(4):1585–1658, 2020.

[KTY04] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. *Traceable Signatures*, page 571–589. Springer Berlin Heidelberg, 2004.

[LCLS10] L. Li, W. Chu, J. Langford, and R. E. Schapire. A Contextual-bandit Approach to Personalized News Article Recommendation. In *International Conference on World Wide Web (WWW)*, 2010.

[LLG01] Xiaoshan Li, Zhiming Liu, and Zhensheng Guo. Formal object-oriented analysis and design of an online ticketing system. In *8th Asia-Pacific Software Engineering Conference (APSEC 2001)*. IEEE Computer Society, 2001.

[LMMOA24] Pascal Lafourcade, Dhekra Mahmoud, Gael Marcadet, and Charles Olivier-Anclin. Transferable, auditable and anonymous ticketing protocol, 2024. `https://github.com/gamarcad/ETS`.

[LMY14] Weiwei Liu, Yi Mu, and Guomin Yang. An efficient privacy-preserving e-coupon system. In *International Conference on Information Security and Cryptology*. Springer, 2014.

[LNGH19] Xuelian Li, Jie Niu, Juntao Gao, and Yue Han. Secure electronic ticketing system based on consortium blockchain. *KSII Trans. Internet Inf. Syst.*, 2019.

[LS14] Phillip Leslie and Alan Sorensen. Resale and rent-seeking: An application to ticket markets. *Review of Economic Studies*, 81(1):266–300, 2014.

[LSF20] T. Li, L. Song, and C. Fragouli. Federated Recommendation System via Differential Privacy. In *International Symposium on Information Theory (ISIT)*, pages 2592–2597, 2020.

[LW16] Bin Liu and Bogdan Warinschi. Universally composable cryptographic role-based access control. Cryptology ePrint Archive, Paper 2016/902, 2016. `https://eprint.iacr.org/2016/902`.

[MDND15] Milica Milutinovic, Koen Decroix, Vincent Naessens, and Bart De Decker. Privacy-preserving public transport ticketing system. In *Data and Applications Security and Privacy- 29th Annual IFIP WG 11.3 Working Conference, DBSec*. Springer, 2015.

[MHP21] A. Mitra, H. Hassani, and G. Pappas. Exploiting Heterogeneity in Robust Federated Best-Arm Identification. *CoRR*, abs/2109.05700, 2021.

[MPR11] Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *Cryptographers' track at the RSA conference*, pages 376–392. Springer, 2011.

[MSJP22] Akash Madhusudan, Mahdi Sedaghat, Philipp Jovanovic, and Bart Preneel. Nirvana: Instant and anonymous payment-guarantees. *IACR Cryptol. ePrint Arch.*, page 872, 2022.

[MSM23] Omid Mir, Daniel Slamanig, and René Mayrhofer. Threshold delegatable anonymous credentials with controlled and fine-grained delegation. *IEEE Transactions on Dependable and Secure Computing*, 2023.

[NAJ15] Lekshmi S. Nair, V. S. Arun, and Sijo Joseph. Secure e-ticketing system based on mutual authentication using RFID. In *Proceedings of the Third International Symposium on Women in Computing and Informatics, WCI 2015*. ACM, 2015.

[Ngu06] Lan Nguyen. Privacy-protecting coupon system revisited. In *Financial Cryptography and Data Security: 10th International Conference, FC 2006 Anguilla, British West Indies, February 27-March 2, 2006 Revised Selected Papers 10*, pages 266–280. Springer, 2006.

[NSI+15] Johanna Nieminen, Teemu Savolainen, Markus Isomaki, Basavaraj Patil, Zach Shelby, and Carles Gomez. IPv6 over BLUETOOTH(R) Low Energy. RFC 7668, October 2015.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Pan16] Madhumita Panda. Performance analysis of encryption algorithms for security. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 278–284. IEEE, 2016.

[Par21] Jeongeun Park. Homomorphic encryption for multiple users with less communications. *IEEE Access*, 9:135915–135926, 2021.

[Pay18] PaySafeCard. Paysafecard. `https://www.paysafecard.com/fr/`, 2018.

[Pro16] GET Protocol. GUTS Ticketing. `https://guts.tickets/`, 2016.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptol.*, 13(3):361–396, 2000.

[PSSY21] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved Mixed-Protocol secure Two-Party computation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2165–2182. USENIX Association, August 2021.

[RBCS23] Alexandre Rio, Merwan Barlier, Igor Colin, and Marta Soare. Multi-agent best arm identification with private communications. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 29082–29102. PMLR, 23–29 Jul 2023.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[RLRD78] Len Adleman Ronald L. Rivest and Michael L. Dertouzos. On data banks and privacy homomorphisms. 1978.

[RNTS07] Jason Reid, Juan M Gonzalez Nieto, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 204–213, 2007.

[Rob22] Léo Robert. *Design and analysis of provably secure protocols: Applications to messaging and attestation. (Conceptions et analyses de protocoles en sécurité prouvable: applications aux messageries et à l'attestation)*. PhD thesis, University of Clermont Auvergne, Clermont-Ferrand, France, 2022.

[RVK+18] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen. A Tutorial on Thompson Sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

[Sab13] Nicolas Van Saberhagen. Monero Research Paper, 2013.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[SBFK20] Johannes Sedlmeir, Hans Buhl, Gilbert Fridgen, and Robert Keller. The energy consumption of blockchain technology: Beyond myth. *Business & Information Systems Engineering*, 62, 12 2020.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 1991.

[SFB+23] Andrei Stoian, Jordan Frery, Roman Bredehoft, Luis Montero, Celia Kherfallah, and Benoit Chevallier-Mames. Deep neural networks for encrypted inference with tfhe. Cryptology ePrint Archive, Paper 2023/257, 2023. `https://eprint.iacr.org/2023/257`.

[Sho04] Victor Shoup. Sequences of games: A tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 01 2004.

[SS21] C. Shi and C. Shen. Federated Multi-Armed Bandits. In *AAAI Conference on Artificial Intelligence*, 2021.

[SSV08] Marc Sel, Stefaan Seys, and Eric R. Verheul. The security of mass transport ticketing systems. In *ISSE 2008 - Securing Electronic Busines Processes, Highlights of the Information Security Solutions Europe 2008 Conference*, 2008.

[SSY21] C. Shi, C. Shen, and J. Yang. Federated Multi-armed Bandits with Personalization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2917–2925, 2021.

[STMC+23] Meltem Sonmez Turan, Kerry McKay, Donghoon Chang, Lawrence E Bassham, Jinkeon Kang, Noah D Waller, John M Kelsey, and Deukjo Hong. *Status report on the final round of the NIST lightweight cryptography standardization process*. June 2023.

[SXXS21] C. Shi, H. Xu, W. Xiong, and C. Shen. (Almost) Free Incentivized Exploration from Decentralized Learning Agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[TD16] A. C. Y. Tossou and C. Dimitrakakis. Algorithms for Differentially Private Multi-Armed Bandits. In *AAAI Conference on Artificial Intelligence*, pages 2087–2093, 2016.

[TFS04] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-times anonymous authentication (extended abstract). In *Advances in Cryptology - ASIACRYPT 2004*. Springer, 2004.

[TH16] Hitesh Tewari and Arthur Hughes. Fully anonymous transferable ecash. *Cryptology ePrint Archive*, 2016.

[Tho33] W. R. Thompson. On the Likelihood that One Unknown probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25:285–294, 1933.

[TTS+24] Daniel Truhn, Soroosh Tayebi Arasteh, Oliver Lester Saldanha, Gustav Müller-Franzes, Firas Khader, Philip Quirke, Nicholas P. West, Richard Gray, Gordon G.A. Hutchins, Jacqueline A. James, Maurice B. Loughrey, Manuel Salto-Tellez, Hermann Brenner, Alexander Brobeil, Tanwei Yuan, Jenny Chang-Claude, Michael Hoffmeister, Sebastian Foersch, Tianyu Han, Sebastian Keil, Maximilian Schulze-Hagen, Peter Isfort, Philipp Bruners, Georgios Kaissis, Christiane Kuhl, Sven Nebelung, and Jakob Nikolas Kather. Encrypted federated learning for secure decentralized collaboration in cancer image analysis. *Medical Image Analysis*, 92:103059, 2024.

[TY24] Tamir Tassa and Avishay Yanai. The multiple millionaires' problem. Cryptology ePrint Archive, Paper 2024/005, 2024. `https://eprint.iacr.org/2024/005`.

[VGV97] M. Vandenwauver, R. Govaerts, and J. Vandewalle. *Role Based Access Control in Distributed Systems*, pages 169–177. Springer US, Boston, MA, 1997.

[Web04] Object storage devices. `https://webstore.ansi.org/standards/incits/ansiincits4002004`, 2004. Accessed: 2010-09-30.

[XTB04] Minhui Xie, Mark Tomlinson, and Bobby Bodenheimer. Interface design for a modern software ticketing system. In *Proceedings of Annual Southeast Regional Conference*. ACM, 2004.

[Yao82] Andrew C. Yao. Protocols for secure computations. *Annual Symposium on Foundations of Computer Science - Proceedings*, pages 160–164, 1982.

[YZ22] Yuan YuanJiang and Ji Ting Zhou. Ticketing system based on NFT. In *24th IEEE International Workshop on Multimedia Signal Processing, MMSP 2022*. IEEE, 2022.

[Zam20] Zama. Concrete tfhe library from zama. `https://docs.zama.ai/tfhe-rs`, 2020.

[ZZLL21] Z. Zhu, J. Zhu, J. Liu, and Y. Liu. Federated Bandit: A Gossiping Approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(1), 2021.

Appendix A

Résumé Long

La notion de *protocole* est centrale dans la diffusion d'information et la communication au sein d'un réseau. Un protocole se définit par une séquence d'échanges entre plusieurs machines dans le but de réaliser une tâche spécifique. Prenons comme exemple celui du protocole HyperText Transfer Protocol Secure (HTTPS), qui permet le transfert sécurisé de pages internet. Plus généralement, un protocole comme HTTPS doit garantir un ensemble de propriétés de sécurité telles que l'intégrité, l'authenticité et la confidentialité des données échangées.

Une propriété de sécurité est une contrainte que le protocole doit respecter. Pour satisfaire les propriétés de sécurité, le concepteur de protocoles dispose d'un ensemble de primitives cryptographiques. Ces primitives cryptographiques englobent un large éventail de fonctionnalités, allant du chiffrement des données à la signature électronique, en passant par le calcul directement sur des données chiffrés. Une fois intégrées dans une séquence d'échanges, ces primitives cryptographiques seront utilisées pour fournir une preuve démontrant que le protocole respecte l'ensemble des propriétés de sécurité exigées et qu'il constitue ainsi une solution viable pour réaliser une tâche donnée, tout en respectant les contraintes de sécurité.

Une multitude de protocoles cryptographiques peuvent exister permettant de résoudre un problème donné, tout en respectant un ensemble de contraintes de sécurité. Ces protocoles ne sont pas tous équivalents. En effet, l'utilisation d'une primitive cryptographique peut entraîner des différences drastiques de performance par rapport à une autre primitive. De plus, un nombre élevé d'utilisateurs du protocole impose une contrainte de performance que le concepteur doit prendre en compte pour proposer des protocoles pertinents.

Dans ce manuscrit, nous nous intéressons à trois problèmes impliquant plusieurs utilisateurs, d'une dizaine à plusieurs centaines. Dans le premier chapitre, nous étudions la conception d'un système de recommandation basé sur de l'apprentissage par renforcement. Le second et troisième problèmes se focalisent respectivement sur la conception d'un système de billets, capable de supporter jusqu'à un millier de visiteurs, et la conception d'un système de transfert de fichiers efficaces conçu pour une grande entreprise répartie sur plusieurs sites.

Protocole de Bandits Sécurisé dans un Contexte Fédéré

Un algorithme de bandits est un algorihme d'apprentissage par renforcement, dans lequel un agent dipose d'un ensemble de choix. Chaque choix est associé à une probabilité de produire une récompense binaire: soit zéro, l'agent n'est alors pas récompensé, soit un et l'agent est récompensé. La probabilité d'obtenir une récompense en choisissant le i-ème

choix, dénotée μ_i. En disposant de toutes les probabilités de récompenses, il est facile de maximiser ses récompenses en ne faisant que le choix de la plus haute probabilité de récompense. Toutefois, le problème devient plus délicat lorsque l'agent ne *dipose pas* des probabilités des récompenses.

Ce problème est référé dans la litérature comme le problème de maximisation des récompense des bandits, dont la représentation commune est celle d'un casino: Supposons un agent disposant de N pièces face à K bandits, un bandit étant une machine à sous équipé d'un bras mécanique que l'agent actionne après avoir inséré une pièce, le bandit retournant soit une récompense avec une certaine probabilité μ soit aucune récompense avec probabilité $1 - \mu$. L'agent va au fur et à mesure de ses tirages, orienter son tirage sur le bandit avec la plus haute probabilité de récompense. Beaucoup d'algorithmes présents dans la litérature sont capables de résoudre ce problème, comme par exemple les algorithmes UCB, Thompson Sampling ou bien encore ϵ-greedy introduits dans [KP14].

Ce problème de maximisation des récompenses pour des bandits peut prendre des formes très diverses selon le contexte. Dans ce travail, nous nous focalisons sur une architecture dîte fédérée, où chaque bandit est *indépendant* des autres bandits. Dans cette architecture fédérée, l'agent souhaitant maximiser ses récompenses est représenté par un serveur de fédération. Cette architecture distribuée est représentée en Figure A.1. Bien que le problème semble identique, nous pouvons faire deux remarques: premièrement, chaque bandit est indépendant des autres, le i-ème bandit est capable de produire une récompense dont la probabilité est paramétrée par une probabilité μ_i. En pratique, μ_i n'est pas connu par le i-ème bandit, mais dispose d'une fonction noté pull capable de produire une récompense, cette fonction étant spécifique au contexte dans lequel on se place. Par exemple, dans un contexte de recommandation de catégorie de films, le i-ème bandit peut correspondre à un genre de films par exemple- aux films d'actions, une possible implémentation de pull serait ainsi de jouer une film d'action et de retourner 1 si les retours d'un spectateur cible sont bons, 0 autrement.

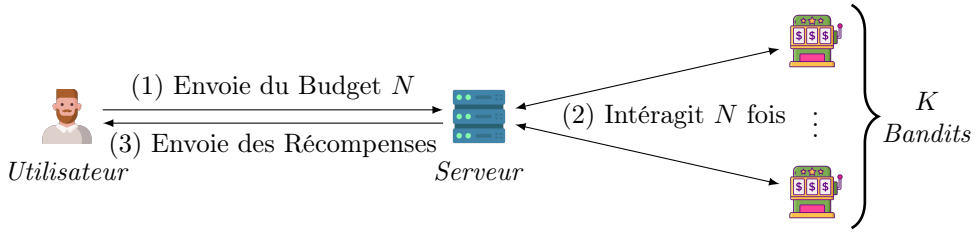


Figure A.1: Architecture des Bandits à Fédérés.

Dans le contexte fédéré, le serveur permet à un utilisateur du système de déléguer la tâche de maximisation des récompenses au serveur, lui donnant le budget initial N, qui lui retourne la somme de toutes les récompenses générées par les bandits une fois le problème de maximisation des récompenses résolu dans un contexte fédéré par un des algorithmes mentionnés. Toutefois, dans un souci de confidentialité, il serait souhaitable que le serveur ne puisse pas identifier le bandit. Par ailleurs, le serveur ne doit pas connaître l'ensemble des récompenses obtenues, même si celui-ci reçoit les

récompenses générées localement par les bandits avec d'en faire la somme et de retourner le somme des récompenses générées à l'utilisateur. Toutes ces contraintes, en plus du problème initial de maximisation des récompenses pour des bandits dans un contexte fédéré, constitue le problème de maximisation des récompenses pour des bandits fédéré sécurisé.

Ce problème a déjà été considéré par notre publication datant de 2022, introduisant le framework nommé SAMBA [CLMS22]. Informellement, ce framework permet à un algorithme de bandits, respectant une condition précise, de se voir intégrer dans SAMBA pour résoudre le problème de maximisation des récompenses fédéré sécurisé. Ainsi, ce framework transforme un algorithme de bandits en un protocole pour résoudre le même problème, cette fois dans un contexte fédéré et de façon sécurisé, c'est-à-dire sans que le serveur ne puisse identifier le bandit ayant la meilleure probabilité de récompenses, mais également les récompenses générées. Toutefois, nous avons montré que SAMBA souffre de deux problèmes: tout d'abord, il apparaît que la récompense retournée par un algorithme de bandits intégré dans SAMBA ne soit pas identique dans tous les cas comparé à l'algorithme de bandits seul. Par ailleurs, le serveur peut malgré tout retrouver de l'information, en particulier sur la probabilité des scores ce qui compromet la sécurité attendue de SAMBA.

Pour surmonter ces deux faiblesses, nous présentons deux protocoles appelés respectivement TANGO et SALSA, capables de résoudre le problème de maximisation des récompenses pour les bandits dans un contexte fédéré. Informellement, TANGO repose sur la même architecture que SAMBA, la différence majeure étant la répartition du travail et ainsi des primitives utilisées: dans SAMBA, le serveur est en réalité deux serveurs, l'un des deux servant de proxy recevant K valeurs provenant des bandits, qui sont ensuites transférées à l'autre serveur pour identifier le meilleur bandit. Ces valeurs sont cruciales dans l'algorithme de bandit mais doivent être gardées secrètes, même s'il est nécessaire de les comparer pour identifier le meilleur bandit. À la différence de SAMBA, dans TANGO, cette comparison est réalisée directement par le proxy sur des valeurs qui sont ici *chiffrées* [Gen09]. Le résultat de la comparaison est ensuite transmis au second serveur. Nous avons montré formellement que TANGO garantie la confidentialité des récompenses générées et de l'identité du bandit choisi pour chaque pièce disponible, et donc par extension l'identité du bandit avec la meilleure probabilité de récompenses. La sécurité de TANGO a été montrée au regard d'un modèle de sécurité capturant toutes les contraintes de sécurité, modèle que nous avons introduit.

Bien que sûr, TANGO a un problème majeur: TANGO est construit à l'aide d'un chiffrement sur lequel il est possible de faire des calculs. Appelé chiffrement homomorphe, ce genre de chiffrement souffre d'un temps d'exécution significatif. Ainsi, TANGO peut paraître lent. En effet, identifier le meilleur parmi 9 bandits nécessite 2.3 secondes, ce qui est problématique vis-à-vis des cas d'applications temps réel comme par exemple en finance pour des recommandations d'actions.

Pour pallier à ce problème, nous proposons un autre protocole SALSA, dont l'approche change considérablement. En effet, ce protocole se base sur du calculs multi-parties sécurisées, dont l'idée générale consiste à répartir une donnée sur plusieurs serveurs, chaque serveur reçevant ce qui est appelé une *part*, une part n'étant pas suffisant pour recouvrer la donnée initiale. Toutefois, comme il a été démontré, il est possible de simuler

un circuit arbitraire sur les parts de sorte à ce que de nouvelles parts, une fois combinée, produise le résultat attendu [DSZ15, PSSY21]. Cette approche nous permet d'atteindre de bonnes performances, nécessitant seulement 0.15 milliseconde pour choisir le meilleur bandit parmi neufs, là où il faut 2.3 secondes pour TANGO. La sécurité de SALSA a été démontré formellement en utilisant le même modèle de sécurité que TANGO.

Système Auditable de Billets Anonyme Transférable

Les systèmes de billets actuellement en place et utilisés par des milliers d'utilisateurs fournissent les fonctionalités standards includant l'achat et le remboursement d'un billet, mais également la validation du billet à l'entrée de l'événement. Toutefois, avec la dématérialisation des billets, il est devenu facile de dupliquer son billet ou bien encore de le vendre en ligne. Par ailleurs, un billet est systèmatiquement nominatif ce qui impliques deux faits: premièrement, il est devient alors impossible pour un utilisateur de transférer son billet à un autre utilisateur, par exemple à un proche. De plus un utilisateur du système devient alors parfaitement identifiable.

Dans un soucis de conserver la vie privée des utilisateurs, nous avons décider de convevoir un système de billets qui surmonte ces limitations. Nommé APPLAUSE, ce protocole offre à l'utilisateur les fonctionalités standards includant l'achat et le reboursement d'un billet, mais également la validation d'un billet à l'entrée d'un événement. Mais il permet également de transférer son billet à un proche, une fonctionalité qui nous paraît essentielle qui permet une résoudre une situation courante où un utilisateur ne peut plus se rendre à l'événement. En plus de ces fonctionalités, un utilisateur est garanti que quelque soit l'intéraction qu'il établit avec le système ou avec un autre utilisateur, l'identité de l'utilisateur ne sera jamais divulgué, lui garantissant ainsi son anonymat.

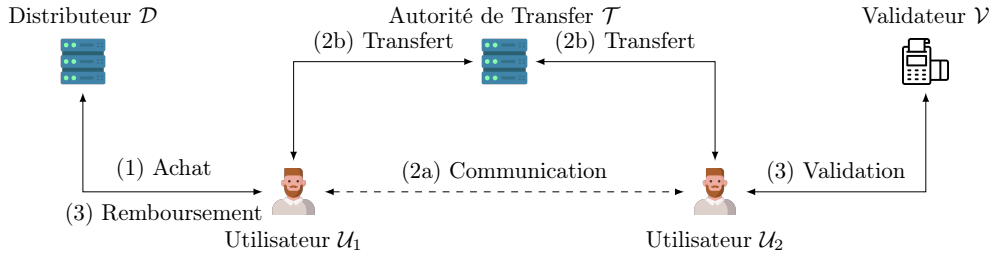


Figure A.2: Architecture pour le Système de Billets.

Informellement, notre protocole APPLAUSE est construit selon l'architecture présentée en Figure A.2. Pour acheter un billet, un utilisateur doit intéragir avec le Distributeur $\mathcal{D}$ avec qui il va exécuter le protocole d'achat. Ce protocole résulte pour l'utilisateur en l'obtention d'un ticket valide. Le Distributeur peut également procédé à son remboursement s'il le souhaite en exécutant le protocole de Remboursement. Par ailleurs, l'utilisateur peut transférer son billet à un autre utilisateur. Pour garantir la validité du transfert, une autorité est considérée dont le rôle est exclusivement dédiée à la validation du transfert. Le transfer d'un billet ne suppose pas nécessairement que les deux utilisateurs se connaissent mais doivent disposer d'un moyen de communication comme par

exemple du Bluetooth [NSI+15] afin d'échanger du matériel cryptographique essentiel au bon déroulement du protocole de tranfert. Enfin, le système permet à un utilisateur détenant un billet de le valider auprès du validateur à l'entrée de l'événement, l'entrée étant autorisée seulement si le billet détenu est valide. Le protocole dans son intégralité repose sur un ensemble de primitives efficaces, ce qui lui permet d'atteindre des performances cryptographiques élevées: 17 milisecondes pour l'achat d'un billet, 68 millisecondes pour le transfert d'un billet, 25 millisecondes pour le remboursement et 45 millisecondes pour la validation, les temps présentés prenant en compte seulement les opérations cryptographiques et non le temps de communication.

L'anonymat d'un utilisateur est une propriété intéressante. Mais dans une situation extrême comme un incendie, il est impératif d'être en mesure d'identifier les utilisateurs participants un événements. Toutefois, l'anonymat étant garantie quelque soit l'intéraction avec le système, il devient alors difficile de surmonter ces contraintes. Pour pallier à ce problème, nous avons construit une variante d'Applause nommée Spotlight, capable de retrouver l'identité d'un utilisateur ayant accédé à l'événement. Plus précisément, lors de l'entrée à l'événement, le validateur va demander à l'utilisateur de lui fournir ce que l'on désigne par une *preuve* qu'il détient un certificat, fourni par une entité tierce nommée le juge. Le point clé pour obtenir cette capacité à retrouver l'identité de l'utilisateur réside dans la capacité du juge à retrouver l'identité de l'utilisateur à partir de cette preuve. Le reste du système reste incapable d'identifier l'utilisateur, même ayant accès à cette preuve fournie par l'utilisateur lors de l'entrée à l'événement. En terme de performance, de par sa proximitié avec Applause, notre protocole Spotlight reproduit presque les mêmes temps d'exécution. La seule différence apparaît pendant la validation du billet qui passe de 45 millisecondes à 165 millisecondes, ce qui reste tout de même pertinent pour un cas d'usage réél.

Transfert de Fichiers Distribuées Basé sur Attributs

Le transfer de fichiers en utilisateurs, en particulier lorsqu'un nombre d'utilisateurs devient grand, doit être efficace. Actuellement, la solution la plus courante pour envoyer un fichier entre un utilisateur et un groupe d'utilisateurs consiste à envoyer le fichier par courriel, le fichier étant conservé sur le serveur de mail pour récupération. Toutefois, cela n'est pas optimal lorsque le nombre d'utilisateurs augmentent, encore plus lorsque les utilisateurs sont éloignés du serveur, ce qui accroît la durée nécessaire pour récupérer un fichier dû à la latence du réseau sur lequel les données transmises. Ce problème s'intensifie lorsque de nouvelles propriétés doivent être garanties comme la confidentialité des fichiers échangés, l'intégrité du fichier ou bien encore l'authenticité du fichier.

Notre objectif est de répondre à cette problèmatique en envisgeant une solution efficace et capable de s'adapter à un problème à large échelle, comme par exemple dans le cas d'une entreprise mondiale disposant d'une multitude de bureaux répartis à travers le monde. Notre protocole, présenté en Figure A.3, répond à cette problématique de façon efficace via l'utilisation de primitives simples. En substance, le protocole démarre avec un utilisateur diposant d'un fichier qu'il souhatie envoyer à un groupe d'utilisateur. Chaque utilisateur de ce groupe dispose d'un attribut qui l'identifie comme faisant partie

de ce groupe. Ceci est crucial car l'expéditeur va alors chiffrer son fichier en associant une politique d'accès. Plus précisément, seuls les utilisateurs disposant d'un attribut satisfaisant cette politique d'accès est susceptible de déchiffrer ce fichier. De ce chiffré sera ensuite dérivé un lien qui sera par la suite signé, garantissant l'intégrité du fichier retourné.

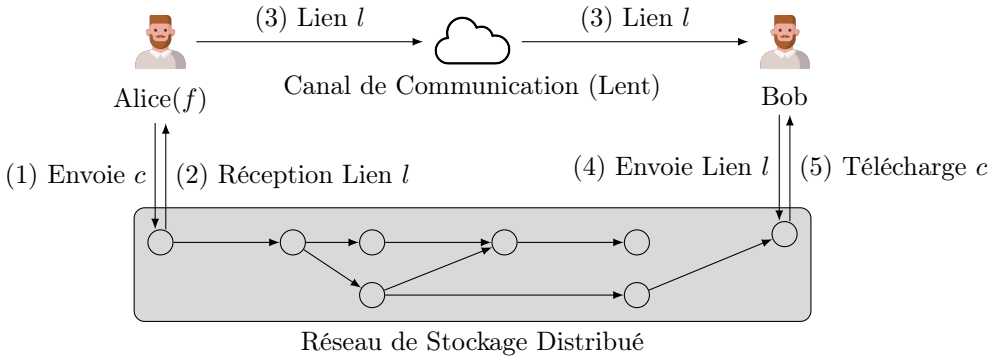


Figure A.3: Représentation de notre système où un expéditeur partage un fichier f à un receveur.