Security of cryptographic protocols and data structures : design and optimisation

Marius Lombard-Platet

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École normale supérieure

Security of cryptographic protocols and data structures: design and optimisation

Soutenue par
Marius Lombard-Platet
Le 08 septembre 2021

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Bart PRENEEL KU Leuven	*Président du jury*
Gildas AVOINE INSA Rennes	*Rapporteur*
Moti YUNG University of Columbia	*Rapporteur*
Michel ABDALLA École normale supérieure	*Membre du jury*
Pascal PAILLIER Zama	*Membre du jury*
Mariana RAYKOVA Google	*Membre du jury*
Pascal LAFOURCADE Université Clermont-Auvergne	*Codirecteur de thèse*
David NACCACHE École normale supérieure	*Codirecteur de thèse*

1794
ENS | PSL

Remerciements

"Schlak, pof".

Je ne saurais dire de quand me date le goût des maths, mais il est certain que cette flamme a été avivée tôt chez moi par les cours énergiques de Madame Farchi, et pour cela, merci.

J'adresse également mes plus vifs remerciements à mes directeurs de thèse, David Naccache et Pascal Lafourcade, tous deux plus motivés et motivants l'un que l'autre, et toujours disponibles pour discuter sécurité, cryptographie ou n'importe quel autre sujet à des horaires parfois raisonnables et parfois franchement absurdes, mais toujours dans la bonne humeur. Merci de m'avoir encadré, de m'avoir initié à la recherche, et de m'avoir fait trouver la voie, sans pour autant que j'y perde la tête.

David, merci de m'avoir pris sous ton aile dès le stage de fin d'études, et de ne jamais hésiter à me pousser hors de ma zone de confort. Merci de toujours vouloir le meilleur pour moi, merci pour ton humour toujours présent. Pascal, merci de m'avoir fait une place dans cette inconnue qu'était l'Auvergne. Merci également d'avoir toujours eu confiance en moi, de toujours discuter de nouvelles idées et de m'avoir enduré et remotivé pendant mes périodes de doute. Également merci d'avoir relu cette thèse.

Thank you to Michel Abdalla, Gildas Avoine, Pascal Paillier, Bart Preneel, Mariana Raykova, Moti Yung, for having accepted to be part of this jury. Thank you especially for Gildas Avoine and Moti Yung for being thesis reporters. I am honoured to present this work to you.

Je tiens également à remercier tous mes coauteurs, avec qui les échanges ont toujours été enrichissants, productifs, et m'ont chacun aidé à grandir tant scientifiquement qu'humainement. Camille, Cheng-Kang, Claudia, David, Hsiao-Ying, Marta, Mirko, Pascal, Pierre, Radu, Rose, Rémi, Yao, thank you all guys for the opportunity and experience. Publishing with you was always a pleasure, with its share of surprises, rebuttal by reviewer #3, all-nigthers, unrelated but fascinating side discussions and unsolvable puzzles that we loved to share anyways.

Merci au premier confinement, qui aura probablement représenté les 56 jours les plus productifs de ma thèse. Merci au second confinement, qui aura probablement représenté les 46 jours les moins productifs de ma thèse.

Il paraît que l'on ne connaît véritablement un sujet que lorsqu'on est capable de l'enseigner. J'ai eu la chance de perfectionner mon savoir dans plusieurs domaines, grâce à des interventions à l'université de Clermont, que ce soit en fac ou à l'IUT, à l'ISIMA, et plus récemment à Dauphine. Merci à mes étudiants pour leur attention et leurs questions, et pour m'avoir fait réviser mes connaissances en C, assembleur, stéganographie et autres machines de Turing, sans oublier des petites excursions dans le code pénal ou la théorie des multivers. Merci aux questions qu'ils m'ont posées, aux retours qu'ils m'ont faits et l'ingéniosité dont ils ont su faire preuve. Merci aux enseignants, Alexandre, Aurélie, David, Olivier, Michelle, Pascal, de m'avoir fait confiance, puis de m'avoir laissé donner mes propres cours. Merci également à Pint of Science et au journal des Centraliens de m'avoir offert une tribune pour y parler de sujets qui me tiennent à coeur.

Un grand merci également à tous les doctorants de la F203, que ce soit la team crypto avec Matthieu puis Léo, ou la team machine learning qu'on aime bien quand même, avec Ala, Alexandre, Amal, Kergann, et également Oliver. Merci pour toutes les discussions, les rigolades, les parties d'échecs, de badminton et de Tux kart endiablées, et tous les autres souvenirs.

Merci également à la dream team PhD qui, malgré vents et marées, a réussi à traverser l'enfer et maintenir le cap. Aurélie, Claudia, Farah, Mirko, Rose, Sami, Tom, le chemin fut plus qu'éprouvant mais le dénouement est là, ou du moins le pire est passé. Merci à Olivier d'avoir fait ce qu'il fallait pour que cette thèse aille à son terme à un moment critique. Merci aussi à Aboubakar, Arthus, Geoffrey, Pierre, Tim, vous rencontrer et travailler avec vous a été un plaisir, et à la prochaine !

Bien sûr, pour tous ces moments passés ensemble, un grand merci à mes amis, ceux de Paris, ceux de Clermont et ceux d'ailleurs, pour les discussions et trolls à n'en plus finir, les bières post-escalade (ou les escalades pré-bière, c'est selon), les escapades dans l'arrière-pays, les soirées jeux, les soirées tout court, les anniversaires, les réveillons et les sorties à la recherche de LA voie à grimper, ou tout simplement pour m'accueillir dans leur centre de bien-être et de rééducation™. Merci pour les souvenirs, pour les bonnes marrades, les moments passés ensemble, merci d'être vous.

Merci à ma famille et particulièrement à mes parents, qui durant ces trois années m'ont soutenu, encouragé, motivé, aidé à franchir les passages à vide et les doutes, qui ont toujours été un soutien pour moi et qui ont écouté avec patience (ou du moins fait semblant) toutes mes explications passionnées. Merci de toujours être là pour moi. Y por supuesto, gracias a ti Clau. Gracias por compartir tu vida conmigo y ser la luz de mi dia a dia. Ni en el mejor de mis sueños.

Contents

1 Introduction

Contents

1.1 Goal and Contributions

The first use of cryptography was for military purposes. The goal was simply to provide confidentiality to messages exchanged on what would later be labelled as insecure channels. Yet, over the last century, the need for confidentiality has often been seen as insufficient, and cryptography itself as a valuable tool, but a tool that required good properties of all the tools it interacted with. We often describe cryptography as an armoured door, preventing attackers from invading your home. However, while the armoured door itself is secure, it relies on hinges, doorframes etc., which also need to be studied for the security of the home. Furthermore, once the armoured door has been designed, it can be used in many different use cases, let it be for a vault, a simple house, the mailbox... Each new use comes with a list of requirements and properties that need to be studied. Hence, there is today a need to clearly understand how secure each software used is, and how to use them securely. As a matter of fact, software developers cannot spend days reading academic papers to assess the security of the solution they are implementing. Rather, they need to

be able to quickly consult, ideally from the software reference, how secure the use case is and how to correctly implement it.

This thesis modestly adds its contribution to the research effort in that domain: improve on the comprehension of existing data structures, and propose new use cases of cryptography with a special attention to provable security. The goal being set, our main axes of research, that we present in this manuscript, are the following:

- Study existing data structures, their security properties, and, if applicable, how they behave in an adversarial environment.
- Allow algorithms to be executed in the cloud (i.e. to be distributed), while maintaining the privacy of the data exchanged. This includes quantifying the data leaked to each participant to the protocol, as well as ensuring that an external observer cannot gain significant information from the data exchanges.

1.2 Publications

We now present our contributions. While some of our publications are presented in this manuscript, we also took the decision of not presenting all our results here, both for the sake of brevity and of coherence.

1.2.1 Security in Data Streams for Duplicate Detection

The results obtained in this section are presented in Part I. Initially, the study of duplicates was related to a new authenticated encryption method we were studying. In this protocol, we identified an attack, which we thwarted with the use of nonces-unique elements, generated by the sender. However, this defence mechanism implied that the receiver had to keep track of the nonces, to avoid reuse. While a database is an obvious solution, we looked instead for smaller structures, as our cryptographic protocol was aimed at IoT devices, with little storage and computing power.

Even though in the end the authenticated encryption appeared non viable (notably because of intrinsic properties of adversarial resistance of duplicate filters exposed in this thesis), the work on duplicate filter is still of scientific interest.

- **Quotient Hash Tables: Efficiently Detecting Duplicates in Streaming Data**

  This contribution presents the Quotient Hash Table (QHT) a new data structure for duplicate detection in unbounded streams. QHTs stem from a corrected analysis of streaming quotient filters (SQFs), resulting in a 33% reduction in memory usage for equal performance. With the same memory amount, QHT reduce by up to 7.7 points of percentage the error rate (about 13% more efficient) when compared to SQF. We also introduce an optimised version of our new data structure dubbed Queued QHT with Duplicates (QQHTD).

  We provide a new and thorough analysis of both algorithms, with results of interest to other existing constructions, and we correct a slight mistake that

was made in a previous version of this paper. We also introduce a novel (non tight) lower bound on the error rate, thus helping to determine how close to optimality a filter is. These theoretical results are confronted with detailed benchmarks, and matched against the performance of other filters from the literature, proving the efficiency of QHTs, both on an entire stream or on a sliding window. Finally, we discuss the effect of adversarial inputs for hash-based duplicate filters similar to QHT.

This work is joint with Rémi Géraud–Stewart and David Naccache. The version presented in this thesis is an extended version of a work presented at the 34th ACM/SIGAPP Symposium On Applied Computing (SAC 2019), and published as [GLPN19].

- **Approaching Optimal Duplicate Detection in a Sliding Window**

  Iterating on the previous work, we focus on the duplicate detection problem over a sliding window. In this work, we formalize the sliding window setting introduced by [SZ08; Yoo10], and show that a perfect (zero error) solution can be used up to a maximal window size $w_{\max}$. Above this threshold we show that some existing duplicate detection filters (designed for the *non-windowed* setting) perform better that those targeting the windowed problem. Finally, we introduce a "queuing construction" that improves on the performance of some duplicate detection filters in the windowed setting.

  We also analyse the security of our filters in an adversarial setting.

  This work is joint with Rémi Géraud–Stewart and David Naccache. It has been presented at the 26th International Computing and Combinatorics Conference (COCOON 2020), and published as [GSLPN20].

1.2.2 Applied Security on Blockchain

- **About Blockchain Interoperability**

  A blockchain is designed to be a self-sufficient decentralised ledger: a peer verifying the validity of past transactions only needs to download the blockchain (the ledger) and nothing else. However, it might be of interest to make two different blockchains interoperable, *i.e.*, to allow one to transmit information from one blockchain to another blockchain. In this work, we give a formalisation of this problem, and we prove that blockchain interoperability is impossible according to a strong definition of a blockchain. Under a weaker definition of blockchain, we demonstrate that two blockchains are interoperable is equivalent to creating a '2-in-1' blockchain containing both ledgers, thus limiting the theoretical interest of making interoperable blockchains in the first place. We also observe that all practical existing interoperable blockchain frameworks work indeed by exchanging already created tokens between two blockchains and not by offering the possibility to transfer tokens from one blockchain to another one, which implies a modification of the balance of total created tokens

on both blockchains. It confirms that having interoperability is only possible by creating a '2-in-1' blockchain containing both ledgers.

This work is joint with Pascal Lafourcade and has been published in Information Processing Letters Volume 161, September 2020, as [LLP20].

1.2.3 Security of a Cryptographic Protocol

These contributions are a bit more theoretical, as they spawn from a new field of research, aiming to distribute and securize existing machine learning protocols. This line of work was a collaboration with the INSA Val de Loire.

- **Secure Best Arm Identification in Multi-Armed Bandits**

  The stochastic multi-armed bandit is a classical decision making model, where an 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 action.

  A popular objective for the agent is that of identifying the arm with the maximum expected reward, also known as the *best-arm identification* problem.

  We address the inherent privacy concerns that occur in a best-arm identification problem when outsourcing the data and computations to a *honest-but-curious* cloud.

  Our main contribution is a distributed protocol that computes the best arm while guaranteeing that (i) no cloud node can learn at the same time information about the rewards and about the arms ranking, and (ii) by analyzing the messages communicated between the different cloud nodes, no information can be learned about the rewards or about the ranking. In other words, the two properties ensure that the protocol has no security single point of failure. We rely on the partially homomorphic property of the well-known Paillier's cryptosystem as a building block in our protocol.

  We prove the correctness of our protocol and we present proof-of-concept experiments suggesting its practical feasibility. This work is joint with Radu Ciucanu, Pascal Lafourcade and Marta Soare. It has been presented at the 15th International Conference on Information Security Practice and Experience (ISPEC 2019) and published as [CLLP+19].

- **Secure Outsourcing of Multi-Armed Bandits**

  We consider the problem of cumulative reward maximization in multi-armed bandits. We address the security concerns that occur when data and computations are outsourced to an honest-but-curious cloud *i.e.*, that executes tasks dutifully, but tries to gain as much information as possible. We are motivated by situations where data used in bandit algorithms is sensitive and has to be protected *e.g.*, commercial or personal data. We rely on cryptographic schemes and propose UCB-DS, a distributed secure protocol based on

the UCB algorithm, which yields the same cumulative reward as UCB while satisfying desirable security properties that we formally prove. In particular, cloud nodes cannot learn the sum of rewards for more than an arm or the cumulative reward. Moreover, by analysing messages exchanged among cloud nodes, external observers cannot learn the cumulative reward or the sum of rewards produced by some arm. We show that the overhead due to cryptographic primitives is linear in the size of the input.

Our implementation confirms the linear-time behaviour and the practical feasibility of our protocol, on both synthetic and real-world data.

This work is joint with Radu Ciucanu, Pascal Lafourcade and Marta Soare. It has been presented at the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2020), and published as [CLLP+20].

1.2.4 Other Publications

Other publications have been made throughout the last three years, on various topics of security, sometimes theoretical, sometimes practical.

- **Keyed Non-Parametric Hypothesis Tests**

  The recent popularity of machine learning calls for a deeper understanding of AI security. Amongst the numerous AI threats published so far, poisoning attacks currently attract considerable attention. In a poisoning attack the opponent partially tampers the dataset used for learning to mislead the classifier during the testing phase.

  This work proposes a new protection strategy against poisoning attacks. The technique relies on a new primitive called keyed non-parametric hypothesis tests allowing to evaluate under adversarial conditions the training input's conformance with a previously learned distribution $\mathfrak{D}$. To do so we use a secret key κ unknown to the opponent.

  Keyed non-parametric hypothesis tests differs from classical tests in that the secrecy of κ prevents the opponent from misleading the keyed test into concluding that a (significantly) tampered dataset belongs to $\mathfrak{D}$.

  This work is joint with Yao Cheng, Cheng-Kang Chu, Hsiao-Ying Lin and David Naccache, and has been presented at the the 13th Network and System Security (NSS 2019), and published as [CCL+19].

- **PlasticCoin: an ERC20 Implementation on Hyperledger Fabric for Circular Economy and Plastic Reuse**

  Cryptocurrencies have gained popularity in the last few years, thanks to the democratization of Bitcoin. However, most of these currencies have a very general purpose, namely of allowing people to pay their products with an

electronic, decentralised currency. Moreover, the lack of trust of these open-network blockchains comes at a significant economic cost. In this work, we present PlasticCoin, a cryptocurrency empowering plastic reuse in a circular economical model, that anyone can join, and that respects the ERC20 specifications on a consortium blockchain. We also explore the economical ecosystem revolving around PlasticCoin, and also introduce a way to print tickets that can temporarily hold the role of physical banknotes. Finally, we show that our system is flexible, and it can be adapted to many other business purposes.

This work is joint with Mirko Koscina and Pierre Cluchet. It has been presented at the Workshop On Social Innovation And Web Intelligence, hosted by the Web Intelligence conference (SIWEB@WI 2019), and has been published as [KLC19].

- **Get-your-ID: Decentralized Proof of Identity**

  In most systems without a centralised authority, users are free to create as many accounts as they please, without any harmful effect on the system. However, in the case of e-voting, for instance, proof of identity is crucial, as sybil identities can be used to breach the intended role of the system. We explore the conditions under which a decentralised proof of identity system can exist. We also propose such a scheme, called Get-your-ID (GYID), and prove its security. Our system allows a user to generate and revoke keys, via an endorsement mechanism, and we prove that under some conditions which we discuss, no user can have more than one active key. We then show how voting protocols can be adapted on top of our system, thus ensuring that no user is able to cast a valid vote more than once.

  This work is joint with Pascal Lafourcade. It has been presented at the 12th International Symposium on Foundations and Practices of Security (FPS 2019), and has been published as [LL19].

- **A silver bullet? A Comparison of Accountants and Developers Mental Models in the Raise of Blockchain**

  This contribution is quite different from the others, as it is a preliminary research on an interdisciplinary research between requirement engineering and security. This exploratory work intends to drive preliminary insights on the different mental models accountants and blockchain developers have on the implementation of blockchain for accounting. Based on the question of whether blockchain applications for accounting could be revolutionary, this work employs a ground theory methodology based on semi-structured interviews and concept analysis to highlight the different approaches to transparency and trust between the selected groups, the challenges of blockchain and the potential effects of this technology in accounting. Although deeper studies are needed, the conclusions highlight the socio-technical nature of accounting; the relevance and changes of the concepts of trust and transparency when marrying both disciplines; and the real relevance of this technology for the processes

of auditing and accounting.

This work is joint with Rose Esmander, Pascal Lafourcade, Claudia Negri Ribalta. It has been presented at the 3rd Interdisciplinary Workshop on Privacy and Trust (iPAT 2020), held in conjunction with the 15th International Conference on Availability, Reliability and Security (ARES 2020), and published as [ELL+20]. An extended version of this work has been published in a selected papers edition of ARES, as [NRLS+21].

- **A Blockchain-based Marketplace Platform for Circular Economy**

  Over the recent years, there has been an intense research on blockchain and its applications in real-life scenarios, both from academic and industrial research. This work proposes an architecture for a blockchain-based marketplace platform. We have implemented and tested our design, which allows user to do simple marketplace transactions as well as more complicated features, such as auctions à la eBay. We explore the functionalities and the implementation details of our solution, while showing how we ensure respect with GDPR.

  This work is joint with Mirko Koscina and Claudia Negri Ribalta, and has been accepted at the 36th ACM/SIGAPP Symposium On Applied Computing (SAC 2021), and has been published as [KLNR21].

2

Technical Introduction

Contents

2.1 Duplicate detection

In this section, we expose the formalism and the definitions related to duplicate detection. Duplicate detection relates to the problem, when given a stream of elements, to detect for each new element whether it occurred previously in the stream or not.

2.1.1 Duplicate Detection Problem

Definition 2.1.1 (Duplicate detection problem)**.** *Let $E = (e_1, \dots, e_n, \dots)$ be a (possibly infinite) sequence of elements $e_i \in \Gamma$ where Γ is some alphabet of size $|\Gamma|$.*

An element e_i from E is a duplicate *if $\exists e_j \in E, j < i$ such as $e_j = e_i$. Otherwise, e_i is* unseen*. Note that by definition e_1 is always unseen.*

The approximate duplicate detection problem *is the question of finding with the best probability, for* each *new element of the stream, whether it is a duplicate in a stream E.*

This problem is equivalent to a dynamic formulation of the approximate membership problem [DP08], which focuses on finding a duplicate in a *fixed* dataset. We recall the following classical result:

Theorem 2.1.1. *Assume that each e_i is sampled uniformly at random from Γ. Then perfect detection requires $|\Gamma|$ memory bits.*

Proof. A perfect duplicate algorithm must be able to store all finite substreams of any stream $E = (e_1, \dots, e_n, \dots)$, *i.e.*, must be able to store any subset of Γ, which we denote by $\mathcal{P}(\Gamma)$. Given that there are $|\mathcal{P}(\Gamma)| = 2^{|\Gamma|}$ of them, according to information theory any such filter requires at least $\log_2(|\mathcal{P}(\Gamma)|) = \log_2(2^{|\Gamma|}) = |\Gamma|$ bits of storage. ■

Because of this result, perfect duplicate detection is often out of reach when $|\Gamma|$ is big — however probabilistic solutions are often sufficient for many applications. Such algorithms make errors: false positives (claiming a duplicate where there isn't) and false negative (missing a duplicate). We mostly consider cases when the amount of memory M is such that $M \ll |\Gamma|$.

On a side note, it is clear that the input distribution plays a central role regarding how efficiently the duplicate detection problem can be solved. For instance, some deterministic streams may be expressed very compactly (such as the output of a PRNG with known seed) making the duplicate detection problem relatively easy. Information-theoretically, if the source has $|\Gamma|$ bits of entropy then the situation is equivalent to having an $|\Gamma|$-bits, uniformly distributed, input.

2.1.2 Filters

As the problem of duplicate detection has been set, we now need to define the structures that will try to solve this problem.

Definition 2.1.2 (Duplicate Detection Filter (DDF))**.** *Let Γ be an alphabet, let $\mathcal{M}$ represent the set of states that can be represented using M bits of memory. A duplicate detection filter $\mathcal{F}$ (or DDF) over the memory M is a tuple $\mathcal{F} = (\mathcal{S}, \mathsf{Detect}, \mathsf{Insert})$, where:*

- $\mathcal{S} \in \mathcal{M}$ *is the* current state
- $\mathsf{Detect} : \Gamma \times \mathcal{M} \to \{\mathit{DUPLICATE}, \mathit{UNSEEN}\}$

- $\mathit{Insert} : \Gamma \times \mathcal{M} \to \mathcal{M}$

Here DUPLICATE corresponds to a guess, given the current filter state, that the provided element is duplicate, and UNSEEN that it is unseen. Insert corresponds to an update of the filter's memory state after observing a new element.

In practice, Detect and Insert are often merged into a single algorithm Stream $\leftarrow$ Insert $\circ$ Detect: when a new stream element arrives, we first detect whether it is a duplicate or not, and then add it in our memory.

Definition 2.1.3 (False positive (resp. negative))**.** *If, for an unseen (resp. duplicate) element e, Detect(e) outputs DUPLICATE (resp. UNSEEN), e is called a* false positive *(resp.* negative*).*

Definition 2.1.4 (FPR, FNR)**.** *The* false positive rate *(*FPR*) of a stream E is the frequency of false positive. The* false negative rate *is similarly defined as the frequency of false negatives.*

Furthermore, for a stream E, we define the value FPR_n *as the FPR of E over the n first elements, and same for* FNR_n*. Similarly, the asymptotic FPR (resp. FNR) is defined as* $\mathrm{FPR}_\infty = \lim_{n\to\infty} \mathrm{FPR}_n$ *(resp.* $\mathrm{FNR}_\infty = \lim_{n\to\infty} \mathrm{FNR}_n$*).*

Note that, at several occasions, we will focus on the probability of false positive (resp. false negative) after n insertions, noted FP_n (resp. FN_n), as these probabilities are most of the time easier to derive. The derivation of, *e.g.*, the FPR from the FP probability is trivial: $\mathrm{FPR}_m = \frac{1}{m}\sum_{i=1}^{m} \mathrm{FP}_i$.

An open research question is whether there exist filters whose FNR and FPR can be kept low when M is bounded.

2.1.3 Duplicate Detection on a Sliding Window

As we will see, the DDP is often more interesting to study on a sliding window.

Let us consider a sliding window of size $w \in \mathbb{N}$, a stream $E = (e_1, \dots, e_n, \dots)$ from an alphabet Γ.

Definition 2.1.5 (Duplicate detection problem on a sliding window (wDDP))**.** *The element e_i of E is said to be unseen in the sliding window, and we note $e_i \notin_w E$ if all w previous elements from E are different from e_i. More formally, $\forall i \leq n, e_i$ is unseen if and only if $\forall k \in \{1, \dots, \min(w, i-1)\}, e_{i-k} \neq e_i$. If this is not the case, e_i is said to be a duplicate within the sliding window, and we note $e_i \in_w E$.*

We similarly redefine the notion of false positives and negatives, as well as the false positive rate and false negative rate, over a sliding window. The FPR (resp. FNR) of a filter on a sliding window of size w is noted FPR^w (resp. FNR^w).

While the goal of this work is to work as much as possible with this definition of false positives on a sliding window, we sometimes use the error rate following Definition 2.1.3, as the related formulae are much easier to use for practical considerations.

2.1.4 Adversarial Resistance to Duplicate Detection

In this section, we consider a scenario in which an adversary tries to fool the filter. Namely, the adversary will try to trigger false positives and false negatives. Motivations for doing so include fooling the system, for instance in the case of fraudulent clicks transactions [MAEA05], a sophisticated DoS relying on cache misses [FCA+00], and so on.

To create a realistic adversary model, we assume like in [CDPLB+17] that the adversary does not have access to the filter's internal memory. Nonetheless, after every insertion she knows whether the inserted element was detected as a duplicate or not. Thus, the attacker is able to carry an adaptive attack, by choosing the next element to send to the filter as a function of all previous insertions. The adaptative advantage allows the attacker to establish strategies that may perform better than random, hence giving an advantage to the attacker in her endeavour, whatever this endeavour might be.

We do not make assumption on the adversary memory: she is able to remember all previous queries she made.

In this adversarial game, the attacker can send an arbitrary stream to the filter, and is allowed to get the result of Stream for every element. Then, at her convenience, the attacker goes into the second phase of the game, in which she has two possible actions:

- Send an unseen element that will be a false positive with high probability (false positive attack);
- Send a duplicate element that will be a false negative with high probability (false negative attack).

Given these premises, we give a formal definition of an adversarial game, adapted to our context.

Definition 2.1.6 (n-false positive/negative adversarial game)**.** *An adversary $\mathcal{A}$ feeds data to a duplicate filter $\mathcal{F}$, and for each inserted element, $\mathcal{A}$ knows whether $\mathcal{F}$ answers DUPLICATE or UNSEEN, but has not access to $\mathcal{F}$'s internal state $\mathcal{M}$. The game has two distinct parts.*

- *In the first part, $\mathcal{A}$ can feed up to n elements to $\mathcal{F}$ and learns $\mathcal{F}$'s response for each insertion. $\mathcal{A}$ can decide to switch to the second part before sending n elements.*
- *In the second part, $\mathcal{A}$ sends a unique element $e^\star$.*

$\mathcal{A}$ wins the n-false positive adversarial game (resp. n-false negative adversarial game) if and only if $e^\star$ is a false positive (resp. a false negative).

Variants of these games over a sliding window of size w are immediate. Note that other works (notably [NY15]) have also investigated the adversarial context of some duplicate filters, in this case, of the Bloom Filter. However, they relied

on a different definition of adversarial attacks, in which the attacker had access to an oracle, emulating the filter's state (which does not evolve), and then emits a challenge $e^\star$. While their definition of an adversarial attack is more fit in the context of a fixed Bloom Filter (the context [NY15] authors investigate), we argue that our definition is more fit in a streaming context.

Definition 2.1.7 (Adversarial False Positive Resistance)**.** *We say that a duplicate filter $\mathcal{F}$ is (p, n)*-resistant to adversarial false positives *if no polynomial-time probabilistic (PPT) adversary $\mathcal{A}$ can win the $n-$false positive adversarial game with probability greater than p.*

Note that if $\mathcal{F}$ is (p, n)-resistant, then it is (p, m)-resistant for all $m < n$.

We define similarly the notion of being *resistant to adversarial false negatives*. Finally, both definitions also make sense in a sliding window context.

2.2 Primer on Multi-Armed Bandits

The problem of *best arm identification in multi-armed bandits* [ABM10] has been initially formulated in the domain of real numbers. We slightly revisit the initial formulation of the problem in order to manipulate integers. The reason behind this adaptation is that later on in our thesis, we rely on public key cryptography tools to add security guarantees to a state-of-the-art best arm identification algorithm.

2.2.1 Problem exposure

Input. The input is twofold:

- **Number of arms** K. Each arm $i \in \{1, \ldots, K\}$ is associated to a reward value $x(i)$ and a reward function r that returns a random integer in an interval $[x(i) - \epsilon, x(i) + \epsilon]$ according to a uniform probability distribution. Whereas each arm i is associated to its specific value $x(i)$, the value of ϵ is common to all arms. The intervals associated to different arms may be overlapping, which makes the setting non-trivial. The best arm i^* is $\arg\max_{i \in \{1,\ldots,K\}} x(i)$.
- **Budget** N. The budget represents how many arm pulls (and implicit reward observations) the user is allowed to do.

Note that for designing a budget-allocation strategy between the arms, only the number of arms K and the budget N are known. There is no initial information about the reward associated to each arm.

Output. The estimated best arm $\widehat{i^*}_\alpha$ that can be learned after making N arm pulls (and subsequent reward observations) according to some allocation strategy α, which defines how the budget is divided between the K arms. The challenge is to design a budget-allocation strategy α that makes the best possible use of the budget. In other words, when selecting the arms to be pulled according to α, the observed rewards allow to acquire as much useful information as possible for identifying i^*.

Performance Measure. We call *simple regret* R_N the difference between the value of the (true) best arm i^* and the arm $\widehat{i^*}_\alpha$ estimated as being the best arm by an allocation strategy α after N arm pulls. Thus, we compare the gap between the value of the identification made by strategy α and that of an *oracle* strategy that knows the values of the arms beforehand. Formally, the performance of strategy α after using a budget N is $R_N(\alpha) = x(i^*) - x(\widehat{i^*_\alpha})$.

Example. We have 3 arms with associated reward values in intervals [3, 23], [25, 45], and [40, 60]. This means that $x(1)$=13, $x(2) = 35$, $x(3) = 50$, and ϵ=10. Assuming a budget of 3, the user may choose to spend one pull for each arm and observe rewards of (for instance) 23, 44, and 41, respectively. Hence, the user could wrongly think that arm 2 is the best, thus getting a regret of $50 - 35 = 15$.

Obviously, increasing the budget would increase the number of pulls that can be done, hence it would increase the chances of correctly identifying the best arm. This can be easily done in the presence of an infinite budget, but the challenge is to identify the best arm using as few pulls as possible, or in other words, to maximize the probability of correctly identifying the best arm while having a limited budget.

2.2.2 Successive Rejects (SR) [ABM10]

The algorithm takes as input the number of arms K and the budget N. Initially, all K arms are candidates. SR divides the budget in $K - 1$ phases. At the end of each phase, a decision is made. The phases' lengths are fixed such that the available budget is not exceeded and the probability of wrongly identifying the best arm is minimized.

More precisely, at each phase $j \in \{1, \ldots, K-1\}$, each still candidate arm in A_j is pulled n_j times according to the fixed allocation (cf. Algorithm 1). At the end of each phase, the algorithm rejects the arm with the lowest sum of observed rewards, that is the arm estimated to be the worst. If there is a tie, SR randomly selects the arm to reject among the worst arms. Then, at the next phase, the remaining arms are again uniformly pulled according to the fixed allocation. Thus, the worst arm is pulled n_1 times, the second worst is pulled $n_2 + n_1$ times, and so on, with the best and the second-best arm being pulled $n_{K-1} + \ldots + n_1$ times. The estimated best arm is the unique arm remaining after phase $K - 1$.

We consider the sums of observed rewards per arm when deciding which arm to reject instead of empirical means as in the original version [ABM10] as a simplification. Indeed, each candidate arm is pulled the same number of times in each phase, hence the ranking of the arms is identical regardless of whether we look at sums or means.

Example. Let there be a multi-armed bandit with 4 arms and $x(1) > x(2) > x(3) > x(4)$, with budget $N = 500$ pulls. We have $\overline{log}(4) = \frac{1}{2} + \sum_{i=2}^{4} \frac{1}{i} = \frac{19}{12}$ and:

Phase 1: each arm 1, 2, 3, 4 is pulled $n_1 = \lceil \frac{12}{19} \frac{500-4}{4+1-1} \rceil = 79$ times

Algorithm 1 SR algorithm (adapted from [ABM10])

$A_1 \leftarrow \{1, \dots, K\}$ ▷ *Initialization*
for all $i \in A_1$ **do**
  $sum[i] \leftarrow 0$
$\overline{\log}(K) \leftarrow \frac{1}{2} + \sum_{i=2}^{K} \frac{1}{i}$
$n_0 \leftarrow 0$

for j from 1 to $K-1$ **do** ▷ *Successive rejects*
  $n_j \leftarrow \left\lceil \frac{1}{\overline{\log}(K)} \frac{N-K}{K+1-j} \right\rceil - \sum_{l=0}^{j-1} n_l$
  for all $i \in A_j$ **do**
    loop n_j times
      $r \leftarrow$ random integer from $[x(i) - \epsilon, x(i) + \epsilon]$
      $sum[i] \leftarrow sum[i] + r$
  $A_{j+1} \leftarrow A_i \setminus \arg\min_{i \in A_j} sum[i]$
return A_K

Phase 2: each arm 1, 2, 3 is pulled $n_2 = \lceil \frac{12}{19} \frac{500-4}{4+1-2} \rceil - n_1 = 26$ times

Phase 3: each arm 1, 2 is pulled $n_3 = \lceil \frac{12}{19} \frac{500-4}{4+1-3} \rceil - (n_1 + n_2) = 52$ times.

In other words, arm 4 is pulled 79 times, arm 3 is pulled 79+26=105 times, each arm 1, 2 is pulled 79+26+52=157 times, totalling $79 + 105 + 2 \times 157 = 498$ pulls.

2.2.3 A Variation: Upper Confidence Bound (UCB)

Instead of identifying the best arm, one can be interested into exploring similar questions. One of these questions is the maximisation of the cumulative reward, in which the player will try and maximise their gains while playing on a limited budget. Of course, the asymptotic result will be the same, as the playing whishing to optimize their gains will choose the best arm, however when the rewards functions are unkown the behaviour might be completely different.

UCB is a class of algorithms commonly used when facing the exploration-exploitation dilemma. Each bandit arm is associated with a distribution whose mean is a priori unknown to the learning agent. When pulling an arm, the learning agent observes an independent reward drawn from the distribution of the chosen arm. The goal of the agent is to maximize the sum of observed rewards. To guide the choice of the learner, arm scores have been proposed [Agr95] to construct *upper confidence bounds* (UCB) based on the empirical mean of arm-specific rewards and the number of arm pulls. In the UCB class of algorithms, an important breakthrough was the introduction of algorithms with a finite-time analysis [ACBF02]. Specifically, in the UCB1 algorithm [ACBF02] that we present in Algorithm 2, the score B_i can be interpreted as the largest statistically plausible mean value of arm i, given the

Algorithm 2 UCB Algorithm [ACBF02]

Input: Budget N, number of arms K

Unknown environment: $\mu_1, \ldots, \mu_K$ that are the expected values of K Bernoulli distributions associated to the K arms *i.e.*, $\forall 1 \leq i \leq K$ the probability of 1 is μ_i and the probability of 0 is 1–μ_i. The learning agent has access only to the output of a reward function $pull(.)$, where $pull(i)$ randomly returns 0 or 1 according to the associated Bernoulli distribution.

Output: Sum of observed rewards for all arms

function UCB(N, K)
  ▷ *Initialization phase: pull each arm once and initialize variables*
  for $1 \leq i \leq K$ **do**
    $r \leftarrow pull(i)$ ▷ *Random reward for arm i*
    $s_i \leftarrow r$ ▷ *Sum of observed rewards for arm i*
    $n_i \leftarrow 1$ ▷ *Number of times the arm i has been pulled*
    $B_i \leftarrow \frac{s_i}{n_i} + \sqrt{\frac{2\ln(K)}{n_i}}$ ▷ *First term for exploitation and the second for exploration*
  ▷ *Exploration-exploitation phase: pull one selected arm at each round t*
  for $K+1 \leq t \leq N$ **do** ▷ *Only a budget of $N-K$ is left*
    $i_m \leftarrow \arg\max_{1 \leq i \leq K}(B_i)$ ▷ *i_m is the arm with maximum B value*
    $r \leftarrow pull(i_m)$ ▷ *Pull arm i_m and update the corresponding variables*
    $s_{i_m} \leftarrow s_{i_m} + r$
    $n_{i_m} \leftarrow n_{i_m} + 1$
    for $1 \leq i \leq K$ **do** ▷ *Update B values for all arms because t changes at each iteration*
      $B_i \leftarrow \frac{s_i}{n_i} + \sqrt{\frac{2\ln(t)}{n_i}}$ ▷ *B_i is an upper confidence bound on μ_i*
  return $s_1 + \ldots + s_K$ ▷ *Return sum of observed rewards for all arms*

observed rewards. After each observation, the scores for all arms are updated. As shown in the pseudo-code in Algorithm 2, the UCB algorithm follows the principle of *optimism in the face of uncertainty* and chooses to pull next the arm with the largest updated B_i score. This principle suggests to follow what seems to be the best arm, based on the *optimistically* constructed scores. The same principle is employed in various sequential decision making problems (see [Mun14] for a survey). Note that an arm i might have the largest score due to its empirical mean (the *exploitation* term) and/or to the uncertainty one has about the true arm value (the *exploration* term), directly dependent on how many rewards from arm i were observed.

2.3 Cryptography

2.3.1 Asymmetric cryptography

Definition 2.3.1 (PKE)**.** *Let λ be a security parameter. A* public-key encryption *(PKE) scheme is defined by $(\mathcal{G}, \mathcal{E}, \mathcal{D})$:*

- $\mathcal{G}(1^\lambda)$ *returns a public/private key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathcal{E}(\mathsf{pk}, m)$ *returns a ciphertext* c.
- $\mathcal{D}(\mathsf{sk}, c)$ *returns with overwhelming probability the plaintext* m*, for* $c = \mathcal{E}(\mathsf{pk}, m)$.

Definition 2.3.2 (Paillier encryption scheme)**.** *We denote by $\mathbb{Z}_n$, the ring of integers modulo n and by $\mathbb{Z}_n^\times$ the set of invertible elements of $\mathbb{Z}_n$.*

The Paillier cryptosystem is a public key encryption scheme where:

- $\mathcal{G}(1^\lambda)$ *generates two prime numbers p and q according to λ, sets $n = p \cdot q$ and $\Lambda = \mathrm{lcm}(p-1, q-1)$ (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 $\mathsf{sk} = (\Lambda, M)$, $\mathsf{pk} = (n, g)$, and returns $(\mathsf{sk}, \mathsf{pk})$.*
- $\mathcal{E}(\mathsf{pk}, m)$ *randomly picks $r \in \mathbb{Z}_n^*$, computes $c = g^m \cdot r^n \mod n^2$, and outputs c.*
- $\mathcal{D}(\mathsf{sk}, c)$ *computes $m = L(c^\Lambda \mod n^2) \cdot M \mod n$, and outputs m.*

2.3.2 Partially Homomorphic Scheme

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

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

We also remark that: $\mathcal{E}(\mathsf{pk}, m_1) \cdot \mathcal{E}(\mathsf{pk}, m_2)^{-1} = \mathcal{E}(\mathsf{pk}, m_1 - m_2)$.

2.3.3 AES-CBC symmetric encryption

AES [NIS01] is a NIST standard for symmetric encryption that encrypts messages of 128 bits. AES is used as a block cipher, for instance using CBC mode (Cipher Block Chaining).

Definition 2.3.3. *AES-CBC The AES-CBC cryptosystem is a symmetric encryption scheme defined by a triple of polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *and a security parameter* λ *such that:*

- $\mathsf{KeyGen}(1^\lambda)$ *generates* Key*, a uniformly random symmetric key of* 128*,* 192 *or* 256 *bits, according to* λ*.*
- $\mathsf{Enc}(\mathsf{Key}, m, IV)$ *splits* m *in blocks of* 128 *bits* $m_0, \ldots, m_n$ *(padding bits may be added if* m_n *is smaller than 128 bits).* Enc *computes* $c_0 = \mathsf{E}(\mathsf{Key}, m_0 \oplus IV)$*, where* E *is the AES encryption [NIS01] and* IV *is a random 128-bits number. By* $x \oplus y$ *we denote the standard bit-wise xor operation between two bit strings* x *and* y*. Then,* Enc *computes* $c_i = \mathsf{E}(\mathsf{Key}, c_{i-1} \oplus m_i)$ *for* $1 \leq i \leq n$ *and returns the tuple* $((c_0, \ldots, c_n), IV)$*.*
- $\mathsf{Dec}(\mathsf{Key}, c, IV)$ *splits* c *in blocks of 128 bits* $c_0, \ldots, c_n$ *and computes* $m_0 = \mathsf{D}(\mathsf{Key}, c_0) \oplus IV$*, where* D *is the AES decryption [NIS01]. Similarly,* Dec *computes* $m_i = \mathsf{D}(\mathsf{Key}, c_i) \oplus c_{i-1}$ *for* $1 \leq i \leq n$ *and returns* $m_0, \ldots, m_n$*.*

2.3.4 IND-CPA

We first recall the notion of negligible function in order to define the IND-CPA security notion.

Definition 2.3.4. *A function* $\gamma \colon \mathbb{N} \to \mathbb{R}$ *is negligible in* λ*, and is noted* $negl(\lambda)$*, if* $\forall c > 0$*,* $\exists \lambda_0, \forall \lambda > \lambda_0$*,* $\gamma(\lambda) < \lambda^{-c}$*.*

We now define how a *probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ tries to break the security of Π.

Definition 2.3.5 (IND-CPA game)**.** *Let* $\Pi =$ (KeyGen, Encrypt, Decrypt) *be a cryptographic scheme. The IND-CPA game, denoted by* $\mathrm{EXP}(\mathcal{A})$*, works as follows: the adversary* $\mathcal{A}$ *chooses two messages* (m_0, m_1) *and receives a challenge* $c = \mathsf{Encrypt}(LR_b(m_0, m_1))$ *from the* challenger *who selects a bit* $b \in \{0, 1\}$ *uniformly at random, and where* $LR_b(m_0, m_1)$ *is equal to* m_0 *if* b=0*, and* m_1 *otherwise.*

The adversary, knowing m_0, m_1 *and* c*, is allowed to perform any number of polynomial computations or encryptions of any messages, using the encryption oracle, in order to output a guess* b' *of the encrypted message in* c *chosen by the challenger.*

Intuitively, Π is IND-CPA secure if there is no PPT adversary who can guess b with a probability significantly better than $\frac{1}{2}$. By $\alpha = \Pr[b' \leftarrow \mathrm{EXP}(\mathcal{A}); b = b']$, we denote the probability that $\mathcal{A}$ correctly outputs her guessed bit b' when the bit chosen by the challenger in the experiment is b.

Definition 2.3.6 (IND-CPA security)**.** *Let* Π *be a PKE scheme. Let* $\alpha = \Pr[b' \leftarrow \mathrm{EXP}(\mathcal{A}); b = b']$ *be the probability of success of a PPT adversary in an an adversarial game on* Π*.*

Π *is IND-CPA secure if* $\alpha - \frac{1}{2} = negl(\lambda)$*.*

Both cryptographic schemes mentioned earlier in this section are IND-CPA:

1. Paillier is IND-CPA under the decisional composite residuosity assumption [Pai99],
2. AES-CBC is IND-CPA under the assumption that AES is a pseudo-random permutation [BDJ+97].

Part I

Security in Data Streams for Duplicate Detection

Part I

Security in Data Streams for Duplicate Detection

3

Quotient Hash Tables

Contents

Abstract

This chapter presents the Quotient Hash Table (QHT) a new data structure for duplicate detection in unbounded streams. QHTs stem from a corrected analysis of streaming quotient filters (SQFs), resulting in a 33% reduction in memory usage for equal performance. With the same memory amount, QHT reduce by up to 7.7 points of percentage the error rate (about 13% more efficient) when compared to SQF. We also introduce an optimised version of our new data structure dubbed Queued QHT with Duplicates (QQHTD).

We provide a new and thorough analysis of both algorithms, with results of interest to other existing constructions, and we correct a slight mistake that was made in a previous version of this work.

We also introduce a novel (non tight) lower bound on the error rate, thus helping to determine how close to optimality a filter is.

These theoretical results are confronted with detailed benchmarks, and matched against the performance of other filters from the literature, proving the efficiency of QHTs, both on an entire stream or on a sliding window.

Finally we discuss the effect of adversarial inputs for hash-based duplicate filters similar to QHT.

This work is joint with Rémi Géraud–Stewart and David Naccache. This is an extended version of a work presented at the 34th ACM/SIGAPP Symposium On Applied Computing, SAC 2019, and published as [GLPN19].

3.1 Introduction and preliminaries

Amongst other possible attacks on a protocol, there exists a very simple one, called '*replay attack*'. In this scenario, an attacker intercepts a valid transaction on the network, and then sends the exact same message, pretending to be a valid user [DS81]. Similarly, *oracle attacks* are models in which the attacker can deduce information about a hidden information, by for instance repeatedly sending invalid data, until the oracle accepts the input. Such attacks have been carried on SSL, IPSEC [Vau02], and have even led to password interception over SSL/TLS channels [CHV+03].

One solution for blocking these attacks consists in the use of a nonce, a unique, cryptographically random, number. The nonce guarantees the 'freshness' of the message, as the receiver discards all messages using previous nonces. In this chapter,

we discuss about how one can store nonces, and detect duplicates, especially when available memory is low.

This problem has been studied in the literature as the *approximate membership problem* [DP08], or the *duplicate detection problem* [EIV07], while the first paper on the topic was written by Bloom [Blo70].

Note that the duplicate detection problem is not only useful in security, but also in many other fields. For instance, the same problem arises naturally in many applications: backup systems [FFH+15] or Web caches [FCA+00], search engine databases or click counting in web advertisement [MAEA05], retrieval algorithms [BRJ+14], data stream management systems [BDM04]. In each of these usecases, duplicate detection can lead to increased performance (a cache able to detect if a page is indeed cached), increased efficiency (a backup system will not backup twice the same file, a web crawler won't scan twice the same page), or increased security (duplicate clicks can be interpreted as fraud).

In many cases, it is generally not possible to store the whole stream in memory, therefore practical solutions to this problem must somehow trade off memory for accuracy. The special case of finding one single duplicate in a given –fixed– dataset has a known optimal solution [JST10; KNP+17], but to the best of our knowledge no such results exists for detecting *all* (or most) duplicates in an unbounded stream.

3.2 Related Work

3.2.1 Hash-based Filters

Filters rely on hashing to efficiently answer the duplicate detection problem. These filters are often a variation over the well-known Bloom filters, introduced in [Blo70]. A Bloom filter uses an array T of M bits (usually M being a power of 2), initially all set to 0. k hash functions h_i, over the range $[0, M-1)$, are also needed. Insertion of an element e is made by setting all $T[h_i(e)]$ to 1. Detection of an element f is the logical AND of all cells $T[h_i(f)]$: if the AND value is 0 (*i.e.*, if at least one bit $T[h_i(f)]$ is equal to 0), then emit UNSEEN, otherwise emit DUPLICATE. An illustration is depicted in Figure 3.1.

It is easy to see that this approach has an FNR of 0. However, as the stream grows, the FPR gets worse, and in the limit of an infinite stream the FPR is 1. Many variants have been proposed in the literature to compensate for this effect [TRL12], mostly by allowing deletion [CM03; CM05], but doing so increases the FNR.

Alternatively, other filters prefer to store fingerprints: this is the rationale behind several recent constructions, such as [DNB13; FAK+14].

The data structures most interesting to our question are the following:

- Stable Bloom Filters (SBF) [DD06]
- Streaming Quotient Filter (SQF) [DNB13]
- Cuckoo Filter [FAK+14]

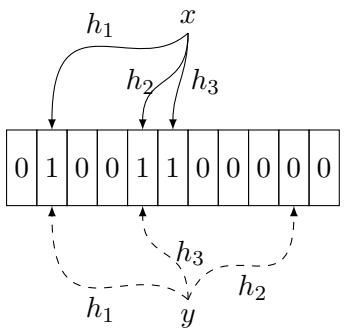


Figure 3.1: Illustration of a Bloom filter with 10 cells and 3 hash functions. The element x is added to the filter, so the 3 corresponding bits of index $h_i(x)$ (not necessarily distinct) are set to 1. When querying for y, the filter returns UNSEEN as one of the hashes $h_i(y)$ points to a null bit.

- Block Decaying Bloom Filter (b_DBF) [SZ08]
- A2 Filter [Yoo10]

Cuckoo filters [FAK+14] requires a minimal adaptation for unbounded streams: in the original paper, failure is emitted after some number of relocations; we just discard the failure. Theoretical analysis of this new structure is not addressed in this chapter.

Structures from [CLJ+17; GWC+10] are not considered in this chapter, as they require an unbounded amount of memory.

Note that, for analysis purposes, all these papers consider hash functions as pseudo-random functions. We make the same assumption.

3.2.2 Streaming Quotient Filter

One construction which we must describe at length is the Streaming Quotient Filter (SQFs) [DNB13]. Given an element e, a certain fingerprint[1] $s(e)$ is stored in an array, at row $h(e)$ and a certain column amongst k. This array constitutes the filter's state.

The filter's construction uses integers q, k, r, $r' < r$ and a hash function $h : \{0,1\}^* \to \{0,1\}^{q+r}$. The filter's state is an array of 2^q rows and k columns, each holding a σ-bit element (with $\sigma = r' + \lceil \log_2(r+1) \rceil$), or the special empty symbol $\perp$. The filter's state is initially $\perp$ in every cell.

Then, [DNB13] describes Stream(e) as follows:

- Compute $h(e)$. Its q most significant bits define $h_q(e)$, and the r least significant bits define $h_r(e)$.

[1] [DNB13] refers to them as "signatures", but we shun this term to avoid any claim of cryptographic properties.

- Let $\omega(h_r(e))$ be the Hamming weight of $h_r(e)$, and let $h_{r'}(e)$ be r' bits deterministically chosen from $h_r(e)$, *e.g.*, the r' least significant bits of $h_r(e)$. Let $s(e) \leftarrow h_{r'}(e) \| \omega(h_r(e))$ where $\|$ denotes concatenation.
- If $s(e)$ is already stored in the row numbered $h_q(e)$, emit DUPLICATE.
- Otherwise, store $s(e)$ in one empty cell of that row. If no empty cell exists in the row, store $s(e)$ in one *random* cell of the row, replacing any fingerprint previously stored there. Emit UNSEEN.

For instance, let us take $q = 5, r = 4, r' = 2$. Take some element e, assume that $h(e) = 101110100$, we get $h_q(e) = 10111, h_r(e) = 0100$. Taking the r' least significant bits for $h_{r'}(e)$, we get $h_{r'}(e) = 00$. We also have $\omega(h_r(e)) = 001$, so e's fingerprint is 00001 and will be stored in the row numbered 23 (as $23 = 0b10111$).

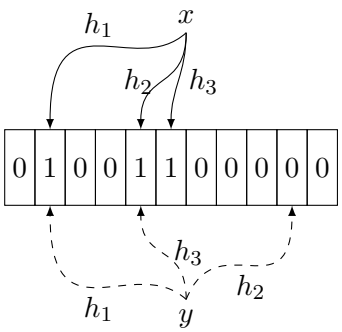


Figure 3.2: Illustration of the SQF. For an unseen element e, its fingerprint $s(e_n) = h_{r'}(e_n) \| \omega(h_r(e_n))$ is inserted in the row $T[h_q(e_n)]$, in an empty cell. In this example, $h_q(e_n) = 2^q - 1$. If there is no such empty cell in $T[h_q(e_n)]$, the fingerprint is stored in a random cell of the row.

3.3 Contributions and Organisation

In this chapter, we introduce a new structure, the QHT (Quotient Hash Tables), which stems from an improvement of an existing filter, the SQF [DNB13].

We first explore in Section 3.4 the limitations of the 'naive' duplicate detection problem, and formulate some lower bounds on filter saturation in the case of an infinite sliding window.

Then, we analyse the SQF design before correcting its errors, thus creating the QHT, in Section 3.5, and provide a correction of SQF error rate. We also provide a formal description of the QHT false positive and false negative rates in Section 3.6, which match the predictions made by Theorem 3.4.1 from Section 3.4.1.

We then compare QHT to the literature. In Section 3.6.3, we show that for the same efficiency, a QHT requires 33% less memory than an SQF in optimal settings.

In Section 3.7 we introduce heuristic improvements to QHT, describing a structure we call QQHTD.

Then, we run extensive benchmarks of QHT and other existing filters in Section 3.8. We experimentally prove that QHTs perform faster than any other filter, and in every situation, is either the best or the second best ranking filter.

Finally, in Section 3.9, we explore the behaviour of QHTs in an adversarial setting.

3.4 Redefining the Problem of Duplicate Detection

While the problem defined in Section 2.1.1 is quite simple and intuitive, we show with Theorem 3.4.1 that this 'naive' definition of duplicates is unfit for duplicate filters. Instead, it is better to use a narrower definition, with the introduction of a sliding window.

3.4.1 Filter Saturation

In order to prove the claim that filters saturate, we interest ourselves to the asymptotic error rates of a filter.

Theorem 3.4.1. *Let* FNR_∞ *and* FPR_∞ *be the asymptotic false negative and false positive rates of any filter (i.e., the FNR and FPR on a stream of size going to infinity).*

If $|\Gamma| \gg M$*, then* $\mathrm{FNR}_\infty + \mathrm{FPR}_\infty = 1$*, which characterises random filters (i.e., filters answering randomly to any query).*

Proof. First note that random filters always verify the relation $\mathrm{FNR} + \mathrm{FPR} = 1$: given that a random filter will return DUPLICATE with a probability of p, an unseen element will be classified as DUPLICATE with probability p, and a duplicate will be classified as UNSEEN with probability $1 - p$, hence the result.

Now, on infinite streams, filters are saturated with information. Given that a filter can only store at most one element per bit (cf. Section 2.1.1), a filter of size M can remember at most M distinct elements. However, after fM insertions (with $f \gg 1$), and assuming that $|\Gamma| \gg M$, the filter remembers at most a proportion $\frac{1}{f} \ll 1$ of stream elements.

It can be seen that the error rate of a filter can be modelised as $\mathrm{FPR} + \mathrm{FNR} = \mathrm{ER} = \mu(\eta)$, where μ is a function $\mu : [0, 1] \to [0, 2]$, and η is the proportion (in terms of bits of information) of the stream that the filter has in memory[2]. Moreover, we have $\mu(0) = 1$ as a filter knowing nothing about the stream cannot answer better than random.

Hence, in our situation of a saturated filter, assuming the function μ is continuous at 0, the filter knows at most $\frac{1}{f}$ of the stream, hence the asymptotic error rate will be $\mathrm{ER}_\infty = \mathrm{FPR}_\infty + \mathrm{FNR}_\infty = \lim_{f \to \infty} \mu(\frac{1}{f}) = 1$. ∎

[2] μ can go up to 2 as theoretically, a filter can be always wrong, hence with an error rate of 2.

Interpretation. The interpretation of these results could suggest that streaming filters are useless: they need more memory than a random filter, despite being asymptotically equivalent. However, this is only true because of our hypotheses and definitions: we define a false negative to be a duplicate element claimed as unseen by the filter. However, after some amount of time, it is often acceptable that the element may be considered as unseen again: for instance a nonce is theoretically unique, but in practice after a reasonable amount of time nonce reuse is not a vulnerability. For this reason, adapting false positive and false negative to sliding windows may be relevant here. Moreover, we assumed that all elements of the stream had the same probability of occurrence. In practice, this hypothesis is not always correct, and as we will later see in Section 3.8 (page 48), filters operating on real data perform significantly better than random filters, and resist better to saturation.

Note that, despite the limitations of the problem defined in Definition 2.1.1, it is the most common definition used in duplicate detection related papers [DNB13; DD06; Yoo10].

Finally, while Definition 2.1.1 does not offer a satisfying problem for real-life applications, the asymptotic relation $\mathrm{FPR}_\infty + \mathrm{FNR}_\infty = 1$ offers a good opportunity to verify the validity of the FPR and FNR formulae.

3.4.2 Lower Bound on Saturation

We see that all filters inevitably saturate before becoming as inefficient as a random filter. Hence, it is natural to ask oneself what the optimal limit is, *i.e.*, what the slowest convergence to saturation is. We provide the following result, which we found after the publication of our paper [GLPN19]:

Theorem 3.4.2. *Let $E = (e_1, \dots, e_n)$ be a stream of elements uniformly selected from an alphabet of size $|\Gamma|$. For any duplicate filter of size M, and any sliding window size w, the error probability EP_n^w (defined by $\mathrm{EP}_n^w = \mathrm{FP}^w + \mathrm{FN}^w$) is such that*

$$\mathrm{EP}_n \geq 1 - \frac{1 - \left(1 - \frac{1}{|\Gamma|}\right)^M}{1 - \left(1 - \frac{1}{|\Gamma|}\right)^{\min(n,w)}}$$

for any $n > M$.

Especially, for any filter of size M, the asymptotic error probability EP_∞^w is such that $\mathrm{EP}_\infty^w \geq 1 - \frac{1-\left(1-\frac{1}{|\Gamma|}\right)^M}{1-\left(1-\frac{1}{|\Gamma|}\right)^w} \underset{|\Gamma| \gg w > M}{\approx} 1 - \frac{M}{w}$.

Proof. Let's first prove the theorem for $w = \infty$. In order to prove this result, note that a perfect filter, as noted in Theorem 2.1.1, must use at least 1 bit to store one element (and the bound is not tight for usual amounts of memory). Since the filter has M bits of memory, we conclude that a perfect filter can store at most M elements in memory. Assume that such a filter exists. Because the stream is random, we can assume without loss of generality that the filter stores the M last elements of the stream: any other strategy cannot yield a lower error rate, given the stream is random.

Let us derive the error rate of such a filter. If an element is already stored in the filter, then the optimal filter will necessarily answer DUPLICATE. On the other hand, if the element is not in memory, a perfect filter will answer DUPLICATE with some frequency f, corresponding to probability $p = \frac{1}{f}$.

An unseen element, by definition, will be unseen in the M last elements of the stream, and hence will not be in the filter's memory, the filter will return UNSEEN with probability $1 - p$. For this reason, this optimal filter has a false positive probability of p.

On the other hand, a duplicate element e_n will be classified as UNSEEN if and only if it was not seen in the last M elements of the stream, and the filter answers UNSEEN. Let C be the event "There is a duplicate of e_n in the M previous elements of the stream", and let D be the event "There is at least one duplicate in the stream $(e_1, \dots, e_n)$". Note that $P[C \cap D] = P[C]$ as $C \subset D$.

We have that e_n will trigger a false negative with probability

$$\begin{aligned}
P_n &= (1 - P[C|D])(1 - p) \\
&= \left(1 - \frac{P[C \cap D]}{P[D]}\right)(1 - p) \\
&= \left(1 - \frac{P[C]}{P[D]}\right)(1 - p) \\
&= \left(1 - \frac{1 - P[\bar{C}]}{1 - P[\bar{D}]}\right)(1 - p) \\
P_n &= \left(1 - \frac{1 - \left(1 - \frac{1}{|\Gamma|}\right)^M}{1 - \left(1 - \frac{1}{|\Gamma|}\right)^n}\right)(1 - p)
\end{aligned}$$

Hence, the error probability of the perfect filter is $\mathrm{EP}_n = P_n + p$, which can be rewritten as $\mathrm{EP}_n = \left(1 - \frac{1-\left(1-\frac{1}{|\Gamma|}\right)^M}{1-\left(1-\frac{1}{|\Gamma|}\right)^n}\right)(1-p) + p = 1 - \frac{1-\left(1-\frac{1}{|\Gamma|}\right)^M}{1-\left(1-\frac{1}{|\Gamma|}\right)^n}(1-p)$, which is minimized when $p = 0$.

Given that, by definition, a perfect filter has the lowest error rate of any filter, we get the desired result.

The adaptation of the proof for any w is immediate. ■

Note, as highlighted in the proof, that this bound is not tight, and that better bounds may exist. Also note that this bound only applies when the stream is random: when patterns are found in the stream, information theory informs that data may be stored more efficiently.

Finally, we observe, for $|\Gamma| \gg M$, that the lower bound on the asymptotic error rate of a perfect filter on an infinite sliding window becomes $\mathrm{ER}_\infty \gtrapprox 1 - e^{\frac{M}{|\Gamma|}} \approx 1$, which is another proof of Theorem 3.4.1.

3.5 Revisiting SQF: Quotient Hash Table

SQF is introduced and analysed in [DNB13]. However, several crucial mistakes were made in that analysis. We focus here on four of them that directly impact the claim of SQF near-optimality.

1. A wrong probability of false positive derivation. The usual definition (see Definition 2.1.3) is $P[\mathsf{e\ is\ detected\ as\ duplicate} \mid \mathsf{e\ is\ not\ a\ duplicate}]$, but the authors mistakenly computed a different value, namely $P[\mathsf{e\ is\ detected\ as\ duplicate} \cap \mathsf{e\ is\ not\ a\ duplicate}]$;

2. In their FP derivation, the authors neglected the terms of order > 1 in geometric sums, which led to very large approximations, especially when the sum is infinite;

3. Similarly, their asymptotic analysis of the FN fundamentally relies on the fact that the last term of a geometric sum converges to zero, from which the authors claim the whole sum is close to zero;

4. Cells in the filter's state are not independent: in particular, in every row, non-empty cells hold different values (by design). This affects the computation of the error rates.

Correcting these errors (see Sections 3.6.1 and 3.6.2 below, pages 36 and 40 respectively), and using the same approximation than [DNB13] used[3], $\binom{2r}{r} \approx \frac{4^r}{\sqrt{\pi r}}$, we derive a correct value of the SQF asymptotic FPR and FNR:

$$\mathrm{FPR}_\infty \approx \frac{k}{2^{r'}\sqrt{\pi r}} \qquad \mathrm{FNR}_\infty \approx 1 - \frac{k}{2^{r'}\sqrt{\pi r}}$$

that disagree with [DNB13] — most significantly, neither the FPR nor the FNR decrease to 0, contrarily to what was claimed. Interestingly, the suggested parameters ($r = 2$, $k = 4$, $r' = r/2 = 1$) indeed achieve an FNR of 0 as the authors claim — but also an FPR of 1. With these values, only 4 distinct fingerprints exist, and they can all be stored in the 4 cells there exist per row. When the filter is full, any duplicate will be reported as such, hence a FNR of 0. But every new element will also be reported as duplicate, hence a FPR of 1.

Further more, there is redundancy between the Hamming weight $\omega(h_r(e))$ of $h_r(e)$ and the reduced remainder $h_{r'}(e)$. For instance, if $h_{r'}(e)$ contains at least one bit set to 1, we know that $\omega(h_r(e)) \neq 0$. Intuitively this means SQF is wasteful, and we could expect to avoid collisions in the filter's state by using a better adjusted encoding: this intuition happens to be correct as we show below (see Section 3.6.3, page 44).

[3] Even though not mentioned in [DNB13], the approximation (stemming from Catalan numbers) is only asymptotically valid, but computations show that the approximation is satisfying even for small values of r, such as the ones used in practical applications.

3.5.1 Full-size Hashing, Memory Adjustments

Our first and main observation is that SQF's fingerprint scheme (a hash and a Hamming weight of part of the hash) can be fruitfully replaced by a single hash function of the same size. Not only does this simplify the theoretical analysis, it also provides a much more efficient use of the available space, as the Hamming weight contains information already contained in the hash.

As a reminder, for an element e, an SQF computes a hash $h(e) = h_q(e)\|h_e^r$ of $q+r$ bits, and from $h_r(e)$ retrieves r' bits, thus forming $h_{r'}(e)$. Then the SQF stores the fingerprint $h_e r'\|\omega(h_r(e))$ with $\omega(h_r(e))$ be the Hamming weight of $h_r(e)$. Hence, we see that both $h_e r'$ and $\omega(h_r(e))$ contain bits of information about $h_r(e)$, which leads to redundancy and lack of compactness in the information stored.

Hence, instead of storing $h_e r'\|\omega(h_r(e))$, we suggest to simply store a hash h'_e of the same size. As we will see later on, this improvement also significantly simplifies the error rate analysis.

Our second improvement to SQF is to use more flexible hash functions, that give much more flexibility in adjusting the total memory M of the filter. As said previously, SQF uses tables of size 2^q, where q is a fixed parameter. This parameter lacks flexibility, as it requires that SQF's memory is a multiple of 2^q. If we allow for a hash function with image on an arbitrary range $[0, N]$ for any N (and not $[0, 2^q - 1]$), we allow our structure to be more resilient to any amount of memory we can have. While this improvement does not lead to a change in cases where the available memory is optimally designed, it however allows for more flexibility for users.

Combining these two effects, we obtain our new structure, which we call the Quotient Hash Table (QHT). We give its pseudocode in Algorithm 3.

The QHT takes three arguments N, s, k. From this we create a table of N rows with k cells each. Each cell has a capacity of s bits, *i.e.*, there are $S = 2^s$ possible fingerprints in our system. The storage of a new element is similar to the one in SQF.

3.5.2 Empty Cells

SQF and QHT as described above make essential use of the "empty" cells in T. The need for this feature is present for all fingerprint-based structures including Cuckoo filters [FAK+14]. Other constructions, not relying on fingerprints, do not face this issue, including SBF [DD06] and b_DBF [SZ08]: in these schemes, 0 always codes for absence. However, a low-level implementation cannot rely on the availability of such a special value. Our options are to initialise all cells to 0 and either:

1. treat 0 as a fingerprint;
2. if an element has a fingerprint of 0, reassign its fingerprint to 1;
3. while an element has a fingerprint 0, compute a new fingerprint based on some deterministic scheme.

Algorithm 3 QHT Setup and Stream

```
1: function Setup(M, s, k)                          ▷ M > sk and 0 < k ≤ 2^s
2:     Let N ← ⌊M/(k · s)⌋
3:     Choose a hash function h over [0, N − 1]
4:     Choose a hash function f over [0, 2^s − 1]
5:     Let T be a N × k array with s-bit cells, initialized to ⊥

1: function Stream(e)
2:     for each cell b_i in row T[h(e)] do
3:         if b_i = f(e) then
4:             return DUPLICATE
5:     Let b_∅ be the first empty cell in row T[h(e)]
6:     if b_∅ does not exist then
7:         b_∅ ←$ T[h(e)]
8:     Store f(e) in cell b_∅
9:     return UNSEEN
```

The first option is at a risk of a high false positive rate, even for small streams: when a new element, whose fingerprint is 0, should be stored in an empty cell, it is instead dismissed as a duplicate. More specifically, before the filter is completely filled, a new element has probability at least $\frac{1}{S}$ to be false positive, where S is the number of distinct possible fingerprints, which leads to a high number of false positive at the beginning of any stream.

The second option is a bit faster than the third: the third option needs, on average, $\frac{S}{S-1} = 1 + \frac{1}{S-1}$ hash computations for each insertion, whereas the second option only needs 1. On the other hand, the third option has better statistical properties, making the analysis simpler. While not specified in their paper [FAK+14], the second solution was chosen in the official Cuckoo filter implementation[4]. On the opposite, we retain the slower, but easier to analyse option of re-fingerprinting.

3.5.3 Semi-sorting

The technique of *semi-sorting* was introduced in [FAK+14] to shave some extra storage. The idea is as follows: treating empty cells as cells containing "0" fingerprint, for each row, sort the cells by their fingerprints, and then encode the result. Given that, for $s = 4$ and $k = 4$, only 3,876 possible sorted states exist, and as such a sorted state can be stored using only 12 bits, as opposed to the 16 bits required to store four 4-bit fingerprints.

The same optimization can be used for QHT. In practice, [FAK+14] recommend its use when using 4 cells of 4 bits each.

[4] https://github.com/efficient/cuckoofilter/blob/aac6569cf30f0dfcf39edec1799fc3f8d6f594da/src/cuckoofilter.h

3.5.4 Comparison with Hash Tables

As our structure QHT relies on hashes, the natural question is to know whether our structure is more efficient than a hash table. The most notable difference is that we do not store the element in its entirety, but rather its hash compaction, as described in [WL93]. Moreover, it can be seen that a hash map is indeed a QHT, wherein the number of rows is equal to 1. QHT being a superset of hash maps, we have the guarantee that we cannot perform worse. Other implementations of hash tables, such as Hashmaps in Java, are closer to our structure. However, we have a fixed number of cells per row, which is not the case in an Hashmap.

In a nutshell, one can say that a QHT is a hash map of fixed size using hash compaction techniques.

3.6 Error Rate Analysis

3.6.1 False Positive Rate

Consider a QHT with N rows, k cells per row, and s-bits fingerprints. For simplicity, further assume that no cell is empty (which is true after some time), and that the stream is sampled uniformly at random from Γ. We want to derive the probability of triggering a false positive.

FPR Intuition. We first give an intuitive, and simple, estimation of the false positive rate. Even though not rigorous, and lacking interesting data about saturation as we discuss later, it helps confirming that the results obtained in the next section are indeed correct.

Let us consider an element e which has never been seen. e is assigned to row $h_q(e)$, and we want to know the probability that its fingerprint $f(e)$ is already stored in one of the cells of the row, resulting in a false positive. In this section, we assume that the stream being uniform, the fingerprints stored in the row $h_q(e)$ are all equiprobable (with probability $\frac{1}{S}$, where in our case $S = 2^s$), so the probability of finding $s(e)$ in one row is simply $\frac{k}{S}$. Hence, we get an estimate that the FPR will be approximately $\frac{k}{S}$.

Despite being simple and elegant, this approach does not give much hindsight about how fast this limit is reached.

FPR Derivation. We now derive an exact value for the FPR of a QHT.

Theorem 3.6.1. *For a QHT of N rows, k cells per row and S possible fingerprints and assuming $NS \ll |\Gamma|$, and for a sliding window of size $w \ll |\Gamma|$, the FPR after m insertions on a sliding window of size w is*

$$\mathrm{FPR}_m^w = \frac{k}{S}\left(1 - \frac{(Nk-1)\left(1-\left(1-\frac{1}{Nk}\right)^m\right)}{m}\right)$$

Proof. We instead prove the following lemma, from which the proof of the theorem becomes trivial (the theorem can be deduced from the lemma with a simple arithmetic mean, given that $\mathrm{FPR}_m = \frac{1}{m}\sum_{i=1}^{m}\mathrm{FP}_i$):

Lemma 3.6.1. *For a QHT of N rows, k cells per row and S possible fingerprints and assuming $NS \ll |\Gamma|$, with a sliding window of size w, the probability FP_m that an unseen element inserted after m other elements triggers a false positive on a sliding window of size w (with $w \ll |\Gamma|$) is*

$$\mathrm{FP}_m^w = \frac{k}{S}\left(1 - \left[1 - \frac{1}{Nk}\right]^m\right)$$

An element e is a false positive if and only if e has not been encountered in the current sliding window, but $\mathsf{Detect}(e) = \mathsf{DUPLICATE}$. This event can be triggered by two possibilities:

- The presence, in the filter, of another element e' with a hash and fingerprint colliding with those of e. This is called a *false duplicate*.
- The presence, in the filter, of the same element e, but which was inserted more than w epochs before.

These two elements are referred to as *false collisioners*: they create a collision when there should not be one. If a false collisioner is still in the filter when e arrives, we refer to the event as a *hard collision*.

Our first remark is that the only false collisioner that may create a hard collision with e is the latest false collisioner inserted before e arrives: let us assume that e_1, e_2 are false collisioners, and that e_1 arrives before e_2. When we insert e_2 in the filter, e_1 is either still in the filter or has been evicted.

- If e_1 has been evicted, then e_1 will not hardly collide with e.
- If e_1 is still in the filter, then e_2 will be claimed as a duplicate and dismissed.

However, if we look at the table T storing all fingerprints, dismissing e_2 is strictly equivalent to replacing e_1 by e_2. Consequently, every false collisioner is erased by any new false collisioner, and only the latest false collisioner can cause a hard collision. As a result, we will only focus on the probability that the latest false collisioner (before e arrives) causes a hard collision.

Probability of non-eviction of the latest false collisioner until e arrives. Let us assume that the latest false collisioner appears at position i of the stream $E = \{e_1, e_2, \ldots, e_m, e\}$, in other words, e_i is the latest false collisioner in the stream before e.

Now, e_i has to remain in the filter until e arrives, even though new elements are added. Let us suppose that element e_j, $i < j \leq m$, evicts e_i from the filter. For e_i to be evicted, the following conditions must be true:

- $h(e_j) = h(e_i)$
- $s(e_j)$ is different from all the other fingerprints stored in the row $T[h(e_j)]$ (knowing that one of the cells contains $s(e_i)$)
- $s(e_j)$ is inserted into the cell in which $s(e_i)$ is stored

Since we know that e_i is the latest false collisioner, we cannot simultaneously have $h(e_j) = h(e_i)$ and $s(e_j) = s(e_i)$. As such, the first two conditions are not independent. Let P_{hs} be the probability that these two conditions are satisfied. Given that e_i is the latest false collisioner, there are only $NS-1$ possibilities for the couple $(h(e_j), s(e_j))$. Moreover, among these $NS-1$ states, only $S-1$ verify the first condition, and among these $S-1$ states, only $S-k$ verify the second condition. Finally, $P_{\text{hs}} = \frac{S-k}{NS-1}$. If we assign the latest event to the probability $P_{\text{selection}}$, we immediately get $P_{\text{selection}} = \frac{1}{k}$.

Finally, the probability $P_{\neg\text{evict}}$ of e_i not being evicted by e_j is $P_{\neg\text{evict}} = 1 - P_{hs}P_{\text{selection}}$ and: $P_{\neg\text{evict}} = 1 - \frac{S-k}{k(NS-1)}$

Now, e_i has to avoid eviction by every element before e arrives, *i.e.*, by all elements $e_{i+1}, \ldots, e_m$, which happens with probability $P(\text{hc})_i = (P_{\neg\text{evict}})^{m-i}$.

At that point, we know the probability that a hard collision happens when the latest false collisioner has been inserted at position i.

Probability of the last false collisioner to be at position j. The probability of any element e' being a false collisioner depends on the position of e' in the stream. Let j be the position of e' in the stream $E = \{e_1, e_2, \ldots, e_m, e\}$: $e_j = e'$.

- If $m - j \geq w$, then e' is a false collisioner and can be either a false duplicate (*i.e.*, $e' \neq e$ but $h(e') = h(e)$ and $s(e') = s(e)$) or equal to e.
- If $m - j < w$, then e' can only be a false duplicate.

In the former case, e' is a false duplicate or equal to e. Let P_{fc} be the probability of this to happen. We immediately get $P_{\text{fc}} = \frac{1}{NS}$.

In the latter case, let P_{fd} be the probability of e' to be a false duplicate. Since we know $e' \neq e$, we have $P_{\text{fd}} = \dfrac{\frac{|\Gamma|}{NS} - 1}{|\Gamma|} = \dfrac{1}{NS} - \dfrac{1}{|\Gamma|}$ (amongst $|\Gamma|$ elements, there are $\frac{|\Gamma|}{NS}$ who have the same hash h and the same fingerprint s than e, including e, hence the result).

Finally, the probability that e_j is the latest false positive is equal to $(1 - P_{\text{fd}})^{m-j} P_{\text{fd}}$ for $j > m - w$, and $(1 - P_{\text{fd}})^{w} (1 - P_{\text{fc}})^{m-w-j} P_{\text{fc}}$ otherwise.

Summing up the probabilities. Thus, for $m > w$, the probability FP_m^w that a false positive happens on the sliding window of size w after m elements are inserted is

$$\begin{aligned}
\mathrm{FP}_m^w &= \sum_{j=1}^{m} (e_j \text{ is the latest false collision } \times e_j \text{ is not evicted until } e \text{ arrives}) \\
&= \sum_{j=1}^{m-w} (1-P_{\mathrm{fd}})^w (1-P_{\mathrm{fc}})^{m-w-j} P_{\mathrm{fc}} \times P(\mathrm{hc})_j + \sum_{j=m-w+1}^{m} (1-P_{\mathrm{fd}})^{m-j} P_{\mathrm{fd}} \times P(\mathrm{hc})_j \\
&= \frac{(1-P_{\mathrm{fd}})^w}{(1-P_{\mathrm{fc}})^w} P_{\mathrm{fc}} \sum_{j=1}^{m-w} (1-P_{\mathrm{fc}})^{m-j} \times P(\mathrm{hc})_j + P_{\mathrm{fd}} \sum_{j=m-w+1}^{m} (1-P_{\mathrm{fd}})^{m-j} \times P(\mathrm{hc})_j
\end{aligned}$$

We have $P(\mathrm{hc})_j = \left(1 - \frac{S-k}{k(NS-1)}\right)^{m-j}$, $P_{\mathrm{fd}} = \frac{1}{NS} - \frac{1}{|\Gamma|}$ and $P_{\mathrm{fc}} = \frac{1}{NS}$, so

$$\begin{aligned}
\mathrm{FP}_m^w &= \left(1 + \frac{NS}{|\Gamma|(NS-1)}\right)^w \frac{1}{NS} \left(1 - \frac{1}{kN}\right)^w \frac{1 - \left(1 - \frac{1}{kN}\right)^{m-w}}{1 - \left(1 - \frac{1}{kN}\right)} \\
&\quad + \left(\frac{1}{NS} - \frac{1}{|\Gamma|}\right) \frac{1 - \left[1 - \frac{1}{kN} + \frac{1}{|\Gamma|}\left(1 - \frac{S-k}{k(NS-1)}\right)\right]^w}{\frac{1}{kN} - \frac{1}{|\Gamma|}\left(1 - \frac{S-k}{k(NS-1)}\right)} \\
&= \left(1 + \frac{NS}{|\Gamma|(NS-1)}\right)^w \frac{k}{S} \left(1 - \frac{1}{kN}\right)^w \left(1 - \left(1 - \frac{1}{kN}\right)^{m-w}\right) \\
&\quad + \left(\frac{1}{NS} - \frac{1}{|\Gamma|}\right) \frac{1 - \left(1 - \frac{1}{kN}\right)^w \left[1 + \frac{1}{|\Gamma|(1-1/(kN))}\left(1 - \frac{S-k}{k(NS-1)}\right)\right]^w}{\frac{1}{kN} - \frac{1}{|\Gamma|}\left(1 - \frac{S-k}{k(NS-1)}\right)}
\end{aligned}$$

Assuming $|\Gamma| \gg NS$ and $|\Gamma| \gg w$, we have

$$\begin{aligned}
\mathrm{FP}_m^w &\approx \frac{k}{S}\left(\left(1 - \frac{1}{kN}\right)^w - \left(1 - \frac{1}{kN}\right)^m\right) + \frac{1}{NS} \frac{1 - \left(1 - \frac{1}{kN}\right)^w}{\frac{1}{kN}} \\
&\approx \frac{k}{S}\left[1 - \left(1 - \frac{1}{kN}\right)^m\right]
\end{aligned}$$

Note that this derivation is only valid for $m > w$. Otherwise, we obtain the same result, but in a different way:

$$\begin{aligned}
\mathrm{FP}_{m \le w}^w &= \sum_{j=1}^{m} (1-P_{\mathrm{fd}})^{m-j} P_{\mathrm{fd}} \times P(\mathrm{hc})_j \\
&= \left(\frac{1}{NS} - \frac{1}{|\Gamma|}\right) \frac{1 - \left(1 - \frac{1}{kN}\right)^m \left[1 + \frac{1}{|\Gamma|(1-1/(kN))}\left(1 - \frac{S-k}{k(NS-1)}\right)\right]^m}{\frac{1}{kN} - \frac{1}{|\Gamma|}\left(1 - \frac{S-k}{k(NS-1)}\right)}
\end{aligned}$$

Neglecting all the terms in $\frac{1}{|\Gamma|}$,

$$\mathrm{FP}^w_{m\leq w} \approx \frac{k}{S}\left(1-\left(1-\frac{1}{kN}\right)^m\right)$$

Which concludes the proof. ■

One observation is that the FPR converges to $\frac{k}{S}$. Furthermore, thanks to the expression of FP_m, we can see that the more rows there are (*i.e.*, the bigger N is), the slower the FPR reaches its asymptotic (saturated) value.

Finally and more importantly, the FPR^w does not depend on w on a first approximation. This is due to the fact that QHT was mostly designed for the cases where w is infinite, and as such w plays no role in the algorithm, and little difference in probabilities when an element is or is not in the sliding window. As such, QHT might be less efficient than other structures on streams with small sliding windows, but still remains a competitive candidate (see Section 3.8.6 below, page 53).

Application to SQF One should be tempted to directly apply this result to an SQF. However, as pointed out earlier, in an SQF fingerprints are correlated and therefore not equiprobable. For instance, consider the SQF with the parameters $r = 3$, $r' = 1$. Only the hash $h^r_1 = 000$ will lead to the fingerprint 000, whereas both hashes $h^r_2 = 001$ and $h^r_3 = 010$ will lead to the fingerprint 001.

However, for an optimal SQF, fingerprints are equiprobable so the analysis above holds for optimal SQFs. In the general case, the authors of [DNB13] have approximated the probability of fingerprint collision in an SQF with $\frac{1}{2r'\sqrt{\pi r}}$. Replacing $\frac{1}{S}$ with this probability in our analysis, we get an *approximate* asymptotic FPR of $\frac{k}{2^{r'}\sqrt{\pi r}}$ for SQFs, as announced in Section 3.5.

3.6.2 False Negative Rate

Theorem 3.6.2. *For a QHT of N rows, k cells per row, and S different fingerprints, assuming $|\Gamma| \gg N$, then as the number of insertions goes to infinity,* $\mathrm{FNR}_\infty = 1-\frac{k}{S}$.

In the following proof, we also show that the false negative rate over a sliding window FNR^w is directly linked to the sliding window size, as it depends on a summation of w terms. The exact derivation is however left for future work.

Proof. Assume that e is a duplicate, we denote by e_i the latest element in the stream such that $e_i = e$. Following the same reasoning as for the FPR, e_i will trigger a false negative if and only if e_i is removed from the filter before e arrives, and if any false duplicate of e, inserted between the removal of e_i and e, is deleted before e arrives.

Let us assume that e_i is deleted at time j (this happens with probability $P^{\mathrm{Del}}_{i,j}$) by something else than a false duplicate. Using similar arguments than in the previous section, we have $P^{\mathrm{Del}}_{i,j} = \left(1-\frac{S-k}{k(NS-1)}\right)^{j-i-1}\frac{S-k}{k(NS-1)}$: the probability of the ith element is deleted by the jth element is equal to the probability of non-deletion by

$j-i-1$ elements ($\left(1-\frac{S-k}{k(NS-1)}\right)^{j-i-1}$) followed by the probability of deletion at the next step ($\frac{S-k}{k(NS-1)}$)

The probability that all false duplicates, inserted after time j, are deleted before e arrives is $1-\mathrm{FP}_{n-j}$.

Let $a=\frac{S-k}{k(NS-1)}$, $b=\frac{k}{S}$ and $c=1-\frac{1}{Nk}$, and denote by $\mathrm{FN}_{i,m}$ the probability, in a stream $E=(e_1,\ldots,e_m,e)$ where the latest duplicate of e is inserted at i, that e_i is deleted before e arrives and no false positive persists until e,

$$\begin{aligned}\mathrm{FN}_{i,m} &= \sum_{j=i+1}^{m} P_{i,j}^{\mathrm{Del}}(1-\mathrm{FP}_{m-j}) \\ &= \sum_{j=i+1}^{m} (1-a)^{j-i-1}\, a\left(1-b\left(1-c^{m-j}\right)\right) \\ \mathrm{FN}_{i,m} &= a\,(1-b)\sum_{j=i+1}^{m}\left[(1-a)^{j-i-1}\right] + ab\sum_{j=i+1}^{m}\left[(1-a)^{j-i-1}\,c^{m-j}\right]\end{aligned}$$

Given that $1-a\neq c$ (because $1-a-c=1-\frac{S-k}{k(NS-1)}-1+\frac{1}{Nk}=\frac{Nk-1}{Nk(NS-1)}\neq 0$ unless $N=k=1$ which would not be an interesting filter), we get:

$$\mathrm{FN}_{i,m} = (1-b)\left(1-(1-a)^{m-i}\right) + ab\frac{(1-a)^{m-i}-c^{m-i}}{1-a-c}$$

The probability FN_m of e to be a false negative is then $\sum_{i=1}^{m} P_{\mathrm{dup},i}\cdot \mathrm{FN}_{i,m}$, where $P_{\mathrm{dup},i}$ is the probability that the latest duplicate already seen is e_i. In a previous version of this work, we wrongly stated that $P_{\mathrm{dup},i}$ was equal to the value $P'_{\mathrm{dup},i}=\left(\frac{|\Gamma|-1}{|\Gamma|}\right)^{m-i}\frac{1}{|\Gamma|}$: this would be true if there was no assumption on the stream, but here we know for a fact that there is already a duplicate somewhere in the stream. Hence, we must not derive $P'_{\mathrm{dup},i}=P\left[e_i \text{ is the latest duplicate}\right]$, but rather $P_{\mathrm{dup},i}=P\left[e_i \text{ is the latest duplicate} \mid \text{there is at least one duplicate}\right]$.

For simplicity, we note by D the event 'there is at least one duplicate'. Observe that D can also be written $\exists k\leq m, e_k=e$. We also remark that the event 'e_i is the latest duplicate' (of e) can be rewritten as $(e_i=e)\cap(\bigcap_{j>i}(e_j\neq e))$. Hence,

$$P_{\mathrm{dup},i}=P\left[(e_i=e)\cap\left(\bigcap_{j>i}(e_j\neq e)\right)\mid D\right]$$

By definition of a conditional probability,

$$\begin{aligned}
P_{\text{dup},i} =& \frac{P\left[(e_i = e) \cap \left(\bigcap_{j>i}(e_j \neq e)\right) \cap D\right]}{P[D]} \\
=& \frac{P\left[\left(\bigcap_{j>i}(e_j \neq e)\right) \cap (e_i = e) \cap (\exists k \leq m, e_k = e)\right]}{P[D]} \\
=& \frac{P\left[\left(\bigcap_{j>i}(e_j \neq e)\right) \cap (e_i = e)\right]}{P[D]} \\
P_{\text{dup},i} =& \frac{P'_{\text{dup},i}}{1 - P\left[\overline{D}\right]}
\end{aligned}$$

Where $\overline{D}$ is the event '$\mathsf{There\ is\ no\ duplicate}$' and has a probability $P\left[\overline{D}\right] = \left(\frac{|\Gamma|-1}{|\Gamma|}\right)^m$.

Hence, $P_{\text{dup},i} = \left(\frac{|\Gamma|-1}{|\Gamma|}\right)^{m-i} \frac{1}{|\Gamma|} \frac{1}{1-\left(\frac{|\Gamma|-1}{|\Gamma|}\right)^m}$.

We obtain the probability FN_m that the $(m+1)^{\text{th}}$ element e of the stream will be a false negative:

$$\begin{aligned}
\text{FN}_m =& \sum_{i=1}^{m} P_{\text{dup},i} \cdot \text{FN}_{i,m} \\
\text{FN}_m =& \sum_{i=1}^{m} \left[\left(\frac{|\Gamma|-1}{|\Gamma|}\right)^{m-i} \frac{1}{|\Gamma|} \left((1-b)\left(1-(1-a)^{m-i}\right) + ab\frac{(1-a)^{m-i} - c^{m-i}}{1-a-c}\right)\right]
\end{aligned}$$

For a stream of n elements, the false negative rate FNR_n is defined as the average error probability: $\text{FNR}_n = \frac{1}{n}\sum_{m=1}^{n} \text{FN}_m$. For a stream of n elements on a sliding window of size w, the false negative rate FNR_n^w over the sliding window is defined as the average over the last w elements: $\text{FNR}_n^w = \frac{1}{w}\sum_{m=n-w+1}^{n} \text{FN}_m$.

Hence,

$$\begin{aligned}
\text{FNR}_n = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{m} \left(\frac{|\Gamma|-1}{|\Gamma|}\right)^{m-i} \frac{1}{|\Gamma|} \frac{1}{1-\left(\frac{|\Gamma|-1}{|\Gamma|}\right)^m} \Big[(1-b) \times \left(1-(1-a)^{m-i}\right) \\
+ab\frac{(1-a)^{m-i} - c^{m-i}}{1-a-c}\Big]
\end{aligned}$$

Expanding and summing the geometric sums,

$$\begin{aligned}\mathrm{FNR}_n = \frac{1}{n|\Gamma|}\sum_{m=1}^{n}\frac{1}{1-\left(\frac{|\Gamma|-1}{|\Gamma|}\right)^m}\Bigg[&(1-b)\sum_{i=1}^{m}\left(\frac{|\Gamma|-1}{|\Gamma|}\right)^{m-i}\\ &-(1-b)\sum_{i=1}^{m}\left(\frac{|\Gamma|-1}{|\Gamma|}\cdot(1-a)\right)^{m-i}+\frac{ab}{1-a-c}\sum_{i=1}^{m}\left(\frac{|\Gamma|-1}{|\Gamma|}(1-a)\right)^{m-i}\\ &+\frac{ab}{1-a-c}\sum_{i=1}^{m}\left(\frac{|\Gamma|-1}{|\Gamma|}c\right)^{m-i}\Bigg]\end{aligned}$$

$$\begin{aligned}\mathrm{FNR}_n = \frac{1}{n}\sum_{m=1}^{n}\Bigg[&1-b-\frac{1-b}{1-(1-1/|\Gamma|)^m}\frac{1-((1-1/|\Gamma|)(1-a))^m}{|\Gamma|-(1-a)(|\Gamma|-1)}\\ &+\frac{1}{1-(1-1/|\Gamma|)^m}\frac{ab}{1-a-c}\frac{1-((1-1/|\Gamma|)(1-a))^m}{|\Gamma|-(1-a)(|\Gamma|-1)}\\ &-\frac{1}{1-(1-1/|\Gamma|)^m}\frac{ab}{1-a-c}\frac{1-((1-1/|\Gamma|)c)^m}{|\Gamma|-c(|\Gamma|-1)}\Bigg]\end{aligned}$$

We can see that the sum can be rewritten as $\mathrm{FNR}_n = 1-b+\sum_{i=1}^{3} k_i \frac{1}{n}\sum_{m=1}^{n}\frac{1-\alpha_i^m}{1-\beta_i^m}$, where k_i, α_i, β_i are various values. To the best of our knowledge, there is no simpler form of the expression $\frac{1}{n}\sum_{m=1}^{n}\frac{1-\alpha^m}{1-\beta^m}$. However, given that $\mathrm{FNR}_\infty = \lim_{n\to\infty}\mathrm{FNR}_n$, and because in our case the terms α and β are < 1, we know that $\lim_{m\to\infty}\frac{1-\alpha^m}{1-\beta^m} = 1$, Cesàro's summation tells us that $\lim_{n\to\infty}\frac{1}{n}\sum_{m=1}^{n}\frac{1-\alpha^m}{1-\beta^m} = 1$ as well, so we get the final result:

$$\begin{aligned}\mathrm{FNR}_\infty = 1-b-\frac{1-b}{|\Gamma|-(|\Gamma|-1)(1-a)}+\frac{ab}{(1-a-c)(|\Gamma|-(|\Gamma|-1)(1-a))}\\ -\frac{ab}{(1-a-c)(|\Gamma|-(|\Gamma|-1)c)}\end{aligned}$$

Finally, assuming $|\Gamma| \gg Nk$ (or even $|\Gamma| \gg N$), *i.e.*, that there are more distinct elements in the stream that what the filter is able to store, we get the limit rate of $1-b$, which is $\mathrm{FNR}_\infty \approx 1-\frac{k}{S}$. ■

A not obvious consequence of the above expression for FNR is that when N increases, which corresponds to using more memory, it is possible to achieve an arbitrary small error false negative rate. The proof is straightforward but tedious, as it requires the substitution of a, b, c by their full expression, before taking the limit. However, a formal computing software such as Mathematica outputted without any problem. Thus, the formal proof was omitted for the sake of simplicity. Another

reason for omitting the proof is that the result implies that we have an infinitely growing memory. However, since the beginning we assume memory to be a limited ressource. Moreover, at some point in the infinite growth, one obtains a filter with more bits of memory than the cardinality of stream elements, thus making void the need of any QHT as one can instead use the optimal solution (see Theorem 2.1.1).

Note that $\mathrm{FNR}_\infty + \mathrm{FPR}_\infty = 1$, as was predicted by Theorem 3.4.2.

The value of FNR_∞ also gives (using the same corrections as for the FPR in Section 3.6.1) the FNR formula for the SQF.

3.6.3 Comparing QHT to SQF

Let us compare the memory required for a QHT to reach the same error rates than an SQF.

Given that both FPR and FNR depend only on k, N and S, imposing the equality on these parameters ensures that both filters have exactly the same FPR and FNR.

Note that S is not a user-chosen parameter, but rather a consequence of other parameters.

Theorem 3.6.3. *For exactly the same FPR and FNR, a QHT requires exactly 33% fewer memory than an SQF.*

Proof. The proof is made through the following paragraphs.

Deriving S from filters parameters. For a QHT, S is derived from the number of bits of the fingerprint s, with the straightforward relation[5]: $S = 2^s$. For an SQF, however, the relation is more complicated.

When r and r' are fixed, for any element e, let $h_r(e)$ be decomposed as $h_r(e) = h_{r'}(e) \| f$, where $h_{r'}(e)$ is the r'-bits word used in the fingerprint, h being the $r - r'$ remaining bits of $h_r(e)$. For $\omega(\cdot)$ the Hamming weight function, we have $s(e) = h_{r'} \| \omega(h_r(e))$. Yet $\omega(h_r(e)) = \omega(h_{r'}(e)) + \omega(f)$. We know that $\omega(h_{r'}(e))$ is entirely dependent on $h_{r'}(e)$, which is already used in the fingerprint. Thus, if we fix $h_{r'}(e)$, there are only $r - r' + 1$ possible values for $\omega(f)$ and thus for $\omega(h_r(e))$. Given that $h_{r'}(e)$ can have $2^{r'}$ different values, we get that $S_{SQF} = 2^{r'} \cdot (r - r' + 1)$.

Comparing required memory. For QHTs, we have the relation $M_{\mathrm{QHT}} = N_{\mathrm{QHT}} k_{\mathrm{QHT}} s$. For SQF, the formula is rather $M_{\mathrm{SQF}} = N_{\mathrm{SQF}} k_{\mathrm{SQF}} (r' + \lceil \log_2(r+1) \rceil)$ (because fingerprints occupy $r' + \lceil \log_2(r+1) \rceil$ bits), with r and r' being parameters in SQF's settings.

[5] If we are on a system without the `empty` feature (see Section 3.5.2), then $S = 2^s - 1$. For an SQF, one can just assign the `empty` value to one of the unassigned fingerprints.

Given that $N_{\text{QHT}} = N_{\text{SQF}}$ and $k_{\text{QHT}} = k_{\text{SQF}}$, the ratio $\frac{M_{\text{QHT}}}{M_{\text{SQF}}}$ is:

$$\begin{aligned}\frac{M_{\text{QHT}}}{M_{\text{SQF}}} &= \frac{N_{\text{QHT}} k_{\text{QHT}} s}{N_{\text{SQF}} k_{\text{SQF}} (r' + \lceil \log_2(r+1) \rceil)} \\ &= \frac{s}{r' + \lceil \log_2(r+1) \rceil}\end{aligned}$$

Given that $s = \lceil \log_2(S_{\text{QHT}}) \rceil = \lceil \log_2(S_{\text{SQF}}) \rceil = \lceil \log_2(2^{r'} \cdot (r - r' + 1)) \rceil$, we have

$$\begin{aligned}\frac{M_{\text{QHT}}}{M_{\text{SQF}}} &= \frac{\lceil \log_2(2^{r'}(r - r' + 1)) \rceil}{r' + \lceil \log_2(r+1) \rceil} \\ &= \frac{r' + \lceil \log_2(r - r' + 1) \rceil}{r' + \lceil \log_2(r+1) \rceil}\end{aligned}$$

Using the recommended settings in [DNB13] ($r = 2$, $r' = 1$) the ratio becomes $\frac{M_{\text{QHT}}}{M_{\text{SQF}}} = \frac{2}{3}$, which concludes the proof. ■

This result proves that a QHT is always superior to its predecessor SQF, by quite a high margin in terms of memory requirements.

3.6.4 Parameter Tuning

We now precise how to optimise our filter parameters. In the case where $w = \infty$, we are able to give a full proof, whereas in the general case, we rely on experimental observations.

For $w = \infty$. As noted in Section 3.6.2, no matter the choice of the parameters we have $\text{FNR}_\infty + \text{FPR}_\infty = 1$: any particular parameters choice will be a trade off between good FPR and good FNR performance, at least asymptotically. However, when the stream is small enough, one may choose parameters that will maximally delay saturation.

Theorem 3.6.4. *For a desired asymptotic FPR corresponding to some ratio $\frac{k}{S}$, the configuration leading to the slowest convergence to saturation is the one with the lowest S value.*

Proof. We know that $M = Nk \log_2(S)$, so plugging this into the FP formula gives

$$\begin{aligned}\text{FP}_m &= \frac{k}{S}\left(1 - \left[1 - \frac{1}{kN}\right]^m\right) \\ &= \frac{k}{S}\left(1 - \left[1 - \frac{\log_2(S)}{M}\right]^m\right)\end{aligned}$$

Which means that, for fixed M and $\frac{k}{S}$ (*i.e.*, for a fixed memory amount and a fixed asymptotic FPR), $\log_2(S)$ must be as small as possible in order to keep the FP low for as long as possible. Since the FP is as low as possible, the FPR (*i.e.*, the average of all previous FPs) is also as low as possible. ■

For instance, assume that we want an asymptotic FPR of 25%. The potential values for the couple (k, S) are $(1, 4)$, $(2, 8)$, $(4, 16)$ and so on (leading respectively to $s = 2, 3, 4$). Because of the above relation, we know that setting $k = 1, s = 2$ will yield the best saturation resistance for the FPR.

Note however that this result does not guarantee a slowest convergence of the FNR. However, experiments show that it is the case indeed.

We compared several QHT of approximately 2^{16} bits, with the same ratio $\frac{k}{S}$, but each with a different value for S. We took streams of 100 000 elements (well under saturation value for this amount of memory), from an alphabet of 2^{20} elements. Each element of the stream was uniformly randomly selected, leading to a stream with about 4.6% of duplicate elements. We averaged the results on 10 runs.

We observe in Table 3.1 that filters with a small value of S do indeed perform better than filters with a bigger value of S, which concludes the experiment.

	$S = 4$	$S = 8$	$S = 16$	$S = 32$	$S = 64$
FPR (%)	**22.57**	23.25	23.53	23.62	23.50
FNR (%)	**35.89**	44.24	50.77	54.55	58.73
FPR + FNR	**58.45**	67.49	74.30	78.17	82.23

Table 3.1: Error rates of QHTs with the same asymptotic FPR. The QHT with both better FPR and FNR is in bold.

For arbitrary w. In this case, we were not able to extract a mathematical formula for optimizing the error rate. Instead, we benchmarked several QHTs of 1Mb, with $k = 1$, and different values of S. We analyzed their efficiency, depending on the sliding window size (ranging from 100 elements to 10M elements), on a stream of 150M random elements of 2^{26} bits each. Results are given in Table 3.2. We indeed observe that, for big sliding windows, it is preferable to select a small value of S, whereas for small to very small sliding windows, big values of S are more efficient. However, in this kind of situation, where the sliding window is very small, probabilistic structures might not be useful, as it is probably easier to store all elements in memory. Hence, we conclude that, for all practical applications, one must chose an S as small as possible. Note that $S = 2^1$ is discouraged, as this only allows for 2 possible fingerprints per row. Given that one fingerprint is used to implement the 'empty' feature (see Section 3.5.2), $S = 2^2$ is the minimum value one can use in QHTs.

3.7 Further Improvements: QQHTD

3.7.1 Keeping Track of Duplicates

In Algorithm 3, we do not insert anything if the element is detected as a duplicate. However, following [FAK+14]'s example, we can insert it anyway, resulting in a structure we call QHT with Duplicates, or QHTD. We briefly discuss its properties.

w \ S	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
10 000 000	**96.17**	96.81	97.39	97.84	98.17	98.43	98.62	98.77	98.89
3 000 000	**88.50**	90.20	91.98	93.34	94.36	95.13	95.72	96.20	96.57
1 000 000	**71.01**	72.71	76.92	80.62	83.51	85.72	87.45	88.83	89.94
300 000	50.09	**43.67**	45.72	49.93	54.40	58.47	62.27	65.50	68.34
100 000	39.83	26.21	**23.19**	24.03	26.11	28.78	31.44	34.16	36.91
30 000	35.56	18.30	12.33	10.33	**10.06**	10.57	11.44	12.49	13.80
10 000	34.31	15.86	8.88	5.96	4.77	**4.32**	4.34	4.50	5.07
3 000	33.92	14.99	7.58	4.24	2.83	2.07	1.71	**1.39**	1.71
1 000	33.71	14.76	7.18	3.77	2.14	1.49	0.89	0.90	**0.65**
100	33.71	14.68	7.00	3.51	1.83	1.00	0.58	0.77	**0.25**

Table 3.2: Error rate (times 100) of QHTs of 1 Mb memory, $k = 1$ and various S size, for different sliding windows, on an artificial stream. The minimum value for each sliding window is in bold.

As we showed previously, the asymptotic FPR of a QHT is $\frac{k}{S}$, which was expected: each row stores k distinct fingerprints, the probability that one of them matches the fingerprint of a unique element is logically $\frac{k}{S}$. Similarly, in QHTD each row stores k fingerprints, not necessarily distinct. The probability that at least one of these fingerprints is the same than the one of an unseen element is $\text{FPR}_{\infty,\text{QHTD}} = 1 - \left(1 - \frac{1}{S}\right)^k$. Given the results of Section 3.4.1, the asymptotic $\text{FNR}_{\infty,\text{QHTD}}$ of QHTD is $\left(1 - \frac{1}{S}\right)^k$.

3.7.2 Queuing cells for a Better Sliding Window

One caveat of the QHT (and QHTD) is the fact that at any insertion, any element of the row is equally likely to be evicted: if this allows an easy FPR and FNR derivation, it makes it functioning a bit counter intuitive. As a matter of fact, one would expect a filter to first forget about the oldest elements before forgetting about the newest ones. Indeed, this behaviour matches the need of a filter operating on a *sliding window*, without taking into account oldest elements.

The solution we provide for QHT is to order the cells of a given row in a FIFO queue, which means that instead of selecting a random cell in the row for insertion, one will append the fingerprint to the end of the queue, and pop the first element (so that the size of the queue remains constant). Combined with QHTD improvement, this yields Algorithm 4.

Note that classical queues (*i.e.*, doubly chained lists) are not suited for our use, as chains require extra storage bits for pointers. Thus, for our QQHTD we create an array of k elements, in which we manually move every element at each "pop". When k is small (typically less than 5, which it usually is), the time overhead is not significant.

Algorithm 4 Queued Quotient Hash Table with Duplicates' (QQHTD) Stream

```
1: for each element e ∈ E do
2:     result ← UNSEEN
3:     Quotient of e: h(e); Fingerprint of e: f(e).
4:     for each cell b_i in the queue T[h(e)] do
5:         if (entry in b_i) = f(e) then
6:             result ← DUPLICATE
7:             break
8:     Pop the first element of the queue T[h(e)] and append f(e) at the end of
       same queue
9:     return result
```

Finally, note that a QQHTD with one cell par row is equivalent to its QHT counterpart. However, with more cells per row, QQHTDs offer a noticeable improvement over QHTs on real data streams (see Table 3.3).

3.8 Benchmarks

3.8.1 Comparison of QHT and QQHTD

In this section, we explore the difference between a QHT and a QQHTD with the same parameters, on the same stream. Note that, for $k = 1$, QHT and QQHTD are the same structure. We took filters of 65,536 bits each, on streams of 100,000 elements each. One stream issued from our 'real' dataset (10.32% of duplicates), the other a random uniform stream on an alphabet of 2^{20} elements (4.62% of duplicates). The stream parameters were chosen so that the error rates would be between 20% and 80%: a small error rate indicates that the filter is almost empty, and an high error rate would indicate saturation. Results are given in Table 3.3.

We observe that QQHTD offer a small advantage on artifical data, but a significant advantage on real datasets. Thus, while the theoretical gain of the QQHTD improvement seem hard to quantify, they are justified in practice, at least on some streams.

Stream (dup. %)	Filter	$k = 2$ $S = 8$	$k = 4$ $S = 16$	$k = 8$ $S = 32$	$k = 16$ $S = 64$
Real (10.3 %)	QHT	27.01	28.07	28.95	29.3
	QQHTD	**24.67**	**24.56**	**24.42**	**24.62**
Artificial (4.62 %)	QHT	**67.39**	74.48	79.32	**82.10**
	QQHTD	67.91	**74.41**	**79.19**	82.26

Table 3.3: Error rate (times 100) of QHTs and QQHTDs with the same parameters on different streams, depending on their parameters k and S. The best behaving filter for each run is in bold.

3.8.2 Comparison of QHT to Other Filters

We first compare QHT to other structures, in the most common setting used in other papers (*i.e.*, on an infinite sliding window).

We used filters of size ranging from 10 kb to 8 Mb. We used a real stream of URLs visited by a crawling robot, extracted with [Tan11] from the April 2018 CommonCrawl's dump [Nag18], which contained 150,000,000 elements. We also used 2 artificial streams of the same length, for which the elements where randomly generated from an alphabet of 2^{24} and 2^{27} elements respectively, leading to a duplicate rate of about 88% and 38% respectively. The source code of the benchmark is available online [6], and a standalone library for immediate use has also been developped [7].

The filters, with their parameters chosen so their asymptotic FPR was as close as possible to the arbitrary value of 25%, are:

- QHT, 1 cell per row, 3 bits per fingerprint. This specific QHT is equivalent to a QQHT or a QQHTD with the same parameters, so we do not include the latter in the benchmark.
- SQF from [DNB13], 1 cell per row, $r = 2$ and $r' = 1$
- Cuckoo Filter from [FAK+14], cells containing 1 element of 3 bits each
- Stable Bloom Filter (SBF) from [DD06], 2 bits per cell, 2 hash functions, targeted FPR of 0.02[8]
- *A*2 Filter from [Yoo10], targeted FPR of 0.1 on the sliding window.
- Block-Decaying Bloom Filter (b_DBF) from [SZ08], sliding window of 6000 elements.

Results, averaged on 5 runs, are given in Table 3.4, and summarised in Table 3.5. Note that, for better readability, the error rates have been multiplied by 100 in the table. Further more, we recall that the error rate, being defined as $ER = \text{FPR} + \text{FNR}$, is bounded by 0 below and 2 above, 1 being the error rate of a random filter. A filter can have worse results than random; for instance a filter which is always wrong has an error rate of 2.

As we can see in Table 3.5, QHT (or QQHTD) are extremely competitive and resist very well to saturation; they also appear to be the most competitive on the real stream. For instance, we observe that QHT can offer an error rate of up to 7.7 points lower than SQF, thus offering a 13% advantage, for the same amount of memory.

Moreover, we observe that b_DBF are efficient, but reach very quickly their saturation. A more detailed analysis of their FPR (see Table 3.4) shows that even

[6] `https://bitbucket.org/team_qht/qht/src/master`

[7] `github.com/mariuslp/libqht`

[8] Our benchmarks actually obtained an asymptotic FPR of around 28%, without us being able to find bugs in our implementation.

Stream (dup. %)	Memory (bits)	SQF	QHT	Cuckoo	SBF	A2	b_DBF
Real (10.3 %)	8e+06	24.61/26.64	14.00/29.78	28.23/38.26	25.10/28.98	37.72/21.21	0.00/45.22
	1e+06	24.95/30.22	14.25/34.14	28.30/40.65	25.23/31.94	38.00/24.11	0.15/45.16
	100,000	24.99/33.22	14.28/37.94	28.33/43.51	25.42/34.03	38.05/26.83	25.75/33.64
	10,000	25.00/41.45	14.29/44.46	28.37/50.49	26.00/50.29	38.01/28.41	99.58/0.20
Art. (88.82 %)	8e+06	21.87/64.62	12.02/70.74	27.80/69.14	25.94/71.85	35.22/52.84	0.00/99.96
	1e+06	24.60/73.67	14.00/83.80	28.41/71.19	26.43/73.30	37.72/60.77	0.17/99.79
	100,000	24.94/74.84	14.26/85.53	28.52/71.44	26.49/73.49	38.00/61.83	27.46/72.51
	10,000	24.98/74.99	14.28/85.69	28.55/71.47	26.50/73.50	38.03/61.97	99.67/0.31
Art. (39.79 %)	8e+06	24.42/72.09	13.86/81.52	28.39/70.82	26.40/73.00	37.54/59.32	0.00/99.99
	1e+06	24.92/74.64	14.24/85.18	28.50/71.41	26.47/73.44	38.00/61.60	0.17/99.82
	100,000	24.99/74.96	14.29/85.66	28.51/71.49	26.49/73.50	38.05/61.91	27.46/72.52
	10,000	25.00/74.99	14.28/85.72	28.52/71.49	26.49/73.51	38.02/61.97	99.69/0.31

Table 3.4: Results of filters for various streams of 150,000,000 elements (FPR/FNR in %)

Stream (dup. %)	Mem. (bits)	SQF	QHT	Cuckoo	SBF	A2	bDBF
Real (10.3 %)	8e+06	51.25	**43.78**	66.48	54.08	58.94	45.22
	1e+06	55.18	**48.38**	68.96	57.17	62.12	45.30
	100,000	58.21	**52.22**	71.83	59.44	64.88	59.39
	10,000	66.45	**58.75**	78.86	76.29	66.42	99.77
Art. (88.82 %)	8e+06	86.49	**82.76**	96.94	97.79	88.06	99.96
	1e+06	98.27	**97.80**	99.60	99.74	98.49	99.96
	100,000	**99.78**	99.79	99.96	99.98	99.84	99.96
	10,000	**99.97**	**99.97**	100.02	99.99	100.00	99.98
Art. (39.79 %)	8e+06	96.51	**95.37**	99.21	99.40	96.86	99.99
	1e+06	99.56	**99.42**	99.91	99.91	99.60	99.99
	100,000	**99.94**	99.95	100.00	99.99	99.96	99.98
	10,000	100.00	100.00	100.00	100.00	**99.99**	100.00

Table 3.5: Error rate (multiplied by 100) on real and artificial streams of 150,000,000 elements. For each stream, the most efficient result is in bold.

though their FPR is close to 0, they get an FNR close to 1. Moreover, as we see in Section 3.8.3, they are significantly slower, which can be a bottleneck for critical applications. Further more, not only are QHT/QQHTD the most efficient filter on both real and artificial streams, they are also very easily tunable, and any asymptotic FPR rate is very simply achievable. This is not the case of other filters, such as A2 or b_DBF, which require careful tuning.

3.8.3 Speed Comparison

We also benchmarked the speed of every filter on real-time detection, on a laptop with Intel i7. We used the same filters as in the previous subsection, with a memory of 1 Mb, on a stream of 150,000,000 elements. We averaged, on these filters, the time needed for $\mathsf{Insert} \circ \mathsf{Detect}$ to execute for each element. Results are shown in Table 3.6. We observe that safe for SQF, QHT is 6 times as fast as any other filter, and 10 times as fast as b_DBF. Even SQF is 50% slower than QHT. Even with additional features, QQHTD are also faster than SQF by a large magin, because fingerprint derivation is more costly in the latter. As a conclusion, we observe that QQHTD are most suited filters for high-speed analysis.

Filter	SQF	QHT	QQHTD	Cuckoo	SBF	A2	b_DBF
Time (μs)	0.423	**0.288**	0.330	2.464	1.578	1.280	2.565

Table 3.6: Amount of time (in μs) required for one iteration of Stream on each filter with 1 Mb of memory (averaged on 10 000 000 insertions). The fastest solution is in bold.

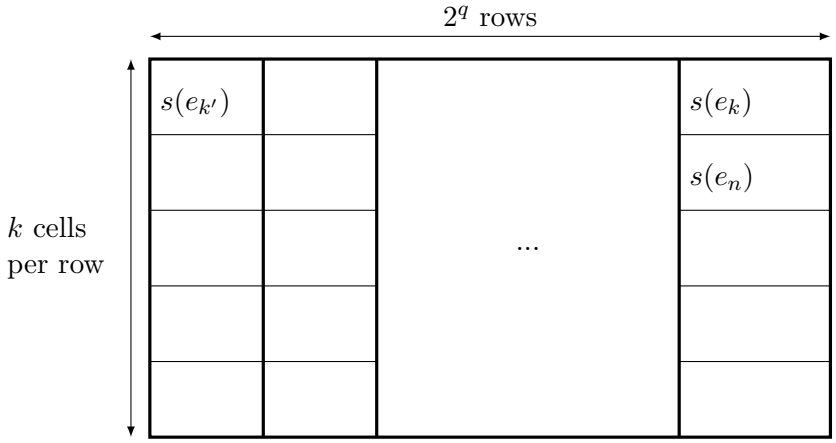


Figure 3.3: Error rate (times 100) of filters of 1Mb, depending on the size of the stream. Hatched area represents over-optimal (impossible) values, see Section 3.4.2.

3.8.4 Saturation Resistance

Another interesting point is the study of how long a filter resists to saturation on a given stream. As we know from Section 3.4.1, a filter will necessarily reach a saturation state in which it will not be more efficient than a random filter. Yet the number of insertions before this saturation is highly dependent on the filter, and a good filter will reach saturation as late as possible.

We ran a simulation of the same filters as previously, on data streams of respectively 1 000, 10 000, 100 000, 1 000 000, 10 000 000, 50 000 000, 100 000 000 and 150 000 000 elements each. One stream was extracted from our real dataset (about 10% duplicates on 150 000 000 elements), the other one is an artificial streams, with elements of 26 bits each (about 8% duplicates on 150 000 000 elements). The results are plotted in Figure 3.3.

Even though A2 filters performs slightly better on small random streams, we see that QHT are extremely competitive on real streams, as the error rate grows much slower than for other filters. Similarly, b_DBF, at their peak performance, perform as well as QHT on real streams (while being noticeably bad on artificial streams). However, b_DBF parameter tuning is quite hard and unintuitive, and to the best of our knowledge there is no easy way of finding the optimum for a given stream. Furthermore, b_DBF are 10 times as slow than QHT (see Section 3.8.3), for the same performance, in the best case.

On artificial filters, we observe that the conversion between unsaturated and saturated states happen after around one million insertions. This is related with the fact that the filters are 1 Mb big: according to information theory, a filter of 1 Mb cannot remember more than one million elements in the general case. Hence, for any filter, there it is reasonable to assume that after one million insertions, the performances are degrading as the filter starts forgetting about previous elements.

From this reasoning, we conclude that while QHT can store about 1 element per bit, some other filters, such as the SBF, cannot and thus have a limited efficiency.

3.8.5 Memory Impact

We also looked at the dual experiment, which is the impact of the filter memory on its efficiency on a given data stream. We consider this experiment to be dual of the previous one, as we saw there is a direct correlation between the memory size, the stream size and the error rate. Hence, we expect to see curves with a similar aspect as in the previous section. This prediction is validated by experience. Because we wanted to observe the 'saturation curve', we slightly changed the parameters from the previous benchmark:

- the filters are the same as in Section 3.8.2,
- we used streams of 10 000 000 elements only,
- the two streams consist of an extract of our real stream (6.88 % duplicates on the first 10 000 000 elements), and an artificial stream of elements of 26 bits (7.09% duplicates)

Note that, for practical purposes, b_DBF filters were only considered in the memory range of 10kb to 30Mb: when there is too little memory, the filter cannot operate. Where there is too much memory, the filter requires counters of more than 64 bits, which we did not implement in our benchmarks. The results are shown in Figure 3.4.

As we can see, the saturation happens when filter has fewer than 10Mb in memory, which is also the number of elements of the random stream. However, on real streams the saturation is much slower, and full saturation is not reached, even when the filters are very small.

On can also observe that there seems to be an inverse relationship between stream size and memory size. This observation is backed by the fact that we know that after m insertions, the probability of a false positive is $\mathrm{FP}_m = \frac{k}{S}\left(1 - \left[1 - \frac{1}{Nk}\right]^m\right)$. As seen in Section 3.6.4, we also have $\mathrm{FP}_m = \frac{k}{S}\left(1 - \left[1 - \frac{\log_2(S)}{M}\right]^m\right)$, and because $\log_2(S) \ll M$ in all practical cases, $\mathrm{FP}_m \approx \frac{k}{S}\left(1 - e^{-m\frac{\log_2(S)}{M}}\right)$. By consequence, the probability of a false positive is related to the ratio $\frac{m}{M}$, a ratio between the size of the stream m and the memory size M, hence the inverse relationship. Note however that such a relationship might be much harder to derive from the false negative probability (if it exists), but practical results allow us to think that the same inverse relationship is also present.

3.8.6 Influence of the Sliding Window

We use the same filters as in Section 3.8.2, with the addition of another QHT (named QHT8 in the legends), with $k = 1$ cell per row, and 8 bits per fingerprint, with 1

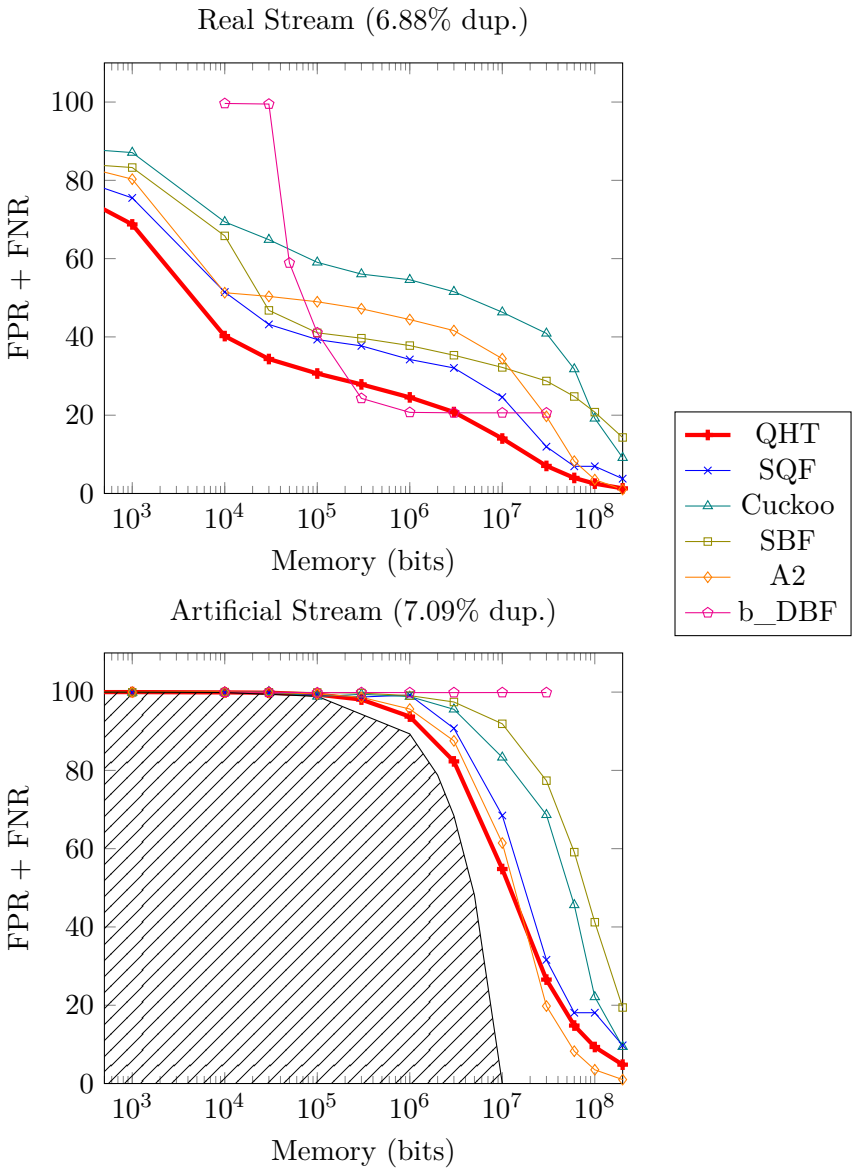


Figure 3.4: Error rate (times 100) of filters depending on the filter size, on streams of 10 000 000 elements. Hatched area represents over-optimal (impossible) values, see Section 3.4.2.

Mb memory, on two streams of 150 million elements: one stream with real data, one stream with random elements from an alphabet of 2^{26} elements.

We compare the filters error rate on different sizes of sliding windows 100, 1 000, 10 000, 100 000, 1 000 000, 10 000 000 and 100 000 000 elements.

Note that, even though included in this benchmark for the sake of completeness, sliding windows of small size (say, $< 10\ 000$ or 100 000) should not be taken into account for real-life implementation: if one really needs to find duplicates on such a small sliding window, the best solution is simply to implement a queue: the resulting structure remain time-competitive for small sizes, and offers an error rate of exactly 0.

Results are plotted in Figure 3.5. We observe that, even though b_DBF have a slight advantage on real data for real-life sliding window sizes, there is no guarantee that there will always perform as good, as their performance on the random dataset is very low. On the other hand, QHTs are better than any other structure, safe b_DBF, and are at least as competitive than b_DBF on a real case, and much better in a random dataset.

Furthermore, even though of no practical uses, QHTs can be tuned to be extremely performant on small sliding windows as well, thus offering some resilience to the user if they happen to use a probabilistic filter when it is not needed.

From this benchmark, we can make two observations. The first one is that, for all filters, there appears to have an optimum sliding window size. The second one is that b_DBF appears to be more performant than QHT in several instances. However, it appears this happens just on a small memory window: if we do the same benchmark with a different filter size, we obtain different results.

3.9 Adversarial Resistance

Now, despite the good performances of QQHTD on normal streams, one may not always assume that the stream is "normal": there may be an attacker trying to fool the filter. For instance if the filter must detect duplicates in order to avoid an attack (nonce requirements), then the question of adversarial resistance is primordial.

Theorem 3.9.1. *No filter $\mathcal{F}$ can resist a false negative attack on an infinite sliding window. More specifically, for any $0 < p < 1$, there exists $n \in \mathbb{N}$ such that $\mathcal{A}$ wins the n-false negative game for $\mathcal{F}$, with probability greater than p.*

Proof. We craft false negatives in $\Omega(M)$ steps (M being the filter's memory size). Let us remind that no structure can remember more than one element per memory bit. For this reason, the structure can remember at most M different elements. Consequently, if the attacker generates a stream of random unseen elements, then on average each element will stay for M insertions in the filter's memory. More generally, after hM insertions (for some rational $h \geq 1$), an element is forgotten with probability at least $P_{\text{forgot}} = 1 - (1 - \frac{1}{M})^{hM} \simeq 1 - e^{-h}$. Thus an attacker simply generates $\Omega(M)$ unique elements before sending the first element again. M can even be estimated via saturation (see Section 3.4.1). Given that CPU time is

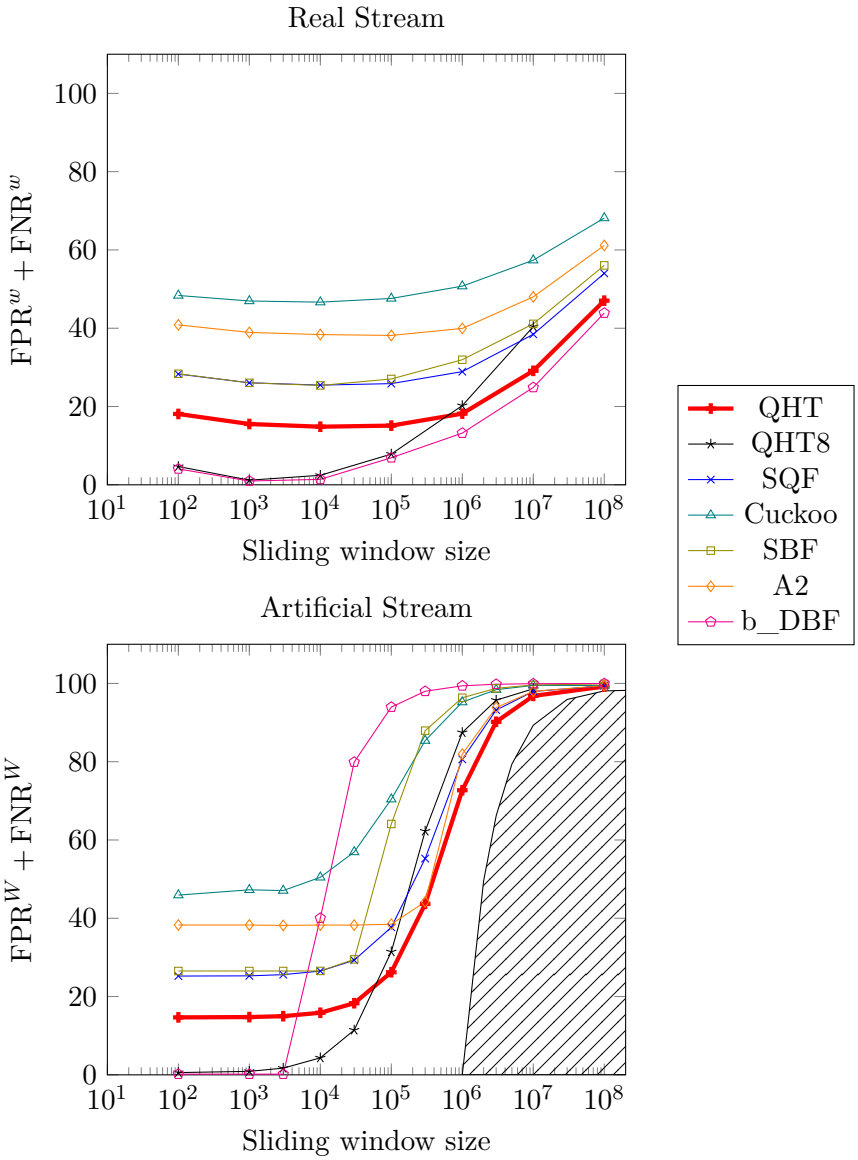


Figure 3.5: Error rate (times 100) of filters of 1Mb, depending on the size of the sliding window, on streams of 150M elements. Hatched area represents over-optimal (impossible) values, see Section 3.4.2.

cheaper than memory requirements, the attacker keeps her advantage over any filter of any size. ■

Theorem 3.9.2. *Assuming finite sliding window size w, and assuming the existence of one-way hash functions and permutations, QHT are $(1-(1-\frac{S-k}{NSk})^{w-1}, w)$-resistant to false negative attacks.*

Proof. Following [NY15] we replace all hash functions by one-way hash functions, and apply a (secret) one-way permutation on incoming elements, then classically store the results in the filter. Because of the permutation, the attacker gains no advantage in adaptively choosing the elements, thus loosing her advantage of adaptively choosing the elements. Her only advantage left is to choose when the last duplicate element of her challenge $e^\star$ has been inserted. It is obvious that the optimal strategy for $\mathcal{A}$ is to send a stream $e^\star, e_2, \ldots, e_w$ in the phase 1, before activating phase 2 on $e^\star$, with e_i being all unique elements, different than $e^\star$. Of course, $\mathcal{A}$ can switch to phase 2 as soon as she detects that she will be victorious.

In this context, $\mathcal{A}$ wins the game if and only if $e^\star$ is evicted by one of the $w-1$ next elements. An element evicts $e^\star$ with probability $\frac{1}{N}\frac{S-k}{S}\frac{1}{k}$, so $\mathcal{A}$ wins the game with probability $1-(1-\frac{S-k}{NSk})^{w-1}$, hence the result. ■

Theorem 3.9.3. *Assuming the existence of one-way functions, a QHT is $(\frac{k}{S}(1-\left(1-\frac{1}{kN}\right)^w), n)$-false positive resistant for any $n < w$.*

Proof. Similarly to the false negative proof, assuming the existence of one-way functions, the adversary has no better strategy than randomly trying new elements. The probability that a random unseen element (over the sliding window) triggers a false positive is FP_m^w, which is bounded above by $\frac{k}{S}\left(1-\left(1-\frac{1}{kN}\right)^w\right)$. ■

3.10 Conclusion

This chapter introduces a new duplicate detection filter, QHT, and its variant QQHTD. QHTs achieve a better utilization of the available space, and as such are more efficient than existing filters. Especially, QHT is always performing better than the filter it is inspired from, SQF. Moreover, QQHTD have more efficiency for detecting duplicates in a real dataset. These results apply equally when we are considering duplicates in a sliding window or in general.

We showed that, for an infinite stream with an infinite number of unseen elements, the number of rows is less important than the fingerprint space, and the number of cells per row. Moreover, we proved that all filters, having reached saturation, are not more efficient than random filters, and as such, a benchmarking of stream filters should only focus on the pre-saturation state, with small streams. We also gave a lower bound on the optimal error rate, thus allowing one to determine how close a structure is to optimality.

However, this optimality bound is not tight. Furthermore, it is probable that such an optimal structure will be extremely slow, and as such will require an efficiency/time adjustment. Further work may include research in this area.

Finally, other future research can be directed in the analysis of the error rate of QQHTDs, as well as a finer study of the parameter tuning in the sliding window configuration.

4

Towards Optimality for the wDDP

Contents

Abstract

Duplicate detection is the problem of identifying whether a given item has previously appeared in a (possibly infinite) stream of data, when only a limited amount of memory is available.

Building on the Bloom filter [Blo70], an early data structure amenable to duplicate detection, several generalizations and variants were proposed that reduce error rates while maintaining constant memory.

Unfortunately the infinite stream setting is ill-posed, and error rates of duplicate detection filters turn out to be heavily constrained: consequently they appear to provide no advantage, asymptotically, over a biased coin toss [GLPN19].

Both as a better-behaved setting and as a construction technique, a *windowed* variant of the duplicate detection problem was introduced in [SZ08; Yoo10] which consists in finding duplicate entries amongst the last w e lements of the stream for a fixed integer w.

In this chapter, we formalize the sliding window setting introduced by [SZ08; Yoo10], and show that a perfect (zero error) solution can be used up to a maximal window size $w_{\max}$. Above this threshold we show that some existing duplicate detection filters (designed for the *non-windowed* setting) perform better that those targeting the windowed problem. Finally, we introduce a "queuing construction" that improves on the performance of some duplicate detection filters in the windowed setting.

We also analyse the security of our filters in an adversarial setting.

This work is joint with Rémi Géraud–Stewart and David Naccache. It has been presented at the 26th International Computing and Combinatorics Conference (COCOON 2020), and published as [GSLPN20].

4.1 Introduction

Throughout this chapter, we are interested in the wDDP, *i.e.*, the problem of duplicate detection over a sliding window w (see Definition 2.1.5).

Note that for $w = \infty$, the problem becomes finding whether an element is a duplicate amongst all previous stream elements. For simplicity in the notation, when we refer to ∞DDP we instead write DDP.

Instances of the wDDP abound in computer science, with applications to file system indexation, database queries, network load balancing, network management [DLOM02], in credit card fraud detection [CDPLB+17], phone calls data mining [CFP+00], etc. A discussion about algorithms on large data streams can be found in [GO03].

Perfect detection is however not always reachable and it might be more practical to work on a further relaxation of the problem, allowing for errors. As such, the problem then becomes to detect duplicates, with a minimal amount of errors. Given that there are two types of error, a compromise between false positives and false negatives must be done.

Approximate duplicate detection has many real-life use cases, and can sometimes play a critical role, for instance in cryptographic schemes where all security and secrecy fall apart as soon as a random nonce is used twice, such as the ElGamal [Gam84] or ECDSA signatures. Other uses include improvements over caches [KS08], duplicate clicks [MAEA05] and others. Please note that approximate detection is a different problem than detection of approximate duplicates [ME97], in which the goal is to find elements similar but not necessarily equal to the target.

As said before, when the window size in wDDP grows infinitely large, it becomes the following problem: find whether $e^\star \in E_n$. Unfortunately any solution to this problem will necessary encounter a phenomenon called "saturation" on large enough data streams [GLPN19], and when it happens the algorithm performs no better than at random.

This is problematic on two grounds: it makes the comparison of several algorithms difficult (since they all asymptotically behave in that fashion), and the unavoidable saturation ruins any particular design's merits. As such, it is more interesting to focus on wDDP rather than DDP.

4.2 Contributions and Related Work

4.2.1 Contributions

In this chapter, we start in Section 4.3 from a naïve solution for the wDDP to then derive bounds for when it can be solved within M memory bits, up to a window size $w_{\max}$, in constant time. We then introduce, Section 4.3.2, a generalization of the naïve solution, and study its error rate. We show that this construction, which we call Short Hash Filter (SHF), can push the value $w_{\max}$ further while operating in constant time — at the cost of some errors. We also provide a different tradeoff, the Compact Short Hash Filter (CSHF), which uses fewer memory but operates in linear time.

Unfortunately, for $w > w_{\max}$ the performance of SHF degrades very rapidly.

We thus briefly turn our attention in to existing data structures designed for the "non-windowed" setting in Section 4.4, and observe how these structures behave in the sliding window context. Based on our observations, we then introduce the "queuing construction" in Section 4.5, a black box transformation of non-windowed data structures into windowed ones, that improves their performance in the wDDP setting. We give formal error rates of our construction.

We also run an exhaustive benchmark in Section 4.6. In this benchmark, we compare the efficiency of our queueing construction compared to the naïve "non-windowed" data structures. We show that depending on the context (especially the sliding window size), SHF, queueing filters and existing "non-windowed" filters each have their area of dominance.

Finally, we provide an analysis of our queueing construction's resistance to adversarial streams in Section 4.7.

4.2.2 Related Work

The notion of sliding window was, as far as we know, first introduced in [MAEA05]; but several variations exist that are incomparable to one another (*e.g.*, [SBA20]). The wDDP formulation we rely on is notably used in [Yoo10; SZ08], which also introduce algorithms for solving the wDDP approximately.

The notion of using subfilters, as in the queuing construction, can be found in the A2 filter's design [Yoo10] and a variation thereof can be found in [SBA20] but in a different DDP formulation. The A2 is built from two Bloom filters, a construction which we generalize and analyse generically in this chapter. Similarly, the construction in [SBA20] only works with Bloom Filters.

A literature review collects the following DDF constructions: A2 filters [Yoo10], Stable Bloom Filters (SBF) [DD06], Quotient Hash Tables (QHT) [GLPN19], Streaming Quotient Filters (SQF) [DNB13], Block-decaying Bloom Filters (b_DBF) [SZ08], and a slight variation of Cuckoo Filters [FAK+14] suggested by [GLPN19]. The structure proposed in [MAEA05] is not designed for wDDP but a variant called 'landmark' sliding window, which consists of a zero-resetting of the memory at some user-defined epochs.

Remark. On the topic metrics used on benchmarking of such filters, there seems to be no consensus in the literature. We measure the error rate $\mathrm{ER} = \mathrm{FPR}^w + \mathrm{FNR}^w$, as it allows a practical ranking of the solutions. An error rate of 0 means a perfect filter, while a filter answering randomly has an error rate of 1. A filter being always wrong has an error rate of 2. Note that the error rate metric is also equal to $1 - J$, where J is Youden's J statistic [You50].

4.3 Approximate Solution and SHF

4.3.1 Optimal and Approximate Optimal wDDF

Before trying to find new solutions, it is important to first observe when the problem can be solved optimally by simple constructions.

Theorem 4.3.1. *For $M \geq w(\log_2(w) + 2\log_2(|\Gamma|))$, the wDDP can be solved exactly (with no errors) in constant time.*

Proof. We explicitly construct a DDF that performs the detection. Storing all w elements in the sliding window takes $w \log_2(|\Gamma|)$ memory, using a FIFO queue Q; however lookup has a worst-time complexity of $O(w)$.

We therefore rely on an ancillary data structure for the sake of quickly answering lookup questions. Namely we use a dictionary D whose keys are elements from Γ and values are counters.

When an element e is inserted in the DDF, e is stored and $D[e]$ is incremented (if the key e did not exist in D, it is created first, and $D[e]$ is set to 1). In order to keep the number of stored elements to w, we discard the oldest element e_{last} in Q.

As we do so, we also decrement $D[e_{\text{last}}]$, and if $D[e_{\text{last}}] = 0$ the key is deleted from D. The whole insertion procedure is therefore performed in constant time.

Lookup of an element $e^\star$ is simply done by looking whether the key $D[e^\star]$ exists, which is done in constant time.

The queue size is $w \log_2 |\Gamma|$, the dictionary size is $w(\log_2 |\Gamma| + \log_2 w)$ (as the dictionary cannot have more than w keys at the same time, a dictionary key occupies $\log_2 |\Gamma|$ bits and a counter cannot go over w, thus being less than $\log_2 w$ bits long). Thus a requirement of $w(\log_2(w) + 2\log_2(|\Gamma|))$ bits for this DDF to work.

Finally this filter does not make any mistake, as the dictionary D keeps an exact account of how many times each element is present in the sliding window. ■

However, this optimal filter requires that the size of Γ is known in advance. The dependence on $\log_2 |\Gamma|$ can be dropped, at the cost of allowing errors.

Theorem 4.3.2. *Let $w \in \mathbb{N}$. Let $M \simeq 2w \log_2 w$, then the wDDP can be solved with almost no error using M memory bits.*

More precisely, it is possible to create a filter of M bits with an FN of 0, an FP of $1 - (1 - \frac{1}{w^2})^w \sim \frac{1}{w}$, and a time complexity of $O(w)$.

Using $M \simeq 5w \log_2 w$ bits of memory, a constant-time filter with the same error rate can be constructed.

Note that we only consider the false positive probability after the filter has inserted at least w elements, *i.e.*, once the filter is full and has reached a stationary regime.

Proof. Here again we explicitly construct the filters that attain the theorem's bounds.

Let h be a hash function with codomain $\{0,1\}^{2\log_2 w}$. The birthday theorem [Wag02] states that for a hash function h over a bits, one must on average collect $2^{a/2}$ input-output pairs before obtaining a collision. Therefore $2^{(2\log_2 w)/2} = w$ hash values $h(e_i)$ can be computed before having a 50% probability of a collision (here, a collision is when two distinct elements of the stream e_i, e_j with $i \neq j, e_i \neq e_j$ have the same hash, *i.e.*, $h(e_i) = h(e_j)$). The 50% threshold we impose on h is arbitrary but nonetheless practical.

Let $\mathcal{F}$ be the following DDF: the filter's state consists in a queue of w hashes, and for each new element e, $\mathsf{Detect}(e)$ returns $\mathsf{DUPLICATE}$ if $h(e)$ is present in the queue, UNSEEN otherwise. $\mathsf{Insert}(e)$ appends $h(e)$ to the queue before popping the queue.

There is no false negative, and a false positive only happens if the new element to be inserted collides with at least one other element, which happens with probability $1 - (1 - \frac{1}{2^{2\log_2 w}})^w = 1 - (1 - \frac{1}{w^2})^w$, hence an FN of 0 and a FP of $1 - (1 - \frac{1}{w^2})^w$. The queue stores w hashes, and as such requires $w \cdot 2\log_2 w$ bits of memory.

Note that this solution has a time complexity of $O(w)$. Using an additional dictionary, as in the previous proof, but with keys of size $2\log_2(w)$, we get a filter with an error rate of about $\frac{1}{w}$ and constant time for insertion and lookup, using $w \cdot 2\log_2 w + w \cdot (2\log_2(w) + \log_2(w)) = 5w \log_2 w$ bits of memory. ■

When $\log_2 |\Gamma| > 5 \log_2 w$ this DDF outperforms the naïve strategy[1], both in terms of time and memory, at the cost of a minimal error. When $\log_2 |\Gamma| > 2 \log_2 w$, it outperforms the exact solution described sooner in terms of memory.

4.3.2 Short Hash Filter and Compact Short Hash Filter Algorithms

4.3.2.1 Short Hash Filter (SHF)

The approximate filter we described uses hashes of size $2 \log_2(w)$ for a given sliding window w. However, this hash size is arbitrary, and while the current hash size guarantees a very low error rate, it can be changed. More importantly, in some practical cases the maximal amount of available memory is fixed beforehand. Fixing the memory is also more practical for benchmarking data structures, as it gives the guarantee that all filters operate under the same conditions.

This gives us the Short Hash Filter (SHF), described in Algorithm 5. The implementation relies on a double-ended queue or a ring buffer, which allows pushing at beginning of a queue and popping at the end in constant time.

Algorithm 5 SHF Setup, Lookup and Insert

```
1: function SETUP(M, w)  ▷ M is the available memory, w the size of the sliding window
2:     h ← hash function of codomain size ⌊M/(2w) − (1/2) log2 w⌋
3:     Q ← ∅                                   ▷ Q is a queue of elements of size h
4:     D ← ∅                                   ▷ D is a dictionary h ⇒ counter (of max value w)

1: function INSERT(e)
2:     Q.Push_Front(h(e))
3:     D[h(e)]++
4:     if Q.length() > w then
5:         h' ← Q.Pop_back()
6:         D[h']--
7:         if D[h'] = 0 then
8:             Erase key D[h']

1: function LOOKUP(e)
2:     if D[h(e)] > 0 then
3:         return DUPLICATE
4:     else
5:         return UNSEEN
```

4.3.2.2 Compact Short Hash Filter (CSHF)

Removing the dictionary from the SHF construction yields a more memory-efficient, but less time-efficient construction, which we dub "compact" short hash filter (CSHF). The CSHF performs in linear time in w, and is a simple queue, the only point is that

[1]The naïve strategy consisting of storing the w elements of the sliding window, requiring $w \log_2 |\Gamma|$ bits of memory.

instead of storing e, the filter stores $h(e)$, where h is a hash function of codomain size $\lfloor \frac{M}{w} \rfloor$.

4.3.2.3 Error probabilities

Let $w > 0$ be a window size and $M > 0$ the available memory.

We write $\mathrm{FN}^w_{\mathrm{SHF}}$ the probability of false negative of an SHF with these parameters. We similarly define $\mathrm{FP}^w_{\mathrm{SHF}}$, $\mathrm{FN}^w_{\mathrm{CSHF}}$, $\mathrm{FP}^w_{\mathrm{CSHF}}$.

Theorem 4.3.3. *We have:*

- $\mathrm{FN}^w_{SHF} = 0$ *and* $\mathrm{FP}^w_{SHF} = 1 - \left(1 - \sqrt{w2^{-M/w}}\right)^w$
- $\mathrm{FN}^w_{CSHF} = 0$ *and* $\mathrm{FP}^w_{CSHF} = 1 - \left(1 - 2^{-M/w}\right)^w$

Proof. This is an immediate adaptation of the proof from Theorem 4.3.2. An SHF has fingerprints of size $h = \frac{M}{2w} - \frac{1}{2}\log_2 w$, while a CSHF has fingerprints of size $h' = \frac{M}{w}$. ■

Remark: A CSHF of size M on a sliding window w has the same error rate than an SHF of sliding window w of size $2M + w\log_2 w$.

Saturation SHF has strictly increasing error probabilities, which reach a threshold of $1/2$ for some maximum window size $w_{\max}$. Beyond this value, these filters saturate extremely quickly: in other words, most SHF will either have an error rate of 0 or 1. The same can be said about the CSHF.

An illustration of this phenomenon can be seen in Figure 4.1, which shows the error rates for SHF with $M = 10^5$, against a uniformly random stream of 18-bit elements ($|\Gamma| = 2^{18}$). The benchmark used a finite stream of length 10^6.

The value $w_{\max}$ can be obtained by solving (numerically) for $\mathrm{FP}^{w_{\max}} = 1/2$ for a given M. Experiments (numerical resolution of $\mathrm{FP}^{\max} = 1/2$, for about 200 different values of M, uniformly distributed *on a log scale* between 10^2 and 10^6) indicate an approximately linear relationship between M and $w_{\max}$: $w^{\mathrm{CSHF}}_{\max} = 0.0627M + 443$ ($r^2 = 0.9981$) and $w^{\mathrm{SHF}}_{\max} = 0.0233M + 186$ ($r^2 = 0.9977$).

4.4 Non-windowed DDFs in a wDDP Setting

4.4.1 Saturation Resistance of DDFs

We now evaluate the saturation rate for several DDFs, in the original DDP setting (without sliding window). Parameters are chosen to yield equivalent memory footprints and were taken from [GLPN19], namely:

- QHT [GLPN19], 1 bucket per row, 3 bits per fingerprint;
- SQF [DNB13], 1 bucket per row, $r = 2$ and $r' = 1$;

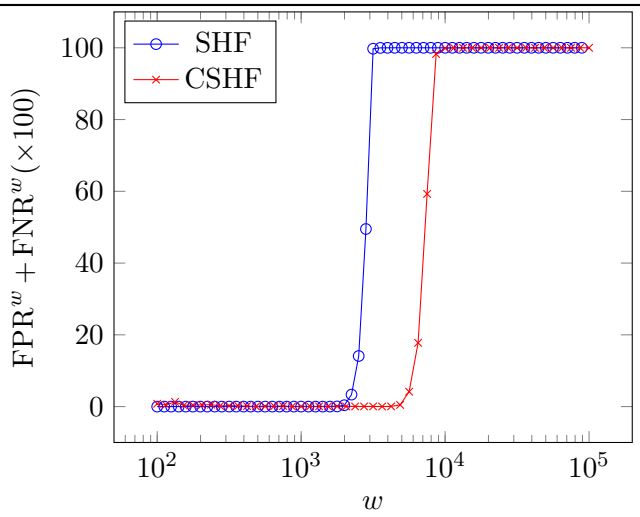


Figure 4.1: Error rates of SHFs and CSHFs for $M = 10^5$ bits, for varying window sizes w.

- Cuckoo Filter [FAK+14], cells containing 1 element of 3 bits each;
- Stable Bloom Filter (SBF) [DD06], 2 bits per cell, 2 hash functions, targeted FPR of 0.02.

The other filters from [GLPN19] were not considered in this benchmark, as they were desgined to operate on a sliding window.

These filters are run against a stream of uniformly sampled elements from an alphabet of 2^{26} elements. This results in around 8% duplicates amongst the 150 000 000 elements in the longest stream used. Results are plotted in Figure 4.2. The optimality bound has been extracted from [GLPN19].

The best results are given by the following filters, in order: QHT, SQF, Cuckoo and SBF. We also observe that QHT and SQF have error rates relatively close to the lower bound, hence suggesting that these filters are close to optimality, especially since the lower bound is not tight.

4.4.2 Performance in wDDP

We now consider the performance of the filters just discussed in the *windowed* setting, for which they were *not* designed. In particular, it is not possible to adjust their parameters as a function of w.

Remarkably, some of these filters still outperform dedicated windowed filters for some window sizes at least, as shown in Figure 4.3. In this benchmark, we used the following filters:

- block decaying Bloom Filter[2] (b_DBF) [SZ08], sliding window of size w

[2]Note that by design, a b_DBF of 10^5 bits cannot operate for $w > 6000$.

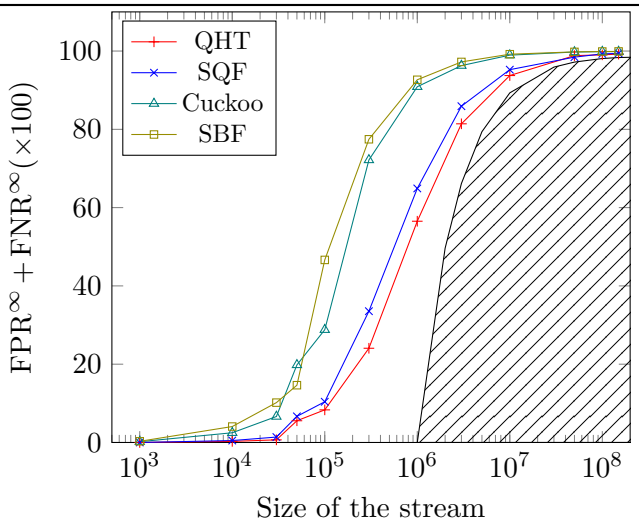


Figure 4.2: Error rate (times 100) of DDFs of 1Mb as a function of stream length. Hatched area represents over-optimal (impossible) values, see Section 3.4.2 (page 31).

- A2 filter [Yoo10], changing subfilter every $w/2$ insertions
- QHT [GLPN19], 1 bucket per row, 3 bits per fingerprint

Nevertheless, we will now discuss the queuing construction, which allow us to build windowed filters from the DDP filters.

4.5 Queuing Filters

We now describe the queuing construction, which produces a sliding window DDF from any DDF. We first give the description of the setup, before studying the theoretical error rates. A scheme describing our structure is detailed in Figure 4.4.

4.5.1 The Queuing Construction

Principle of operation. Let $\mathcal{F}$ be a DDF. Rather than allocating the whole memory to $\mathcal{F}$, we will create L copies of $\mathcal{F}$, each using a fraction of the available memory. Each of these *subfilters* has a limited timespan, and is allowed up to c insertions. The subfilters are organised in a queue.

When inserting a new element in the queuing filter, it is inserted in the topmost subfilter of the queue. After c insertions, a new empty filter is added to the queue, and the oldest subfilter is popped and erased.

As such, we can consider that each subfilter operates on a sub-sliding window of size c, which makes the overall construction a DDF operating over a sliding window of size $w = cL$.

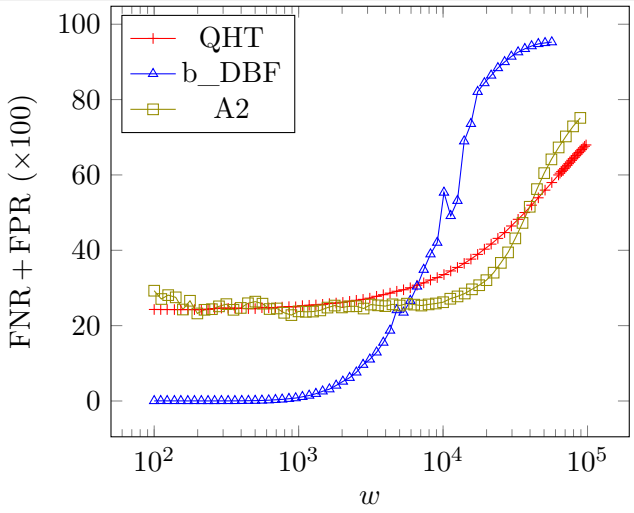


Figure 4.3: Error rates for QHT, b_DBF, and A2. While A2 and b_DBF were designed and adjusted to the wDDP, this is not the case of QHT. Still, QHT outperforms these filters for some values of w.

Insertion and lookup. The filter returns DUPLICATE if and only if at least one subfilter does. Insertion is a simple insertion in the topmost subfilter.

Queue update. After c insertions, the last filter of the queue is dropped, and a new (empty) filter is appended in front of the queue.

Pseudocode. We give a brief pseudocode for the queuing filter's functions Lookup and Insert, as well as a Setup function for initialisation, in Algorithm 6. We introduced for simplicity a constructor $\mathcal{F}$.Setup that takes as input an integer M and outputs an initialized empty filter $\mathcal{F}$ of size at most M. Here `subfilters` is a FIFO that has a `pop` and `push_first` operations, which respectevely removes the last element in the queue or inserts a new item in first position.

4.5.2 Error Rate Analysis

The queuing filter's properties can be derived from the subfilters'. False positive and false negative rates are of particular interest. In this section we consider a queuing filter $\mathcal{Q}$ with L subfilters of type $\mathcal{F}$ and capacity c (which means that the last subfilter is dropped after c insertions).

Remark. By definition, after c insertions the last subfilter is dropped. Information-theoretically, this means that all the information related to the elements inserted in that subfilter has been lost, and there are c such elements by design. Therefore, in the steady-state regime, the queuing filter holds information about at least $c(L-1)$

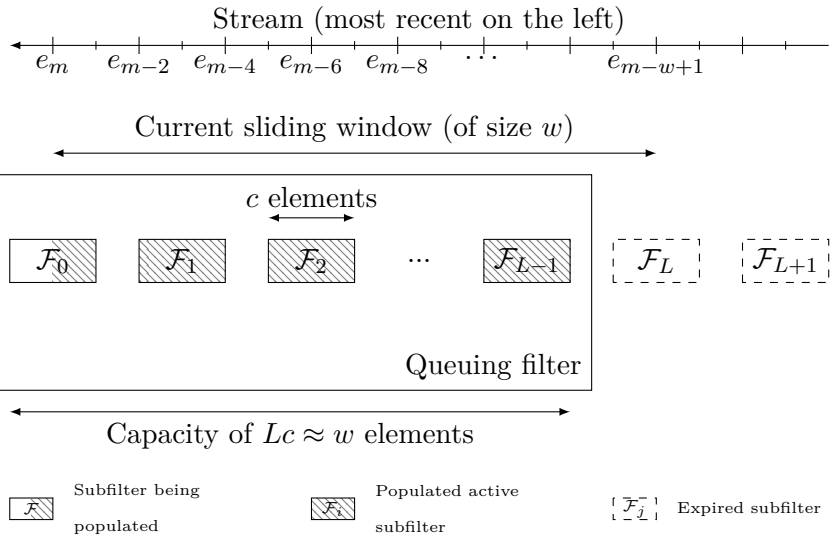


Figure 4.4: Architecture of the queuing filter, which consists of L subfilters $\mathcal{F}_i$, each containing up to c elements. Once the newest subfilter has inserted c elements in its structure, the oldest one expires. As such, the latter is dropped and a new one is created and put under population at the beginning of the queue. In this example, the sub-sliding window of $\mathcal{F}_1$ is $(e_{m-2}, e_{m-3}, e_{m-4})$.

elements (immediately after deleting the last subfilter) and at most cL elements (immediately before this deletion).

Hence, if $w < cL$, the queuing filter can hold information about *more than* w *elements*.

False Positive Probability

Theorem 4.5.1. *Let* $\mathrm{FP}^{w}_{\mathcal{Q},m}$ *be the false positive probabilityof* $\mathcal{Q}$ *after* $m > w$ *insertions, over a sliding window of size* $w = cL$*, we have*

$$\mathrm{FP}^{w}_{\mathcal{Q},m} = 1 - (1 - \mathrm{FP}_{\mathcal{F},c})^{L-1} (1 - \mathrm{FP}_{\mathcal{F},m \bmod c})$$

where $\mathrm{FP}_{\mathcal{F},m}$ *is the false positive probability of a subfilter* $\mathcal{F}$ *after* m *insertions.*

Proof. Let $E = (e_1, \ldots, e_m, \ldots)$ be a stream and $e^\star \notin_w E$.

Therefore, $e^\star$ is a false positive if and only if at least one subquery $\mathcal{F}_i.\mathsf{Lookup}(\mathsf{e}^\star)$ returns DUPLICATE. Conversely, $e^\star$ is *not* a false positive when all subqueries $\mathcal{F}_i.\mathsf{Lookup}(\mathsf{e}^\star)$ return UNSEEN, *i.e.*, when $e^\star$ is not a false positive for each subfilter.

Each subfilter has undergone c insertions, except for the first subfilter which has only undergone $m \bmod c$, we immediately get Theorem 4.5.1. ■

Algorithm 6 Queuing Filter Setup, Lookup and Insert

```
1: function Setup(F, M, L, c)        ▷ M is the available memory, F the subfilter
   structure, L the number of subfilters and c the number of insertions per subfilter
2:     subfilters ← ∅
3:     counter ← 0
4:     m ← ⌊M/L⌋
5:     for i from 0 to L − 1 do
6:         subfilters.push_first(F.Setup(m))
7:     store (subfilters, L, m, counter)
```

```
1: function Lookup(e)
2:     for i from 0 to L − 1 do
3:         if subfilters[i].Lookup(e) then
4:             return DUPLICATE
5:     return UNSEEN
```

```
1: function Insert(e)
2:     subfilters[0].Insert(e)
3:     counter++
4:     if counter == c then
5:         subfilters.pop()
6:         subfilters.push_first(F.Setup(m))
7:         c ← 0
```

Remark. In the case $w < cL$, as mentioned previously, there is a non-zero probability that $e^\star$ is in the last subfilter's memory, despite not belonging to the sliding window.

Assuming a uniformly random input stream, and writing $\delta = cL - w$, the probability that $e^\star$ has occurred in $\{e_{m-cL}, \dots e_{m-w+1}\}$ is $1 - \left(1 - \frac{1}{|\Gamma|}\right)^\delta$. For large $|\Gamma|$ (as is expected to be the case in most applications), this probability is about $\frac{\delta}{|\Gamma|} \ll 1$. Hence, we can neglect the probability that $e^\star$ is present in the filter, and we consider the result of Theorem 4.5.1 to be a very good approximation even when $w < cL$.

False Negative Probability

Theorem 4.5.2. *Let $\mathrm{FN}^w_{\mathcal{Q},m}$ be the false negative probability of $\mathcal{Q}$ after $m > w$ insertions on a sliding window of size $w = cL$, we have*

$$\mathrm{FN}^w_{\mathcal{Q},m} = u_c^{L-1} u_{m \bmod c}$$

where we have introduced the short-hand notation $u_\eta = p_\eta \,\mathrm{FN}_{\mathcal{F},\eta} + (1 - p_\eta)\,(1 - \mathrm{FP}_{\mathcal{F},\eta})$ where $\mathrm{FN}_{\mathcal{F},\eta}$ (resp. $\mathrm{FP}_{\mathcal{F},\eta}$) is the false negative probability (resp. false positive) of the subfilter $\mathcal{F}$ after η insertions, and $p_\eta = \frac{1-\left(1-\frac{1}{|\Gamma|}\right)^\eta}{1-\left(1-\frac{1}{|\Gamma|}\right)^w} \approx \frac{\eta}{w}$.

Proof. Let $E = (e_1, \dots, e_m, \dots)$ be a stream, let w be a sliding window and let $e^\star \in_w E$.

Then $e^\star$ is a false negative if and only if all subfilters $\mathcal{F}_i$ answer $\mathcal{F}_i$.Detect$(e^\star)$ = UNSEEN. There can be two cases:

- $e^\star$ is present in $\mathcal{F}_i$'s sub-sliding window;
- $e^\star$ is not present in $\mathcal{F}_i$'s sub-sliding window.

In the first case, $\mathcal{F}_i.\mathsf{Detect}(e^\star)$ returns UNSEEN if and only if $e^\star$ is a false negative for $\mathcal{F}_i$. This happens with probability $\mathrm{FN}_{\mathcal{F},c}$ by definition, except for $\mathcal{F}_0$, for which the probability is $\mathrm{FN}_{\mathcal{F},m \bmod c}$.

In the second case, $\mathcal{F}_i.\mathsf{Detect}(e^\star)$ returns UNSEEN if and only if $e^\star$ is not a false positive for $\mathcal{F}_i$, which happens with probability $1 - \mathrm{FP}_{\mathcal{F},c}$, execpt for $\mathcal{F}_0$, for which the probability is $1 - \mathrm{FP}_{\mathcal{F},m \bmod c}$.

Finally, each event is weighted by the probability p_c that $e^\star$ is in $\mathcal{F}_i$'s sub-sliding window:

$$\begin{aligned}
p_c &= \Pr[e^\star \text{ is in } \mathcal{F}_i \text{ sub-sliding window} \mid e^\star \in_w E] \\
&= \frac{\Pr[e^\star \text{ is in } \mathcal{F}_i \text{ sub-sliding window } \cap\, e^\star \in_w E]}{\Pr[e^\star \in_w E]} \\
&= \frac{\Pr[e^\star \text{ is in } \mathcal{F}_i \text{ sub-sliding window}]}{\Pr[e^\star \in_w E]} \\
&= \frac{1 - \Pr[e^\star \text{ is not in } \mathcal{F}_i \text{ sub-sliding window }]}{1 - \Pr[e^\star \notin_w E]} \\
p_c &= \frac{1 - \left(1 - \frac{1}{|\Gamma|}\right)^c}{1 - \left(1 - \frac{1}{|\Gamma|}\right)^w}
\end{aligned}$$

This concludes the proof. ■

Remark. As previously, the effect of $w < cL$ is negligible for all practical purposes and Theorem 4.5.2 is considered a good approximation in that regime.

4.5.3 FNR and FPR

From the above expressions we can derive relatively compact explicit formulas for the queuing filter's FPR and FNR when $m = cn$ for n a positive integer.

Theorem 4.5.3. *Let $\mathrm{FPR}^w_{\mathcal{Q},m}$ be the false positive rate of $\mathcal{Q}$ after $m = cn > w$ insertions on a sliding window of size $w = cL$, we have*

$$\mathrm{FPR}^w_{\mathcal{Q},cn} = 1 - \frac{(1 - \mathrm{FP}_{\mathcal{F},c})^{L-1}}{c} \sum_{\ell=0}^{c-1} (1 - \mathrm{FP}_{\mathcal{F},\ell}).$$

Proof. We start by recalling that $\mathrm{FP}^w_{\mathcal{Q},m} = \mathrm{FP}^w_{\mathcal{Q},m \bmod c}$, which we use two times in

the proof.

$$\begin{aligned}
\mathrm{FPR}^{w}_{\mathcal{Q},cn} &= \frac{1}{cn}\sum_{k=1}^{cn}\mathrm{FP}^{w}_{\mathcal{Q},k} \\
&= \frac{1}{cn}\sum_{j=0}^{n-1}\sum_{\ell=1}^{c}\mathrm{FP}^{w}_{\mathcal{Q},jc+\ell} \\
&= \frac{1}{c}\sum_{\ell=1}^{c}\mathrm{FP}^{w}_{\mathcal{Q},\ell} \\
&= \frac{1}{c}\sum_{\ell=0}^{c-1}\mathrm{FP}^{w}_{\mathcal{Q},\ell} \\
&= \frac{1}{c}\sum_{\ell=0}^{c-1}1-(1-\mathrm{FP}_{\mathcal{F},c})^{L-1}(1-\mathrm{FP}_{\mathcal{F},\ell}) \\
\mathrm{FPR}^{w}_{\mathcal{Q},cn} &= 1-\frac{1}{c}(1-\mathrm{FP}_{\mathcal{F},c})^{L-1}\sum_{\ell=0}^{c-1}(1-\mathrm{FP}_{\mathcal{F},\ell})
\end{aligned}$$

■

Theorem 4.5.4. *Let* $\mathrm{FNR}^{w}_{\mathcal{Q},m}$ *be the false negative rate of* $\mathcal{Q}$ *after* $m = cn > w$ *insertions on a sliding window of size* $w = cL$*, we have*

$$\mathrm{FNR}^{w}_{\mathcal{Q},cn} = \frac{u_c^{L-1}}{c}\sum_{\ell=0}^{c-1}u_\ell$$

Proof. Similarly, let us first recall that $\mathrm{FN}^{w}_{\mathcal{Q},m} = \mathrm{FN}^{w}_{\mathcal{Q},m \bmod c}$.

$$\begin{aligned}
\mathrm{FNR}^{w}_{\mathcal{Q},cn} &= \frac{1}{cn}\sum_{k=1}^{cn}\mathrm{FN}^{w}_{\mathcal{Q},k} \\
&= \frac{1}{cn}\sum_{j=0}^{n-1}\sum_{\ell=1}^{c}\mathrm{FN}^{w}_{\mathcal{Q},jc+\ell} \\
&= \frac{1}{c}\sum_{\ell=1}^{c}\mathrm{FN}^{w}_{\mathcal{Q},\ell} \\
&= \frac{1}{c}\sum_{\ell=0}^{c-1}\mathrm{FN}^{w}_{\mathcal{Q},\ell} \\
&= \frac{1}{c}\sum_{\ell=0}^{c-1}u_c^{L-1}u_\ell \\
\mathrm{FNR}^{w}_{\mathcal{Q},cn} &= \frac{u_c^{L-1}}{c}\sum_{\ell=0}^{c-1}u_\ell
\end{aligned}$$

■

As for the probabilities, the expressions derived above for the FNR and FNR are valid to first order in $(w - cL)/|\Gamma|$, *i.e.*, they are good approximations even when $w \approx cL$.

4.5.4 Optimising Queuing Filters

Let us relax, temporarily, the a priori constraint that $w = cL$. The parameter L determines how many subfilters appear in the queuing construction, and it might be interesting to determine what its optimal value should be. Summing up the false positive and false negative rates, we have a total error rate $\mathrm{ER}^{w}_{\mathcal{Q},cn} = 1 - \alpha\beta^{L-1} + \alpha'\beta'^{L-1}$, where $\beta = 1 - \mathrm{FP}_{\mathcal{F},c}$, $\beta' = u_c$, $\alpha = \frac{1}{c}\sum_{\ell=0}^{c-1} 1 - \mathrm{FP}_{\mathcal{F},\ell}$ and $\alpha' = \frac{1}{c}\sum_{\ell=0}^{c-1} u_\ell$ are all parameters which depend on w, c and the choice of subfilter type $\mathcal{F}$.

Because $u_\eta = p_\eta\,\mathrm{FN}_{\mathcal{F},\eta} + (1 - p_\eta)\,(1 - \mathrm{FP}_{\mathcal{F},\eta})$, differentiating with respect to L, knowing that $w = Lc$, and equating the derivative to 0, one can find the optimal value for L by solving for x the following formula, which has been obtained via Mathematica:

$$
\begin{aligned}
&-\alpha\beta^{-1+x}\log(\beta) + (\beta + \mathrm{FN}_{\mathcal{F},c}(-1+x))^{-2+x}\,x^{-x}\Big[\\
&\quad -\alpha\beta + \mathrm{FN}_{\mathcal{F},c}\left(-\beta(-2+x) + \mathrm{FN}_{\mathcal{F},c}(-1+x)\right)\\
&\quad + (\alpha + \mathrm{FN}_{\mathcal{F},c}(-1+x)) \times\\
&\qquad\qquad (\beta + \mathrm{FN}_{\mathcal{F},c}(-1+x))\left(\log\left(\beta + \mathrm{FN}_{\mathcal{F},c}(-1+x)\right) - \log(x)\right)\Big] = 0
\end{aligned}
$$

If numerically solving the equation for individual cases is feasible, it seems unlikely that a closed-form formula exists. Hence, it is maybe more interesting to find experimentally the optimal value, which we do a few pages below in Section 4.6.2.

4.5.5 Queuing Filters from Existing DDFs

Our queuing construction relies on a choice of subfilters. A first observation is that we may assume that all subfilters can be instances of a single DDF design (rather than a combination of different designs).

Indeed, a simple symmetry argument shows that a heterogenous selection of subfilters is always worse than a homogeneous one: the crux is that all subfilters play the same role in turn. Therefore we lose nothing by replacing atomically one subfilter by a more efficient one. Applying this to each subfilter we end up with a homogenous selection.

It remains to decide which subfilter construction to choose. The results of an experimental comparison of different DDFs (details about the benchmark are given in Section 4.4.1) are summarized in Figure 3.3. It appears that the most efficient filter (in terms of saturation rate) is the QHT, from [GLPN19].

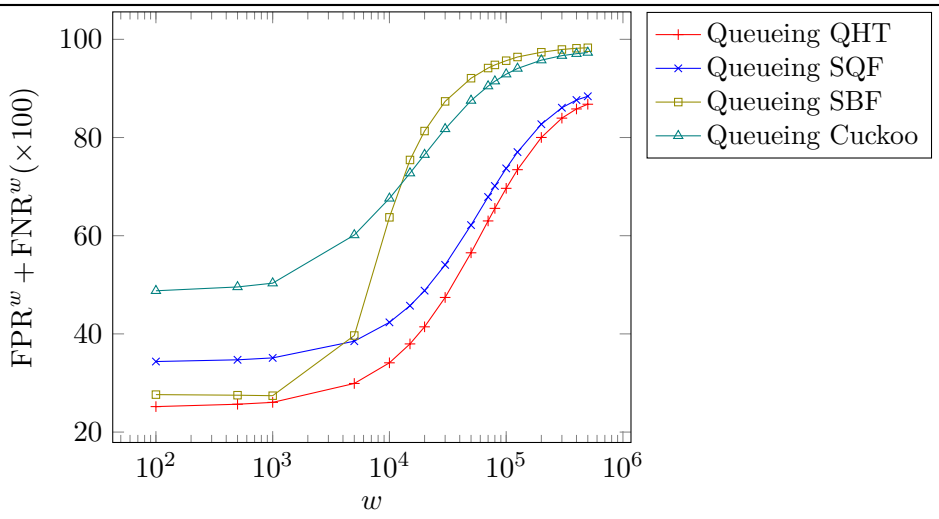


Figure 4.5: Error rate (times 100) of queuing filters as a function of window size, $M = 10^5$, $L = 10$, $|\Gamma| = 2^{18}$, on a stream of size 10^7.

4.6 Experiments and Benchmarks

This section provides details and additional information on the benchmarking experiments run to validate the above analysis. All the related code is accessible online on GitHub: `https://github.com/mariuslp/qht_v2`.

4.6.1 Benchmarking Queuing Filters

Applying the queuing construction to DDFs from the literature, we get new filters which are compared in the wDDP setting.

In Section 4.5.5 we suggested the heuristic that the DDFs with the least saturation rate in the DDP would yield the best (error-wise) queuing filter for the wDDP. This heuristic is supported by results, summarized in Figure 4.5. For this benchmark we used the following parameters: uniform stream from an alphabet of size $|\Gamma| = 2^{18}$, memory size $M = 100,000$ bits, sliding window of size $w = 10,000$, and we measure the error rate (sum of FNR^w and FPR^w).

A surprising observation is that when Lw approaches the size of the stream, there is a drop in the error. This is an artifact due to the finite size of our simulations; the stream should be considered infinite, and this drop disappears as the simulation is run for longer (see Section 4.6.4). This effect also alters the error rates for smaller window sizes, albeit much less, and we expect that filter designers care primarily about the small window regime. Nevertheless a complete understanding of this effect would be of theoretical interest, and we leave the study of this phenomenon for future work.

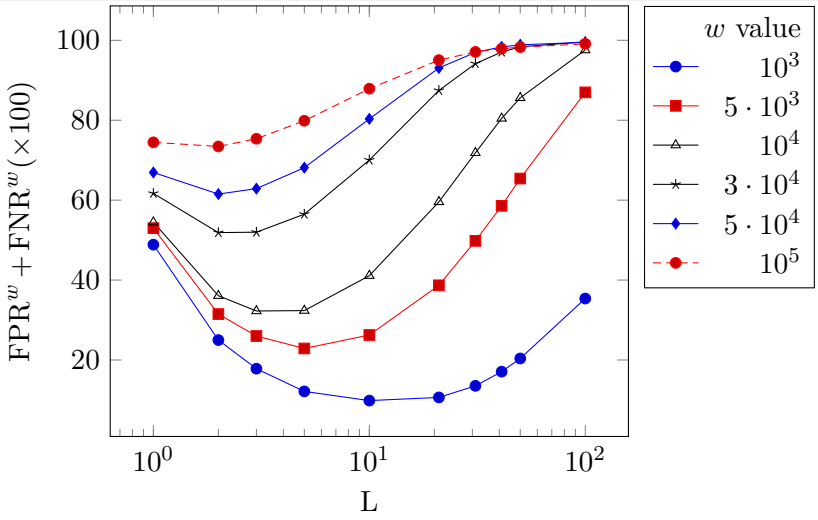


Figure 4.6: Evolution of the error rate of a queueing QHT as a function of L, for several window sizes, with $M = 10^5$, $|\Gamma| = 2^{18}$, on a stream of size 10^6.

4.6.2 The Number of Subfilters

The number of subfilters L is an important parameter in the queuing construction, as it affects the filter's error rate in a nontrivial way. An illustration of this dependence is shown in Figure 4.6 which plots the error rate of a queueing QHT on an uniform stream of alphabet size $\Gamma = 2^{16}$, with 10^5 elements in the stream, on various sliding window sizes.

We observe that the optimal value for L does indeed depend on the desired sliding window. However, other experiments on alphabets of other sizes yield very similar results, hence validating the observation made in Section 4.5.4 that the optimal number of subfilters does not depends on the alphabet, at least in first approximation.

4.6.3 Filters vs Queued Filters

Using the same stream as previously, we can build queued filters (with an optimal value L for each considered sliding window) and compare their performances to that of non-modified filters. Results on the QHT and SQF are shown in Figure 4.7, results for the Cuckoo and SBF are shown in Figure 4.8.

We observe that queueing filters do not necessarily behave better than their 'vanilla' counterparts, especially on large sliding windows. This can be interpreted by the fact that the DDPs were optimised for infinite sliding windows, and as such operate better than their queueing equivalent on large sliding windows.

4.6.4 Effects of the Simulation's Finiteness

Theoretical results about the queuing construction apply in principle to an infinite stream. However, simulations are necessarily finite, and for very large windows

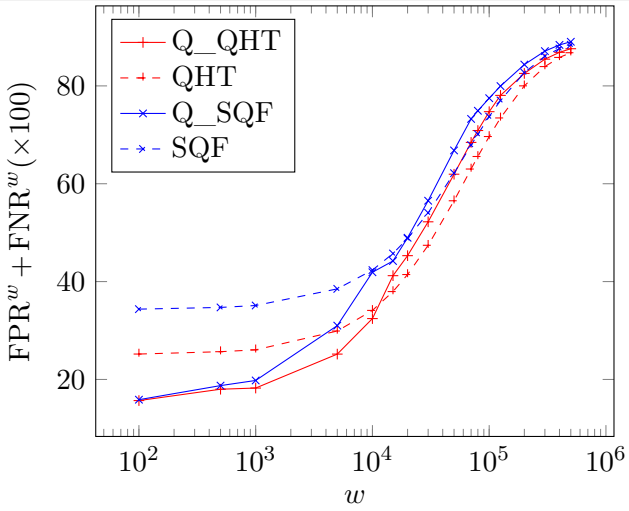


Figure 4.7: Comparing performances of QHT and SQF filters, in 'vanilla' setting or when placed in our queueing structure.

(that are approximately the same size as the whole stream) this causes interesting artefacts in the error rates.

Note that these effects have very little impact on practical implementations of queuing filters, since almost all use cases assume a window size much smaller than the stream (or, equivalently, a very large stream). Nevertheless we illustrate the effect of the finite simulation and the parameters affecting it, if only to motivate a further analytical study of this phenomenon.

Figure 4.9 (page 78) measures the error rate as a function of w, for different stream sizes N. The split between FPR and FNR is made in Figure 4.10, page 79. A visible decrease in error rate can be found around $w \approx N$. When looking in more details (see Figure 4.10),the peak of false positive, and more importantly its decrease to the asymptotic error rate of the QHT (here, $\frac{1}{7} \approx 14\%$) seems to be related with the size of the alphabet, here 2^{16}, which was confirmed by further experiments. However, these experiments showed that other parameters, such as the filter memory size, also played a role, and we did not achieve to establish a clear relation between the alphabet size, the memory size and the false positive rate. Such a relation would be an interesting addition to this work, and would help understand better how queueing filter error rates work. For high values of w, the plateau that we can see comes from the fact that, for $w > L \cdot N$, only one subfilter is used for storing the whole stream. Hence, all values $w > L \cdot N$ are identical, and equal to the error rate of an underlying filter, of size M/L, on a stream of size N.

As can be seen on this simulation, there is only disagreement around $w \approx N/L$, and increasing N results in a later and smaller peak.

It is also possible to run simulations for different alphabet sizes Γ, which shows that the peak's position increases with $|\Gamma|$, although the relationship is not obvious to quantify.

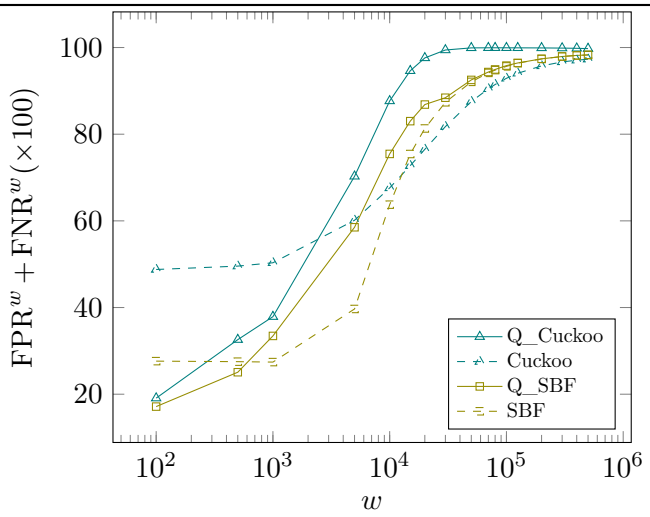


Figure 4.8: Comparing performances of the Cuckoo and SBF filters, in 'vanilla' setting or when placed in our queueing structure.

4.7 Adversarial Resistance of Queueing Filters

As DDFs have numerous security applications, we now discuss the queuing construction from an adversarial standpoint. We consider an adversarial game in which the attacker wants to trigger false positives or false negatives over the sliding window. One motivation for doing so is causing cache saturation or denial of service by forcing cache misses, triggering false alarms or crafting fradulent transactions without triggering fraud detection systems.

In this chapter, we have introduced two new constructions, the SHF and the queueing construction. However, the security of SHF is trivial to analyse. By construction, false negative attacks are impossible, and one can easily see that, assuming the existence of collision-resistance hash functions, false positive attacks are impossible as well. For this reason, in this section we only focus on the security of the queueing construction.

Theorem 4.7.1 (Bound on false positive resistance)**.** *Let $\mathcal{Q}$ be a filter of L subfilters $\mathcal{F}_i$, with c insertions maximum per subfilter, let w be a sliding window.*

If $\mathcal{F}$ is (p, c)-resistant to adversarial false positive attacks and $cL \leq w$, then $\mathcal{Q}$ is $(1-(1-p)^L, w)$-resistant to adversarial false positive attacks on a sliding window of size w.

If $cL > w$, the adversary has a success probability of at least $1-(1-p)^L$.

Proof. If $cL \leq w$, then information-theoretically the subfilters only have information on elements in the sliding window. The false positive probability for $\mathcal{Q}$ is $1-(1-\mathrm{FP}_{\mathcal{F},c})^L$, which is strictly increasing with $\mathrm{FP}_{\mathcal{F},c}$. Hence, the optimal solution is reached by to maximising the false positive probability in each subfilter $\mathcal{F}_i$. By hypothesis the latter is bounded above by p after c insertions.

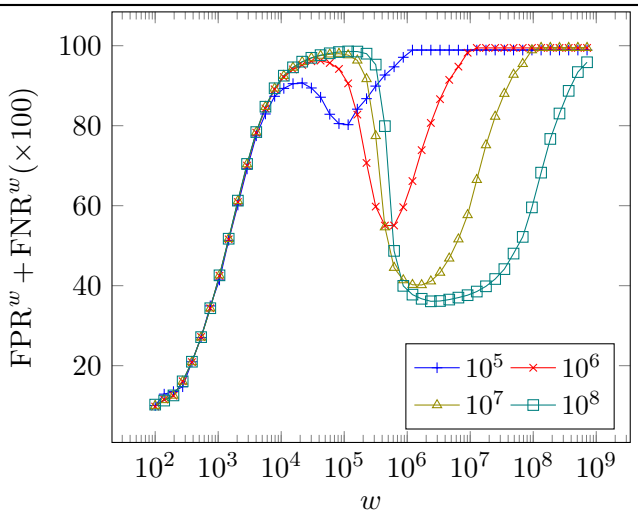


Figure 4.9: Error rate for queueing QHTs ($L = 10$, $M = 10^4$, $|\Gamma| = 2^{16}$, varying w, QHT of parameters $k = 1, s = 3$) with streams of size 10^5 to 10^8.

On the other hand, if $cL > w$ then the oldest filter holds information about elements that are not in the sliding window anymore. Hence, a strategy for the attacker trying to trigger a false positive on $e^\star$ could be to make it so these oldest elements are all equal to $e^\star$. Let E be the optimal adversarial stream for triggering a false positive on the sliding window w with the element $e^\star$, when $cL \leq w$. The adversary $\mathcal{A}$ can create a new stream $E' = e^\star|e^\star|\ldots|E$ where $e^\star$ is prepended $cL - w$ times to E.

After w insertions, the last subfilter will answer DUPLICATE with probability at least p, hence giving a lower bound on $\mathcal{A}'s$ success probability. If, for some reason, the last subfilter answers DUPLICATE with probability less than p, then the same reasoning as for when $cL \leq w$ still applies, hence we get the corresponding lower bound (which is, in this case, an equality). ■

Theorem 4.7.2 (Bounds on false negative resistance)**.** *Let $\mathcal{Q}$ be a filter of L subfilters of kind $\mathcal{F}$, with c insertions maximum per subfilter, and let w be a sliding window.*

If $\mathcal{F}$ is (p, c)-resistant to adversarial false negative attacks, then $\mathcal{A}$ can win the adversarial game on the sliding window w with probability at least p^L.

Furthermore, for q the lower bound on the false positive probability $\mathrm{FP}_{\mathcal{F},c}$ for a given stream, if $w \leq (L-1)c$ then $\mathcal{Q}$ is $(\min(1-q, p)^{L-1}p, w)$-resistant to false negative attacks on the sliding window w. On the other hand if $w > (L-1)c$ then $\mathcal{Q}$ is $(\max(1-q, p)^L, w)$-resistant to false negative attacks on the sliding window w.

Proof. Let us first prove that a PPT adversary $\mathcal{A}$ can win the game with probability at least p^L. For this, let us consider the adversarial game against the subfilter $\mathcal{F}$: after c insertions from an aversarial stream E_c, $\mathcal{A}$ choses a duplicate $e^\star$ which will be a false negative with proability p. Hence, if $\mathcal{A}$ crafts, for the filter $\mathcal{Q}$, the following

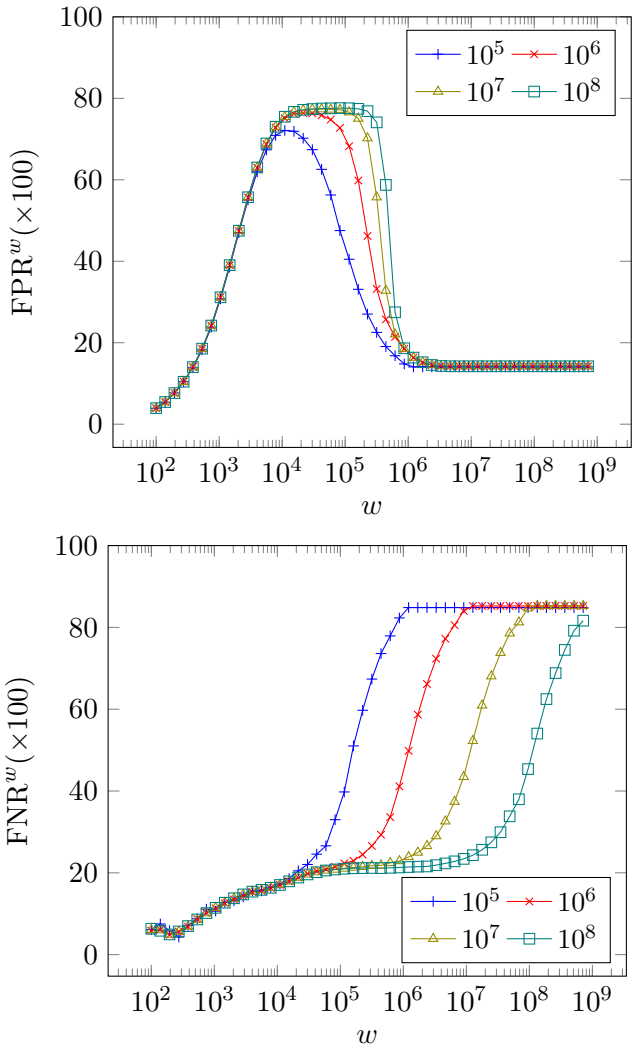


Figure 4.10: False positive and false negative curves of the previous graph, see Figure 4.9.

adversarial stream $E' = E_c \mid E_c \mid \cdots \mid E_c$ consisting of L concatenations of the stream E_c, then $e^\star$ is a false negative for $\mathcal{Q}$ if and only if it is a false negative for all subfilters $\mathcal{F}_i$, hence a success probability for $\mathcal{A}$ of p^L.

Now, Let us prove the case where $w \leq (L-1)c$. In this case, at any time, $\mathcal{Q}$ remembers all elements from inside the sliding window. As we have seen in the previous example, the success probability of $\mathcal{A}$ is strictly increasing with the probability of each subfilter to answer UNSEEN. The probability of a subfilter to answer UNSEEN is:

- $\mathrm{FN}'_{\mathcal{F},c}$ if $e^\star$ is in the subfilter's sub-sliding window;
- $1 - \mathrm{FP}'_{\mathcal{F},c}$ if $e^\star$ is not in the subfilter's sub-sliding window

where FN' and FP' are the probabilities of false negative and positives on the adversarial stream (which may be different from a random uniform stream).

However, since $e^\star$ is a duplicate, it is in at least one subfilter's sub-sliding window. As such, the optimal strategy for $\mathcal{A}$ is to maximise the probability of all subfilters to answer UNSEEN. Now, $\mathrm{FN}'_{\mathcal{F},c}$ is bounded above by p and $1-\mathrm{FP}'_{\mathcal{F},c}$ is bounded above by $1-q$, so the best strategy is where as many filters as possible answer UNSEEN with probability $\max(p, 1-q)$, knowing that at least one filter must contain $e^\star$ and as such its probability for returning UNSEEN is at most p, hence the result.

Now, let us consider the case when $w > (L-1)c$. We have already introduce the element $e^\star$ in the last w elements, and we want to insert it again. It is possible, for the adversary, to create the following stream $E = (e_1, e_2, \ldots, e_{c-1}, e^\star, e_{c+1}, \ldots, e_{Lc}, e_{Lc+1})$, and to insert $e^\star$ afterwards.

When e_{Lc+1} is inserted, all elements $(e_1, \ldots, e_{c-1}, e^\star)$ are dropped as the oldest subfilter is popped. Hence, in this context $e^\star$ is not in any subfilter anymore, so by adapting the previous analysis, $\mathcal{A}$ can get a false negative with probability at most $\max(1-q, p)^L$. ■

4.8 Conclusion

In this work, we have given bounds on the optimal solutions for the sliding window duplicate detection problem, and given solutions when optimality is reachable. When memory is too small, or conversely when the sliding window is too big, we first suggest the SHF, which experimentally is near-optimal in cases where optimality can be reached, and behaves well when the sliding window is slightly bigger than $w_{\max}$. If the sliding window is significantly bigger than $w_{\max}$, we introduce the queueing construction, which takes an existing ∞DDF and adapts it for the sliding window context. However, For even larger sliding windows, the queueing filter is actually less performant than the 'vanilla' setting, as the ∞DDF were optimised for large (or rather *infinite*) sliding windows.

Thus, we have listed different strategies for solving the wDDP. However, there is yet no guarantee of optimality, which we leave for future work.

Part II

Applied Security on Blockchain

5

About Blockchain Interoperability

Contents

Abstract

A blockchain is designed to be a self-sufficient decentralised ledger: a peer verifying the validity of past transactions only needs to download the blockchain (the ledger) and nothing else. However, it might be of interest to make two different blockchains interoperable, *i.e.*, to allow one to transmit information from one blockchain to another blockchain. In this chapter, we give a formalisation of this problem, and we prove that blockchain interoperability is impossible according to the classical definition of a blockchain. Under a weaker definition of blockchain, we demonstrate that two blockchains are interoperable is equivalent to creating a '2-in-1' blockchain containing both ledgers, thus limiting the theoretical interest of making interoperable blockchains in the first place. We also observe that all practical existing interoperable blockchain frameworks work indeed by exchanging already created tokens between two blockchains and not by offering the possibility to transfer tokens from one blockchain to another one, which implies a modification of the balance of total created tokens on both blockchains. It confirms that having interoperability is only possible by creating a '2-in-1' blockchain containing both ledgers.

This work is joint with Pascal Lafourcade and has been published in Information Processing Letters Volume 161, September 2020, as [LLP20].

5.1 Introduction

Blockchain was first introduced in 2008 by Nakamoto in [Nak09]. In their paper, the anonymous author(s) described the first decentralised ledger: a database in which anyone can write, and that is not controlled by a single or a conglomerate of identities. Since then, many other blockchains have been described: Ethereum [But14], Ripple [SYB14] and many others. In May 2019, 248 active blockchains were listed on [Cry19].

While many different blockchains exist, there is no direct way of reaching interoperability, at least without a trusted third party. Consider for instance a client willing to convert their Bitcoins to Ether: they would need to consume the amount of Bitcoins they wants to convert and to generate the equivalent amount of Ether. While Bitcoin consumption may be reachable (by sending coins to a non-existing address, such as the address 0), it is impossible to spontaneously generate Ether (or any other kind of cryptocurrency). For now, the problem is solved with the help of trusted brokers (also called *escrows*), even though other solutions are on their way [JRB19; TS15].

The issue of interoperability is solved in some cases, like "atomic exchanges" and hash-locking [But16], in which game theory ensures that a broker only benefits when following the protocol. However the question of trustless interoperability in the general context remains open.

Contributions. We introduce a theoretical background to blockchain interoperability, providing a formal definition of a blockchain and of interoperability. We then prove that, by definition, interoperability between two public blockchains is

impossible. However, we contend that there may be special conditions under which two blockchains can be interoperable. This leads us to prove the equivalence between two interoperable public blockchains and a ledger emulating both blockchains on two separate registries.

Related Work. The concept of sidechains (a sidechain is a blockchain attached to another blockchain, with exchanges possible between the two blockchains) has been explored in [BCD+14]. The authors describe a two-way peg in which a sidechain is fed with an SPV proof, a short proof of the transaction allowing for lightweight clients. The sidechain plays the role of a lightweight client, and can thus allow subsequent operations following the SPV proof. However, this pegging system requires a contest period, during which it is assumed that people will verify that the SPV proof does not come from a fork. Hence, additional trust is required in this model. In a paper from 2016 [But16], Buterin lists ways of reaching interoperability, and focuses on trusted inter-chains exchanges, where one sends money on blockchain A and receives some in blockchain B.

Similarly, the Interledger protocol [TS15] (ILP) allows one to automatize money transfers while leveraging the risk of fraud, thanks to micro-transactions. Yet, ILP is more about escrow synchronization than interoperability as we define it later on. In an ILP transaction from blockchain $\mathcal{A}$ to blockchain $\mathcal{B}$, one must find an escrow having enough money on $\mathcal{B}$ (or several escrows having in total enough money), so the transfer can occur. More generally, we consider that interoperability can for instance allow money to 'disappear' from $\mathcal{A}$ and to 'reappear' on $\mathcal{B}$, without the need for trusted escrows.

Interoperability has been notably implemented in the blockchain network Kadena [MQP18], in which transfers from one blockchain of the network to another is possible. The money is destroyed on one side and generated on the other. Kadena also uses smart contracts for securing escrow transfer. However, there is no indication that Kadena can operate with chains outside of their specific network. So in our terminology, we say that Kadena is a "N-in-1 blockchain", which is to say one blockchain, with several ledgers.

To the best of our knowledge, no theoretical work on interoperability has been done to date. Our work, rather than giving a practical implementation of an interoperable blockchain, gives a theoretical background to the topic, and explores the conceptual meaning of having interoperable blockchains.

Outline. In the next section, we formally define a blockchain and interoperability. In Section 5.3, we prove that it is impossible by design to have interoperability between blockchains. In Section 5.4, we show that interoperability is possible with a weaker definition of the blockchain. Before concluding, in Section 5.5, we prove that interoperability is equivalent to having a blockchain with two ledgers.

5.2 Preliminaries

Sets and tuples are noted in calligraphic font: $\mathcal{A}$, algorithms in serif: Mine. When a deterministic algorithm, say $\mathsf{Algorithm}$, returns some value x from some input i, we use the notation $x \leftarrow \mathsf{Algorithm}(i)$. If $\mathsf{Algorithm}$ is randomised, we use the notation $x \xleftarrow{\$} \mathsf{Algorithm}(i)$. A list of elements $e_1, \ldots, e_n$ (in this order) is represented by $[e_1, \ldots, e_n]$. We denote concatenation of two lists a and b with $a \| b$. The set of elements belonging to $\mathcal{A}$ but not to $\mathcal{B}$ is noted $\mathcal{A} \backslash \mathcal{B}$ (this set is also called the difference of $\mathcal{A}$ and $\mathcal{B}$.)

5.2.1 Blockchain Definition

Various definitions of blockchain have already been given [GKL15; AKG+18]. In this work, we rather give a formalization of blockchains, which we believe is easier to use for proving theoretical results such as the one in this chapter.

Intuitively, a blockchain is a chain of transactions. More precisely, each element of the chain (each block) contains several transactions (or one or none), as well a proof needed for consensus to take place. For instance in Bitcoin [Nak09] or similar Proof of Work blockchains, the proof is a nonce (a random number such that the hash of the block is below a threshold value); in a Proof-of-Stake such as the Casper version for Ethereum [But14] the proof consists of the successive bets on what the next block will be; in a Proof-of-Elapsed-Time as designed by Intel [IBM17], the proof is instead a certificate obtained from the SGX (a trusted enclave). Note that it exists blockchains not requiring proofs (for instance, one can argue that PBFT consensus does not require proof), in which case we consider the proof is empty.

Definition 5.2.1 (Blockchain)**.** *Let $\mathcal{T}$ be a set of transactions and $\mathcal{P}$ be a set of proofs. A blockchain is a tuple of elements $\mathcal{B} = (\mathcal{L}, \mathcal{W}, \mathsf{Emit}, \mathsf{Mine})$, where:*

- *A ledger $\mathcal{L}$ is a list of transactions with their proofs defined by: $\mathcal{L} = [([t_{1,1}, t_{1,2}, \ldots], p_1), \ldots, [([t_{n,1}, t_{n,2}, \ldots], p_n)]$ with $t_{i,j} \in \mathcal{T}$ and $p_i \in \mathcal{P}$.*
- *$\mathcal{W}$ is such that $\mathcal{W} \subset \mathcal{T}$, $\mathcal{W}$ is called the pool of waiting transactions.*
- Emit *is a deterministic algorithm taking one transaction $t \in \mathcal{T}$ and $\mathcal{W}$ as input, and returning an updated pool $\mathsf{Emit}(t, \mathcal{W}) = W \cup t$.*
- Mine *is an algorithm taking $\mathcal{L}, \mathcal{W}$ and returning a new ledger $\mathcal{L}'$, a new pool $\mathcal{W}'$, where for any $\mathcal{W} \subset \mathcal{T}$, and for $(\mathcal{L}', \mathcal{W}') \xleftarrow{\$} \mathsf{Mine}(\mathcal{L}, \mathcal{W})$, we have that $\mathcal{L}'$ is of the form $\mathcal{L} \| [(transacs, p)]$, where $transacs$ is a list containing all elements from $\mathcal{W} \backslash \mathcal{W}'$, and $p \in \mathcal{P}$ a proof.*

Furthermore, after a call to Emit *or* Mine*, the ledger $\mathcal{L}$ and the waiting pool $\mathcal{W}$ of $\mathcal{B}$ are updated with the values returned by said algorithms. In other words,* Mine *and* Emit *are not pure functions [Mil14], as they have side effects on the blockchain.*

At this point, transactions are appended (or not) to the blockchain after a call to Mine. We hereby give a formal definition of what a valid transaction is.

Definition 5.2.2 (Valid Transaction)**.** *Let* $\mathcal{B} = (\mathcal{L}, \mathcal{W}, \mathsf{Emit}, \mathsf{Mine})$ *be a blockchain, and let* t *be a transaction* ($t \in \mathcal{W}$)*,* t *is a valid transaction for* $\mathcal{B}$ *(currently in state* $\mathcal{L}$*) if and only if there exists a block in the ledger returned by* Mine *containing* t*.*

As we can see, the validity of a transaction depends on the state of the ledger; if a transaction is valid at one point, it may not be valid forever, and reciprocally. For instance, a transaction from user U to user V is valid only as long as U has enough funds. Yet, after the emission and the insertion of the transaction in the blockchain, U may issue other transactions, emptying their wallet. This is the classical issue of double spending.

The same is true for smart contracts: here, they are seen as a special subset of transactions, and they affect the state of the ledger. Because Mine has access to the whole ledger, it can take into account the smart contract's side effects.

Note that Mine is a randomized algorithm, and as such, there is no guarantee that all users will agree on the same ledger. Because blockchain is a decentralised ledger, state synchronisation must be ensured. For this, we introduce a synchronisation algorithm, called Consensus.

Definition 5.2.3 (Decentralised Blockchain)**.** *A decentralised blockchain is a tuple* $\mathcal{B}' = (\mathcal{L}, \mathcal{W}, \mathsf{Emit}, \mathsf{Mine}, \mathsf{Consensus})$ *where:*

- $\mathcal{B} = (\mathcal{L}, \mathcal{W}, \mathsf{Emit}, \mathsf{Mine})$ *is a blockchain,*
- Consensus *is a deterministic algorithm, taking as input* $\mathcal{B}$*, a set* $\mathcal{S}$ *of tuples* $(\mathcal{L}_i, \mathcal{W}_i)$ *such that* $\forall (\mathcal{L}_i, \mathcal{W}_i) \in \mathcal{S}$*, we have that* $(\mathcal{L}_i, \mathcal{W}_i, \mathsf{Emit}, \mathsf{Mine})$ *is a blockchain. Furthermore, for* $(\mathcal{L}^*, \mathcal{W}^*) \leftarrow \mathsf{Consensus}(\mathcal{B}, \mathcal{S})$*, then* $(\mathcal{L}^*, \mathcal{W}^*) \in \mathcal{S} \cup (\mathcal{L}, \mathcal{W})$*. In other words, from a list of potential new blocks,* Consensus *chooses (or accepts) one of them, or rejects them all (and returns* $(\mathcal{L}, \mathcal{W})$*).*
- *After a call to* Consensus*,* $\mathcal{B}'$*'s ledger and waiting pool components are replaced with the values returned by said algorithms.*

The idea of Consensus is that when a peer updates their local version of the blockchain, they first receive possibly more than one new version (*i.e.*, new blocks) from peers. However only one of these new blocks will be accepted, and all the network must agree on this block.

Definition 5.2.4 (Secure Blockchain)**.** *We say that a decentralised blockchain* $(\mathcal{L}, \mathcal{W}, \mathsf{Emit}, \mathsf{Mine}, \mathsf{Consensus})$ *is* secure *if it is computationally hard for a user to craft a new ledger* $\mathcal{L}'$ *and a new transaction pool* $\mathcal{W}'$ *such that for all* S *such that* $(\mathcal{L}', \mathcal{W}') \in \mathcal{S}$*, we have both that* $\mathsf{Consensus}(\mathcal{B}, \mathcal{S}) = (\mathcal{L}', \mathcal{W}')$ *and* $\mathcal{L}$ *is not a prefix of* $\mathcal{L}'$*.*

This definition makes a blockchain immune against history rewriting (and double spending), as it is computationally hard to rewrite old blocks.

5.2.2 Interoperability Definition

The concept of interoperability is to enable two blockchains to work together. A classic blockchain $\mathcal{A}$ accepts transactions because given the current state of $\mathcal{A}$'s ledger, the transaction does not violate $\mathcal{A}$'s rules. Similarly, we say that a blockchain $\mathcal{A}$ that is interoperable with blockchain $\mathcal{B}$ accepts transactions because, given the current state of $\mathcal{A}$ and $\mathcal{B}$'s ledgers, the transaction does not violate $\mathcal{A}$'s rules. Furthermore, if the rules for said transaction only imply conditions on $\mathcal{A}$'s ledger, then the transaction does not require $\mathcal{B}$ to be valid, and as such does not make use of the interoperability. So an interoperable transaction on $\mathcal{A}$ must be dependent on $\mathcal{B}$'s ledger: if $\mathcal{B}$'s ledger is equal to some values, then the transaction is valid; otherwise it is invalid.

We now give a formalization of this definition.

Definition 5.2.5 (Blockchain Interoperability)**.** *Let* $\mathcal{A} = (\mathcal{L}_A, \mathcal{W}_A, \mathsf{Emit}_A, \mathsf{Mine}_A, \mathsf{Consensus}_A)$ *and* $\mathcal{B} = (\mathcal{L}_B, \mathcal{W}_B, \mathsf{Emit}_B, \mathsf{Mine}_B, \mathsf{Consensus}_B)$ *be two decentralised blockchains. Let* Ω_A *(resp.* Ω_B*) be the set of all possible values for* $\mathcal{A}$*'s ledger* $\mathcal{L}_A$ *(resp.* $\mathcal{L}_B$*).* $\mathcal{A}$ *is* interoperable *with* $\mathcal{B}$ *if there exists:*

- *a transaction* $t \in \mathcal{T}$*,*
- *a non-empty subset* $\omega_A \subset \Omega_A$*,*
- *a non-empty proper subset* $\omega_B \subsetneq \Omega_B$

such that there exists a block containing t *that is accepted by* $\mathsf{Consensus}_A$ *if* $\mathcal{L}_A \times \mathcal{L}_B \in \omega_A \times \omega_B$*, and rejected otherwise.*

$\mathcal{A}$ *and* $\mathcal{B}$ *are* interoperable *if they are both interoperable with each other.*

5.3 General Impossibility of Interoperability

Our first result is to show that it is impossible to have interoperability between two blockchains in general.

Theorem 5.3.1. *Under the Definitions 5.2.3 and 5.2.5, blockchain interoperability is impossible.*

Proof. Assume that an interoperable transaction t exists. Then there is a set ω_B of possible ledger values of $\mathcal{B}$ for which a block containing t is accepted by $\mathsf{Consensus}_A$, if $\mathcal{L}_B \in \omega_B$. Moreover, if $\mathcal{L}_B \in \Omega_B \backslash \omega_B$, then $\mathsf{Consensus}_A$ will refuse any block containing t.

However, $\mathsf{Consensus}_A$ only takes $\mathcal{A}, S$ as arguments, where S is a set of tuples $(\mathcal{L}_i, \mathcal{W}_i)$ (see Definition 5.2.3). As a consequence, $\mathsf{Consensus}_A$ is independent from $\mathcal{B}$, and especially from $\mathcal{L}_B$. Then, if t is accepted by $\mathsf{Consensus}_A$ when $\mathcal{L}_B \in \omega_B$, then t is also accepted by $\mathsf{Consensus}_A$ when $\mathcal{L}_B \in \Omega_B \backslash \omega_B$; this implies that $\Omega_B \backslash \omega_B = \emptyset$, *i.e.*, $\omega_B = \Omega_B$, which is a contradiction with the hypothesis of Definition 5.2.5, namely that ω_B is a proper subset of Ω_B.

■

This result is actually quite straightforward if we remember that a blockchain is, by construction, made to be self-sufficient: no blockchain can rely on external data. Especially, no blockchain can rely on another blockchain for asserting the validity of a transaction. Hence, interoperability is a contradiction of one of the intrinsic characteristics of blockchain.

The interpretation of the result is as follows: without additional assumptions, interoperability between two blockchains is impossible. Therefore, to achieve interoperability further assumptions need to be made. For instance, in the two-way pegged blockchain mechanism, a dispute period is required for each interoperabilty operation; as the blockchain cannot know by itself whether the proposed SPV proof is the one of the latest block.

5.4 Interoperability with a Weaker Definition

Even though blockchain is not suited for interoperability *stricto sensu*, we can generalise our blockchain definition, in order to make a blockchain interoperable.

Hypothesis 1. *We assume that for two blockchains* $\mathcal{A} = (\mathcal{L}_A, \mathcal{W}_A, \mathsf{Emit}_A, \mathsf{Mine}_A, \mathsf{Consensus}_A)$ *and* $\mathcal{B}$*, with* $\mathcal{A}$ *both* Mine_A *and* $\mathsf{Consensus}_A$ *have access to both* $\mathcal{A}$ *and* $\mathcal{B}$*:* $\mathsf{Consensus}_A$ *is of the form* $\mathsf{Consensus}_A(\mathcal{A}, \mathcal{B}, \mathcal{S})$*, and* Mine_A *is of the form* $\mathsf{Mine}_A(\mathcal{L}_A, \mathcal{L}_B, \mathcal{W}_A, \mathcal{W}_B)$*.*

We now use the notation $\mathsf{Consensus}_A(\mathcal{A}, \mathcal{B}, \cdot)$ to note the new consensus algorithm. Hence, the 'version' of $\mathsf{Consensus}$ in the previous definition, is now noted $\mathsf{Consensus}_A(\mathcal{A}, \emptyset, \cdot)$. Similarly, the non-interoperable version of Mine is now noted as $\mathsf{Mine}_A(\mathcal{L}_A, \emptyset, \mathcal{W}_A, \emptyset)$.

Definition 5.4.1 (Interoperable transaction)**.** *Under the assumption Hypothesis 1, a transaction* t *on the blockchain* $\mathcal{A}$ *is said to be interoperable with* $\mathcal{B}$ *if* t *can be accepted by* $\mathsf{Consensus}_A(\mathcal{A}, \mathcal{B}, \cdot)$ *but cannot be accepted by* $\mathsf{Consensus}_A(\mathcal{A}, \emptyset, \cdot)$*.*

In this context, we have the following result.

Theorem 5.4.1. *Under Hypothesis 1, it is possible to build interoperable blockchains.*

Proof. Note that we already know that interoperable blockchains exist, such as Kadena [MQP18] or other blockchains listed in [JRB19], but we give an example of interoperable blockchain in our own theoretical framework.

Consider two decentralised blockchains $\mathcal{A} = (\mathcal{L}_A, \mathcal{W}_A, \mathsf{Emit}_A, \mathsf{Mine}_A, \mathsf{Consensus}_A)$ and $\mathcal{B}$. On the blockchains we define *accounts*. An account ownership is defined by the knowledge of a private key, and for simplicity the public key is assimilated to the account itself. A transaction $t \in \mathcal{T}_A$ (resp. $\mathcal{T}_B$) specifies the sender's public key, the receiver's public key, an amount and a signature of the previous fields by the sender's private key. An account i on blockchain $\mathcal{A}$ (resp. $\mathcal{B}$) is designed by i_A (resp i_B).

We build blockchain $\mathcal{B}$ so that it is interoperable with blockchain $\mathcal{A}$ in the following sense: a user can 'create' money on $\mathcal{B}$ if and only if at least the same amount of money has been consumed on $\mathcal{A}$, by sending it to a 'bin' account.

We first note that accounts owned by nobody exist. In our scheme, in most cryptosystems the public key 0 (consisting of only zeroes) is not linked to any private key. Thus, while the account 0_A exists and money can be transferred on this account, it cannot be claimed by anyone.

Let us construct B in order to fulfil the previous requirements. First, let us define the interoperability transactions $t^*(m_B, pk_B)$, which sends some amount of money m_B from 0_B to the account pk_B on $\mathcal{B}$. $t^*(m_B, pk_B)$ is only valid if there is at least one transaction on $\mathcal{L}_A$ sending m to the account 0_A, with $m > m_B$. [1]

Then, let us construct Mine_B: a transaction t is valid for $\mathsf{Mine}_B(\mathcal{L}_B, \mathcal{L}_A, \mathcal{W}_B, \mathcal{W}_A)$ if and only if t is valid for $\mathsf{Mine}_\mathsf{A}(\mathcal{L}_A, \emptyset, \mathcal{W}_A, \emptyset)$, or if both statements are true:

- t is an interoperability transaction transferring some amount of money m from 0_B to an account on $\mathcal{B}$,
- $\mathcal{L}_A$ contains a transaction sending at least m on the zero-address 0_A.

Similarly, $\mathsf{Consensus}_B(\mathcal{B}, \mathcal{A}, \cdot)$ is conceived to accept new ledgers that would have been accepted by $\mathsf{Consensus}_A(\mathcal{B}, \emptyset, \cdot)$, as well as ledgers where the new blocks are constituted solely of transactions that are accepted by $\mathsf{Consensus}_A(\mathcal{B}, \emptyset, \cdot)$ and valid interoperability transactions (valid in the meaning that at the time of their incorporation in the ledger, the sender has enough funds to emit the transaction).

With this construction, we immediately get that $\mathcal{B}$ is interoperable with $\mathcal{A}$: a user can transfer assets from $\mathcal{A}$ to $\mathcal{B}$, which is shown by the fact that some transactions (here denoted t^*) are only valid on $\mathcal{B}$ if the sender has enough funds on $\mathcal{A}$.

■

5.5 Equivalence of Interoperable Blockchains with a Single Blockchain

Even though interoperable blockchains can be tweaked into existence, we argue that they are conceptually equivalent to a single blockchain. More precisely, we argue that they are equivalent to one blockchain, composed of two ledgers. Such a blockchain can be easily implemented: if the first bit of the transaction is 0, then apply the transaction to the first ledger, and if 1 to the second.

We say that two blockchains are equivalent if any valid transaction on one blockchain corresponds to one valid transaction on the other blockchain. This definition implies that two equivalent blockchains will have very similar evolutions of their ledgers. As Mine is not deterministic, we cannot ensure that the two ledgers will be identical, but the definition we give is enough for practical uses.

[1] Note that for a real cryptocurrency more checks would be needed for any practical use, notably because of the fact that in the current setting, anyone can withdraw m from 0_B as many times as they want. However, for the sake of simplicity, we only describe a simple, naive interoperability operation here, so these checks are omitted.

Definition 5.5.1 (Blockchain equivalence)**.** *Let there be two blockchains* $\mathcal{A} = (\mathcal{L}_A, \mathcal{W}_A, \mathsf{Emit}_A, \mathsf{Mine}_A)$ *and* $\mathcal{B} = (\mathcal{L}_B, \mathcal{W}_B, \mathsf{Emit}_B, \mathsf{Mine}_B)$ *accepting transactions from* $\mathcal{T}_A$ *and* $\mathcal{T}_B$*, respectively.* $\mathcal{A}$ *and* $\mathcal{B}$ *are said to be equivalent if there exists a bijection* $\varphi : \mathcal{T}_A \to T_B$ *such that, if both ledgers are equivalent, then there is an equivalence of the valid transactions. In mathematical terms,* $\varphi(\mathcal{L}_A) = \mathcal{L}_B \Rightarrow \forall t_A \in \mathcal{T}_A, t_A$ *is a valid transaction for* $\mathcal{A} \Leftrightarrow \varphi(t_A)$ *is a valid transaction for* $\mathcal{B}$*).*

Note that $\varphi(\mathcal{L}_A)$ is the generalization of φ to ledgers: if $\mathcal{L}_A = [[t_{1,1}, t_{1,2}, \dots]\,, \dots, [t_{n,1}, t_{n,2}, \dots]]$, then $\varphi(\mathcal{L}_A) = [[\varphi(t_{1,1}), \varphi(t_{1,2}), \dots], \dots, [\varphi(t_{n,1}), \varphi(t_{n,2}), \dots]]$, in the case of a ledger without proofs. If the ledger has proofs (see Definition 5.2.1), φ would need to work on a projection of the ledger: a projection in which every poof is removed. This subtlety has been removed from the definition for the sake of simplicity.

For instance, let us assume that two blockchains are equivalent, and two smart contracts being the reciprocal image of each other. This means that whatever transaction triggers one smart contract, the effects on the state of the blockchain will be equivalent to the effects on the state of the image blockchain: in both cases, the acceptable elements after the transaction are the same (up to a bijection). This definition does not guarantee that the smart contract will behave identically: for instance, one could image a smart contract updating a useless write-only variable, which by definition does not affect the set of future acceptable transactions as it is write-only. However, it ensures that the behaviour of the blockchain is strictly the same in both cases.

Theorem 5.5.1. *A decentralised blockchain* $\mathcal{A}$ *interoperable with a blockchain* $\mathcal{B}$ *is equivalent to a decentralised blockchain* $\mathcal{C}$ *containing both* $\mathcal{A}$ *and* $\mathcal{B}$*'s ledgers.*

Proof. Let $\mathcal{A} = (\mathcal{L}_A, \mathcal{W}_A, \mathsf{Emit}_A, \mathsf{Mine}_A, \mathsf{Consensus}_A)$ and $\mathcal{B}$ be two decentralised blockchains, with $\mathcal{A}$ being interoperable with $\mathcal{B}$. $\mathcal{A}$ being interoperable with $\mathcal{B}$, we have Mine_A of the form $\mathsf{Mine}_A(\mathcal{L}_A, \mathcal{L}_B, \cdot)$, and $\mathsf{Consensus}_A$ of the form $\mathsf{Consensus}_A(\mathcal{A}, \mathcal{B}, \cdot)$.

Let $\mathcal{T}_A$ (resp. $\mathcal{T}_B$) be the set of transactions for $\mathcal{A}$ (resp. $\mathcal{B}$). Note that $\mathcal{T}_A$ contains interoperability transactions. Let $\mathcal{C}$ be the tuple $\mathcal{C} = (\mathcal{L}_C, \mathcal{W}_C, \mathsf{Emit}_C, \mathsf{Mine}_C, \mathsf{Consensus}_C)$.

We define the set of transactions for $\mathcal{C}$, $\mathcal{T}_C = (\mathcal{T}_A \times \emptyset) \cup (\emptyset \times \mathcal{T}_B)$. Let c_A (resp. c_B) be the canonical projector of $\mathcal{T}_C$ on $\mathcal{T}_A$ (resp. $\mathcal{T}_B$).

For $(\mathcal{L}_A \| [(transacs_A, p_A)], \mathcal{W}'_A) = \mathsf{Mine}_A(\mathcal{L}_A, \mathcal{L}_B, c_A(\mathcal{W}_C))$ and $(\mathcal{L}_B \| [(transacs_B, p_B)], \mathcal{W}'_B) = \mathsf{Mine}_B(\mathcal{L}_B, c_B(\mathcal{W}_C))$, we define: $\mathsf{Mine}_C(\mathcal{L}_C, \mathcal{W}_C) = (\mathcal{L}_C \| \, [(transacs_A \times \emptyset \| \emptyset \times transacs_B, p_A \times p_B),\ (\mathcal{W}'_A \times \emptyset) \cup (\emptyset \times \mathcal{W}'_B)])$

Simply put, Mine_C is a parallelisation of Mine_A and Mine_B: a block proposed by Mine_C is a block comprised of the transactions accepted by Mine_A and Mine_B.

Similarly, $\mathsf{Consensus}_C$ is built as a parallelisation of $\mathsf{Consensus}_A$ and $\mathsf{Consensus}_B$. If $\mathsf{Consensus}_A(\mathcal{A}, \mathcal{B}, c_A(\mathcal{S}_C)) = (\mathcal{L}^*_A, \mathcal{W}^*_A)$ and $\mathsf{Consensus}_B(\mathcal{B}, c_B(\mathcal{S}_B)) = (\mathcal{L}^*_B, \mathcal{W}^*_B)$, then we define $\mathsf{Consensus}_C = (\mathcal{L}_A \times \emptyset \| \emptyset \times \mathcal{L}_B, \mathcal{W}_A \times \emptyset \cup \emptyset \times \mathcal{W}_B)$.

By construction, we see that $\mathcal{C}$ is a decentralised blockchain. Furthermore, by construction each transaction accepted by Mine_C is either accepted by Mine_A or

Mine_B and, conversely, each transaction accepted by Mine_A or Mine_B is accepted by Mine_C, hence the equivalence of the blockchains.

■

In practice, Theorem 5.5.1 means that creating two interoperable blockchains is equivalent to creating one blockchain, with a ledger divided into two separate registries: a '2-in-1' blockchain. So while creating interoperable blockchains (with a lax definition of a blockchain) is possible, we argue that the conceptual interest of doing so is limited. However, it may be interesting to create an interoperable blockchain on top of an already existing blockchain. Doing so allows both blockchains to fully operate, without the older blockchain being affected by anything. Nonetheless the obvious restriction is that only one of the two blockchains will be interoperable with the other, with all the limits implied by this fact.

5.6 Conclusion

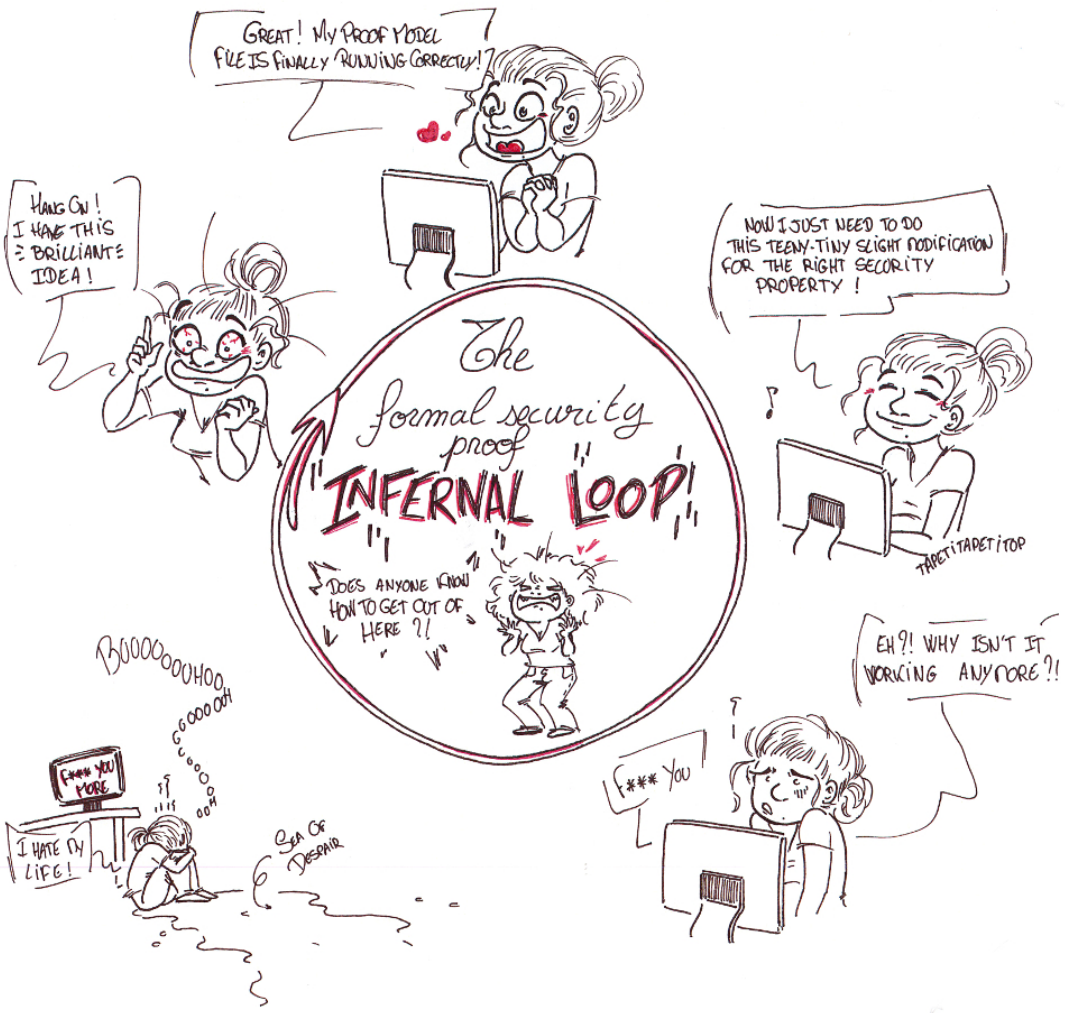
In this chapter, we explored the possibility of making two blockchains interoperable, and focused on interoperability on the consensus layer. We showed that, under classical definitions, it is impossible to make a blockchain interact with anything other than itself. If we relax the definition, we do get the possibility of interoperable blockchains, but doing so is equivalent to creating a '2-in-1' blockchain, *i.e.*, a blockchain with two ledgers. However, note that this result does not say interoperability is impossible, as we noted that solutions already exist for this problem. More importantly, our results show that 'one-way' interoperability can be achieved without losing information, which can be useful when one develops a new blockchain that can read on top of another, independent, blockchain. Moreover, in the case of full interoperability between two blockchains, the result could be interpreted as limited, as it shows that the two blockchains can be simulated by one blockchain with two ledgers. However, another way of exploring this result is to realize that a two-ledger blockchains can be split into two blockchains without loss of information. This can be useful in contexts where one ledger is highly resource consuming, while the other is lighter and is run by more actors.

It should also be noted that interoperability can be achieved at other levels than consensus, for instance using game theory on smart contracts and offchain exchanges, or relying on data oracles. Formalizing interoperability on these layers is left for future work.

Finally, we believe that the formalization we used in this chapter can be reused for studying other properties of blockchain, thus leading to formal proofs of other properties of blockchain.

Part III

Security of a Cryptographic Protocol



6

Secure Best Arm Identification in Multi-Armed Bandits

Contents

Abstract

The stochastic multi-armed bandit is a classical decision making model, where an 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 action. A popular objective for the agent is that of identifying the arm with the maximum expected reward, also known as the *best-arm identification* problem. We address the inherent privacy concerns that occur in a best-arm identification problem when outsourcing the data and computations to a *honest-but-curious* cloud.

Our main contribution is a distributed protocol that computes the best arm while guaranteeing that (i) no cloud node can learn at the same time information about the rewards and about the arms ranking, and (ii) by analyzing the messages communicated between the different cloud nodes, no information can be learned about the rewards or about the ranking. In other words, the two properties ensure that the protocol has no security single point of failure. We rely on the partially homomorphic property of the well-known Paillier's cryptosystem as a building block in our protocol. We prove the correctness of our protocol and we present proof-of-concept experiments suggesting its practical feasibility.

This work is joint with Radu Ciucanu, Pascal Lafourcade and Marta Soare. It has been presented at the 15th International Conference on Information Security Practice and Experience (ISPEC 2019) and published as [CLLP+19].

6.1 Introduction

In a stochastic multi-armed bandit model, a learning agent sequentially needs to decide which *arm* (option/action) to pull from K arms with unknown associated values available in the learning environment. After each pull, the environment responds with a feedback, in the form of a stochastic *reward* from an unknown distribution associated with the arm chosen by the agent. This is a dynamic research topic with a wide range of applications, including clinical trials for deciding on the best treatment to give to a patient [Tho33], on-line advertisements and recommender systems [LCL+10], or game playing [KS06; CM07; Mun14].

In this chapter, we focus on a popular objective in multi-armed bandits, that of *best arm identification*: given a set of K arms and a limited budget of N pulls, the goal of the agent is to design a budget-allocation strategy that maximizes the probability of identifying the arm with the maximum expected reward. This problem has been extensively studied in the machine learning community [ABM10; CLK+14; EDMM06; GGL12; KCG16; SLM14], but to the best of our knowledge, there is no previous work that considers this problem from a privacy-preserving viewpoint. Next, we illustrate the problem via a motivating example.

6.1.1 Use Case Example

A classical real-world application of the *best-arm identification* problem is as follows. Before launching a new product on the market, companies can create several versions

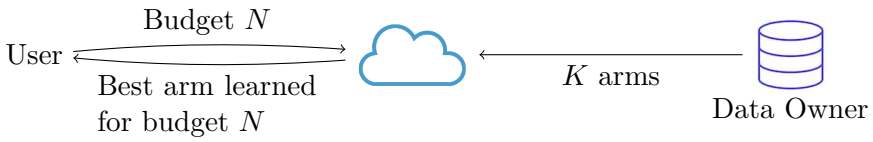


Figure 6.1: System architecture.

of the product that are put into a testing phase. By *product*, we refer here to any type of object/service that might be offered by a company and that may contain (or be obtained as a result of analyzing) private data. Each version of the product has distinguishing characteristics and the company surveys potential customers about the version they prefer. The company's objective is that once the testing phase is over, it can put on the market the version that is likely to yield the best sales. The goal of the best-arm identification problem is to define algorithms that maximize the probability of identifying the best arm (here, the best version among the K alternative versions), given a limited budget of N observations (here, customer surveys). Therefore, best-arm identification algorithms are a good fit for the product testing phase.

Now, imagine the scenario where a company collected over the years a large quantity of customer surveys that it no longer needs for its purposes. This data may actually be useful for other smaller companies that cannot afford doing their own customer surveys, but nonetheless want to simulate the test of different versions of their product. This brings us to the system architecture depicted in Figure 6.1. The *data owner* is the company that owns a large quantity of customer surveys that it wants to monetize. The *user* is the small company that wants to simulate the testing of different versions of its product, without conducting its own customer survey. It may actually be cheaper to pay a limited budget to reuse pieces of existing data, rather than doing a new survey with real customers.

The interaction between the data owner and the user is done using some *public cloud*, where initially the data owner outsources its data, then the users interact directly with the cloud. More precisely, each user allocates some budget to the cloud and reuses the available surveys for deciding which version of its own product should be put on the market. The budget would refer here to the number of survey answers the user wants the cloud to use before outputting the best option. As a simplified example, assume that the available data consists of user preferences about the characteristics of security devices they would buy for protecting their homes. There are 1M surveys available and consulting a survey costs 0.1$. If a small company wants to know which type of device people from their market are more likely to buy, the precision of the answer it receives from the cloud depends on the paid budget. If it pays 100$, it is more likely to get a clearer image about the type of device that is the most likely to be purchased, than if it pays 5$. But in both cases the obtained information is not 100% sure because only a sample of the available data is consulted.

The aforementioned use case can be easily reformulated to other real-world sce-

narios, such as health or medical data, cosmetics (*e.g.*, trials for finding the best anti-wrinkle cream), data concerning political preferences, education and employment records, to name a few.

As already mentioned, we consider a scenario where the multi-armed bandits (*i.e.*, the *data*) as well as the best arm identification algorithm (*i.e.*, the *computation*) are outsourced to some public cloud. We assume that the cloud is *honest-but-curious*: it executes tasks dutifully, and try to gain information on the ranking of the arms and their associated values from the data they receive. We address the privacy concerns that occur when outsourcing the data and computations.

Indeed, the externalized data can be communicated over an untrustworthy network and processed on some untrustworthy machines, where malicious public cloud users may learn private data that belongs only to the data owner. This is why we require the data owner to encrypt all information about the arms before outsourcing the data to the public cloud.

Moreover, each cloud user observes a result of the best arm identification algorithm that is proportional to the budget that the user pays. It should be impossible for a malicious cloud user to compose observations of several runs of the best arm identification algorithm in order to learn the best arm with a higher confidence, and then sell this information to some other user.

6.1.2 Summary of Contributions and Chapter Organization

In Section 2.2, we give background information on the problem of best-arm identification in multi-armed bandits.

In Section 6.2, we first present the considered security model, then the needed security tools, and finally the distributed security protocol, that is the main contribution of the chapter. We rely on the partial homomorphic property of Paillier's cryptosystem [Pai99] as a building block in our protocol. The difficulty of our setting comes from the fact that we need additions, multiplications, and comparisons to solve the best arm identification problem, whereas a partially homomorphic cryptosystem such as Paillier's provides only homomorphic additions. Therefore, in our protocol we distribute the computation among several participants and insure that each of them can only learn the specific information needed for performing their task, and no any other information. Thus, we show that our distributed protocol has no single point of failure, in the *honest-but-curious* cloud model. The overhead due to the security primitives of our protocol depends only on the number of arms K and not on the budget N. This is a desirable property because in practice the budget N (*i.e.*, the number of arm pulls that we are allowed to do) is often much larger than the number of arms K among which we can choose.

We prove the security of our protocol in Section 6.3, and we present proof-of-concept experiments suggesting its feasibility in Section 7.4.5. We discuss related work in Section 6.5, and conclusions and future work in Section 6.6.

6.2 Secure Protocol

6.2.1 Security Model

We assume that the reward functions associated to the arms as well as the best arm identification algorithm are outsourced to some cloud. We assume that the cloud is honest-but-curious *i.e.*, it executes tasks dutifully, but tries to extract as much information as possible from the data that it sees. The user indicates to the cloud her budget and receives the best arm that the cloud can compute using the user's budget. The user does not have to do any computation, except for eventually decrypting $\widehat{i^*}$ if she receives this information encrypted from the cloud. We expect the following security properties:

1. No cloud node can learn at the same time information about the rewards and about the ranking of the arms.
2. By analyzing the messages communicated between the different cloud nodes, no information can be learned about the rewards or about the ranking.

The two aforementioned properties essentially ensure that the desired protocol has no security single point of failure. In particular, the first property says that (i) there may be some cloud node that knows the ranking of the arms (hence also the best arm), but it is not allowed to know which rewards are associated to these arms, and (ii) there may also be some cloud node that knows some rewards, but it is not allowed to know which arms are associated to these rewards. If all cloud nodes collude, the cloud can learn the rewards associated to the arms[1]. We do not consider collusions in our model.

6.2.2 Security Background

We use Pailler's public key encryption scheme [Pai99]. We first recall the definition of public-key encryption. Pailler's encryption scheme is IND-CPA secure. We recall the definition of IND-CPA before presenting the scheme itself that has an additive homomorphic property that we use in our protocol.

6.2.3 Secure Algorithm

We revisit the successive rejects (`SR`) algorithm, presented in Section 2.2.2, page 14, in order to satisfy the properties outlined in Section 6.2.1. We consider K arms. We note $[\![n]\!]$ the set of the n first integers: $[\![n]\!] = \{1, \ldots, n\}$. Recall that `SR` has K–1 phases and at each phase j, it uses a budget of n_j to pull each of the still candidate arms. At the end of each phase, `SR` rejects the worst arm, based on all pulls observed since the beginning.

[1] In case of collusions, if several users spent successive budgets to learn the best arm among the same set of arms, the cloud could compose the observed rewards. Hence the cloud could compute the best arm using as budget the total budget of the users and leak this information to some malicious user.

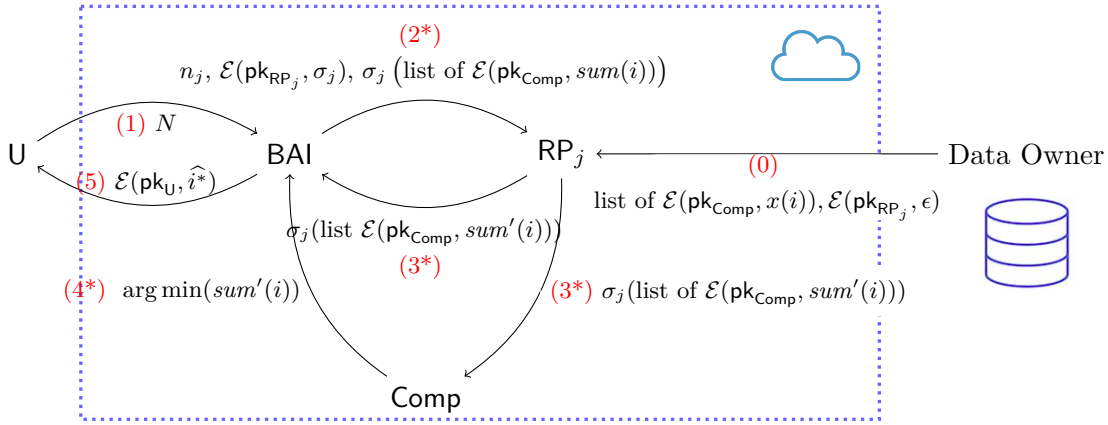


Figure 6.2: Workflow of the secure algorithm. We use numbers to indicate the order of the steps. The steps annotated with * are repeated for each phase $j \in \{1, K-1\}$. For the communications BAI $\to$ RP$_j$, RP$_j$ $\to$ BAI, and RP$_j$ $\to$ Comp, the list concerns all the arms that are still candidates *i.e.*, the set A_j.

In the sequel, each time we refer to some (pk, sk), and associated encryption/decryption functions, we assume they are done using Paillier's cryptosystem [Pai99]. In particular, we rely on the homomorphic addition property of Paillier's cryptosystem *i.e.*, $\mathcal{E}(\mathsf{pk}, x+y) = \mathcal{E}(\mathsf{pk}, x) \cdot \mathcal{E}(\mathsf{pk}, y)$.

In our security protocol, we assume $K+3$ participants:

- DO is the Data Owner, who is not in the cloud. DO sends the encrypted arm values $\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, x_i)$ and $\mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \epsilon)$ for $i \in [\![K]\!]$ and $j \in [\![K-1]\!]$.
- U is the User, a participant that is not in the cloud. The user generates $(\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}})$ and shares pk_{U} and the budget N with the cloud. The cloud nodes compute $\widehat{i^*}$ and at the end BAI sends $\mathcal{E}(\mathsf{pk}_{\mathsf{U}}, \widehat{i^*})$ to the user, who is able to decrypt it using her secret key sk_{U}.
- BAI (Best-Arm Identification) is the node responsible for executing the $K-1$ phases of the SR algorithm. BAI generates $K-1$ uniformly selected permutations σ_j of $[\![K+1-j]\!]$ (as there are $K+1-j$ candidate arms at round j). Each σ_j is shared with the node RP$_j$, but not with Comp. At each phase, BAI knows which arm is the worst and should be rejected, and after the last phase it knows which arm is the best. However, BAI does not know which rewards are associated to the arms because the rewards are encrypted with $\mathsf{pk}_{\mathsf{Comp}}$.
- Comp is the node responsible of choosing the worst one among the sums of rewards associated to the candidate arms. Comp generates $(\mathsf{pk}_{\mathsf{Comp}}, \mathsf{sk}_{\mathsf{Comp}})$ and shares $\mathsf{pk}_{\mathsf{Comp}}$ with all other cloud nodes and DO.
- RP$_1$, ... , RP$_{K-1}$ are $K-1$ nodes, each of them knowing the value ϵ that is needed to generate a reward for each arm. Each node RP$_j$ generates $(\mathsf{pk}_{\mathsf{RP}_j}, \mathsf{sk}_{\mathsf{RP}_j})$ and shares $\mathsf{pk}_{\mathsf{RP}_j}$ with BAI and DO.

The algorithm, which is summarized in Algorithm 7, consists of:

- Initialization done by BAI is:
  - Based on the total budget N, compute $n_1, \ldots, n_{K-1}$ that is the number of times each of the candidate arms should be pulled at phase $1, \ldots, K-1$, respectively.
  - Uniformly select a permutation σ_1 of $[\![K]\!]$ and send $\mathcal{E}(\mathsf{pk}_{\mathsf{RP}_1}, \sigma_1)$ to RP_1. A new permutation σ_j on $[\![K+1-j]\!]$ is randomly selected at each round, and sent to RP_j.
  - For each arm i, compute $sum[\sigma_1(i)] = \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, 0)$.

  During the $K-1$ phases of the algorithm, these encrypted sums are updated by the nodes RP_j.

- $K-1$ phases where nodes BAI, RP_j, and Comp interact as shown in Figure 6.2. We add the following specifications:
  - Each RP_j updates the encrypted sums using the homomorphic addition property of the Paillier's cryptosystem: for a round j and a candidate arm i with sum $sum[\sigma_j(i)]$, we get the updated sum $sum'[\sigma_j(i)]$ by homomorphically adding $\left(\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, x(\sigma_j(i)))\right)^{n_j} \times \prod_{l=1}^{n_j} \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, k_l)$ to $sum[\sigma_j(i)]$, where k_l is uniformly selected in $[-\epsilon, \epsilon]$ by RP_j.
  - When Comp computes the index of the worst arm, if two or more arms have the same worst sum of rewards, then Comp selects uniformly at random one of these arms as the worst one. This ensures that the index of the worst arm has a uniform distribution.

We summarize in Table 6.1 what each cloud node knows and does not know, in order to satisfy the desired security properties. We formally prove the security properties in Section 6.3. Next, we briefly outline why we need so many nodes:

- Assuming that all RP_j nodes are a single one, this node would know all rewards since the beginning of the algorithm hence it would learn the ranking of the arms.
- Assuming that Comp and RP_j are the same, then it would leak which arm is associated to which sum, hence the best arm could be leaked.
- Assuming that Comp and BAI are the same, then BAI would learn the plain rewards in addition to the ranking that it already knows.
- Assuming that BAI and RP_j are the same, then it would leak to BAI the sum of rewards associated to each arm.

Node	BAI	Comp	RP_j
Does know	• ranking of arms (including best arm)	• sums of rewards	• arms still candidate at phase j • arms already rejected before phase j • sums of rewards added at phase j
Does not know	• sums of rewards of any arm (Theorem 6.3.1)	• mapping between sums of rewards and the arms that produced them (Theorem 6.3.3) • ranking of arms (including best arm) (Theorem 6.3.2)	• ranking of arms (including best arm) (Theorems 6.3.4 and 6.3.5) • sums of rewards from phases $1, \dots, j-1, j+1, \dots, K-1$ (Theorem 6.3.6)

Table 6.1: What each cloud node knows and does not know.

6.2.4 Complexity

We give here a brief description of the complexity, in terms of the number of calls to $\mathcal{E}$ and $\mathcal{D}$ (the costliest operations).

- At Step 0, DO computes $\forall i \in [\![K]\!], \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, x(i))$. It also encrypts ϵ for each RP_j, thus having $O(K)$ complexity.
- At Step 2, BAI computes a new encrypted permutation, that can be encoded as $[\mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \sigma_j(1)), \dots, \mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \sigma_j(K+1-j)]$, thus having $O(K-j) = O(K)$ complexity.
- At Step 3, RP_j computes the added rewards. Given the algorithm in Section 6.2.3, this step has $O(K)$ complexity.
- At Step 4, Comp decrypts all partial sums, with a complexity of $O(K)$, before sending the argmin to BAI.

Steps 2, 3, 4 are repeated $K-1$ times. The total complexity of these three steps is then $O(K^2)$, and the total complexity of the algorithm is $O(K^2)$.

Note that the complexity of the algorithm is independent from the total budget N, which is a great advantage as typical budgets for these kinds of problems are often elevated and usually much larger than the number of arms. More precisely, the complexity related to N is hidden by the complexity of the encryptions.

6.3 Security Proofs

In this section, we prove that our secure algorithm presented in Section Section 6.2.3 satisfies the two desirable security properties outlined in Section 6.2.1: we prove

the first property from Section 6.3.2 to Section 6.3.4, and the second property in Section 6.3.5.

6.3.1 Notations and Security Hypothesis

For a node A, we note $data_{\mathsf{A}}$ the data to which A has access and $\mathcal{A}^{pb}(d)$ the answer of a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ having knowledge of d, trying to solve the problem pb. We recall that, in our notation conventions, $[\![K]\!]$ denotes the set of positive integers lower than or equal to K: $[\![K]\!] = \{1, \ldots, K\}$.

Lemma 6.3.1. *For a list $l = [l_1, \ldots, l_n]$, a permutation σ and the permuted list $\sigma(l) = [l_{\sigma(1)}, \ldots, l_{\sigma(n)}]$, a PPT adversary $\mathcal{A}(l_\sigma)$ cannot invert one element with probability better than random. More specifically,*

$$P\left[\mathcal{A}^{\sigma^{-1}}(\sigma(l)) \in \{i, \sigma^{-1}(i)\}_{i \in [\![K]\!]}\right] = \frac{1}{K}$$

where $\mathcal{A}$ returns a tuple $(i, g(i))$ and $g(i)$ is $\mathcal{A}$'s guess for the preimage of i.

Proof. This is immediate, as all preimages are equally likely if σ is uniformly selected. ■

Lemma 6.3.2. *Let $\mathcal{A}$ be a PPT adversary. Consider the adversarial game in which $\mathcal{A}$ choses three messages m_0, m_1, z and sends them to the challenger $\mathcal{C}$. $\mathcal{C}$ choses a random bit b, and returns a tuple (c_0, c_1, s) where $c_0 = \mathcal{E}(\mathsf{pk}, m_0), c_1 = \mathcal{E}(\mathsf{pk}, m_1)$, and $s = \mathcal{E}(\mathsf{pk}, m_b + z) = c_b \cdot \mathcal{E}(\mathsf{pk}, z)$. $\mathcal{A}$ must then guess the value of b.*

If $\mathcal{E}(\mathsf{pk}, \cdot)$ is IND-CPA secure, then $\mathcal{A}$ does not have any advantage in this adversarial game:

$$\left| P\left[\mathcal{A}(c_0, c_1, s) = b\right] - \frac{1}{2} \right| < negl(\lambda)$$

Proof. Assume there is a PPT adversary $\mathcal{A}$ able to win the game with significant advantage $x + \text{negl}(\lambda)$: then $\mathcal{A}$ can guess b with probability $\frac{1}{2} + \frac{x}{2} + \text{negl}(\lambda)$. We then prove that an PPT adversary $\mathcal{B}$ can break the IND-CPA property of Paillier. We can assume that when $\mathcal{A}$ is given c_0, c_0', s as input (where c_0' is another encryption of m_0), then the advantage of $\mathcal{A}$ is negligible: this gives us a lower bound of the advantage of $\mathcal{A}$ in a more general adversarial game.

Let us consider an IND-CPA game and an adversary $\mathcal{B}$, in which $\mathcal{B}$ choses m_0, m_1 and sends them to the challenger. The challenger randomly selects the bit b and sends back $c_b = \mathcal{E}(\mathsf{pk}, m_b)$. Then, $\mathcal{B}$ selects a message z and computes $\mathcal{E}(\mathsf{pk}, z)$, before computing $s = \mathcal{E}(\mathsf{pk}, m_b) \cdot \mathcal{E}(\mathsf{pk}, z)$. $\mathcal{B}$ also computes $c_0' = \mathcal{E}(\mathsf{pk}, m_0)$. Then, $\mathcal{B}$ calls $\mathcal{A}(c_0', c_b, s)$, retrieves (in polynomial time) from $\mathcal{A}$ the guessed value b^*, and returns b^*.

If $b = 0$, then $\mathcal{B}$ has actually called $\mathcal{A}(c_0', c_0, s)$, which guesses the correct b^* with probability $\frac{1}{2} + \text{negl}(\lambda)$. On the other hand, if $b = 1$, then $\mathcal{B}$ has actually called $\mathcal{A}(c_0', c_1, s)$, which gives the correct b^* with probability $\frac{1}{2} + \frac{x}{2} + \text{negl}(\lambda)$.

b being randomly chosen, $\mathcal{B}$ correctly guesses b with probability $\frac{1}{2}\cdot\frac{1}{2}+\frac{1}{2}\cdot\left(\frac{1}{2}+\frac{x}{2}\right)+\text{negl}(\lambda)=\frac{1}{2}+\frac{x}{4}+\text{negl}(\lambda)$, thus yielding to $\mathcal{B}$ an advantage of $\frac{x}{4}+\text{negl}(\lambda)$ in the IND-CPA game, in polynomial time. This is a contradiction with the fact that Paillier is IND-CPA secure. ■

6.3.2 Security Proofs for BAI

Lemma 6.3.3. *From the data obtained at round j, a honest-but-curious* BAI *does not know the sum of the rewards of any arm. More precisely, for R the set of possible rewards,*

$$\left|P\left[\mathcal{A}^{reward}(data_{\mathsf{BAI}^j})\in\{i,reward(i)\}_{i\in[\![K+1-j]\!]}\right]-\frac{1}{|R|}\right|<negl(\lambda)$$

where $A^{reward}(data_{\mathsf{BAI}^j})$ returns a tuple $(i,g_{reward}(i)))$, with $g_{reward}(i)$ being $\mathcal{A}$'s guess of the sum of rewards of i.

Proof. At round j, BAI has access to the permuted list of the encrypted partial sums, as well as to the permutation σ_j and the index i_{min_j} of the lowest-ranking element from round j. From the first two arguments, BAI can access to the (unpermuted) list of the encrypted partial sums of the arms rewards $se_j=\Big[\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},sum_{\alpha_1}),\ldots,\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},sum_{\alpha_{K+1-j}})\Big]$, where the α_i are the arms still present in the algorithm at step j. So we can equivalently say that $data^j_{\mathsf{BAI}}=[se_j,i_{min_j}]$.

Assume that there exists a PPT adversary $\mathcal{A}$ able to break the above inequality, with advantage $x+\text{negl}(\lambda)$: given $[se,i_{min_j}]$ as input, $\mathcal{A}$ returns some tuple (i,g_{reward}) where g_{reward} is the guessed reward of the arm α_i. The guess is correct with probability $\frac{1}{|R|}+x+\text{negl}(\lambda)$. Also note that, on average, $i=1$ with probability $\frac{1}{K+1-j}$.

Let us consider a classical IND-CPA game as previously defined. $\mathcal{B}$ first simulates an execution of blind bandits, with arm rewards of its own choice. From this, $\mathcal{B}$ gets a list of simulated sums $s\hat{u}m_1,\ldots,s\hat{u}m_{K+1-j}$ and the lowest ranking arm $\hat{i}_{min_j}$.

$\mathcal{B}$ selects a random value $s\hat{u}m_0$ different from all sums, sends the tuple $(\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},s\hat{u}m_0),\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},s\hat{u}m_1))$ to the challenger, and receives $\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},s\hat{u}m_b)$.

Then, $\mathcal{B}$ calls $\mathcal{A}(\Big[\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},s\hat{u}m_b),\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},s\hat{u}m_2),\ldots,\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}},s\hat{u}m_K+1-j)\Big],\hat{i}_{min_j})$ which returns (i,g_{reward}). If $i=1$ and $g_{reward}=s\hat{u}m_1$ then $\mathcal{B}$ returns 1. Otherwise $\mathcal{B}$ returns a random guess.

$\mathcal{B}$ is correct in the following cases:

- $i=1$, $b=1$ and $\mathcal{A}$ guesses correctly. This event happens with probability $\frac{1}{K+1-j}\frac{1}{2}\left(\frac{1}{|R|}+x\right)$.
- $i=1,b=1$, $\mathcal{A}$ does not return $s\hat{u}m_1$ and $\mathcal{B}$ makes a correct random guess. This happens with probability $\frac{1}{K+1-j}\frac{1}{2}\left(1-\frac{1}{|R|}-x\right)\frac{1}{2}$.

- $i = 1, b = 0$, $\mathcal{A}$ does not return $\hat{sum}_1$, and $\mathcal{B}$ makes a correct random guess. This happens with probability $\frac{1}{K+1-j}\frac{1}{2}\left(1 - \frac{1}{|R|}\right)\frac{1}{2}$.
- $i \neq 1$ and $\mathcal{B}$ makes a correct random guess. This happens with probability $\left(1 - \frac{1}{K+1-j}\right)\frac{1}{2}$.

Summing it up, $\mathcal{B}$ makes a correct guess with probability $\frac{1}{K+1-j}\frac{1}{2}\left(\frac{1}{|R|} + x\right) + \frac{1}{K+1-j}\frac{1}{2}\left(1 - \frac{1}{|R|} - x\right)\frac{1}{2} + \frac{1}{K+1-j}\frac{1}{2}\left(1 - \frac{1}{|R|}\right)\frac{1}{2} + \left(1 - \frac{1}{K+1-j}\right)\frac{1}{2}$, *i.e.* with probability $\frac{1}{2} + \frac{1}{4(K+1-j)}x$, which yields an advantage of $\frac{1}{4(K+1-j)}x + \text{negl}(\lambda)$ to $\mathcal{B}$. Hence, $\mathcal{B}$ has a non-negligible advantage in the IND-CPA game, which is a contradiction with the fact that Paillier's cryptosystem is IND-CPA secure. ■

Theorem 6.3.1. *From the data obtained up to round j, a honest-but-curious* BAI *does not know the sum of the rewards of any arm. More precisely, for R the set of possible rewards,*

$$\left| P\left[A^{reward}(data_{\mathsf{BAI}^{\leq j}}) \in \{i, reward(i)\}_{i\in[\![K]\!]}\right] - \frac{1}{|R|}\right| < negl(\lambda)$$

where the data $data_{\mathsf{BAI}^{\leq j}}$ is the data obtained by BAI *during the first j rounds and $reward(i)$ is the reward of the i-th arm.*

Proof. We notice that $data_{\mathsf{BAI}^{\leq j}}$ is equal to $[data_{\mathsf{BAI}^1}, \ldots, data_{\mathsf{BAI}^j}] = [[se_1, i_{min_1}], \ldots, [se_j, i_{min_j}]]$. We know that each ciphertext from se_{j+1} results from the homomorphic addition of one ciphertext from se_j and one other unknown ciphertext[2]. Given Lemma 6.3.2, the set $[se_j, se_{j+1}]$ is indistinguishable from the set $[se_j, se'_{j+1}]$ where se'_{j+1} is a list of ciphertexts, unrelated to the ones in se_j. Hence, $data_{\mathsf{BAI}^{\leq j}}$ is indistinguishable from the list $[se_1, se'_2, \ldots, se'_j, i_{min_1}, \ldots, i_{min_j}]$, where se'_i is a list of ciphertexts unrelated to se'_j or se_1.

Assume that there exists a PPT adversary $\mathcal{A}$ able to break the above inequality, with an advantage of $x + \text{negl}(\lambda)$. The data available to $\mathcal{A}$ basically consists of j iterations, of various sizes, of the problem addressed in Lemma 6.3.3. Then, if $\mathcal{A}$ can solve our current adversarial game with non negligible advantage, $\mathcal{A}$ can immediately solve the problem in Lemma 6.3.3 with non negligible advantage (from one set of ciphertexts, $\mathcal{A}$ will generate other sets, and immediately places itself in the current problem). Because a non negligible advantage to the above problem breaks IND-CPA security, we conclude to a contradiction. ■

6.3.3 Security Proofs for Comp

Lemma 6.3.4. *Let $j \in [\![K-1]\!]$. From the data received at the round j, a honest-but-curious* Comp *cannot infer the ranking of any arm. More specifically,*

$$P\left[\mathcal{A}^{rank}(data_{\mathsf{Comp}^j}) \in \{i, ranking(i)\}_{i\in[\![K+1-j]\!]}\right] = \frac{1}{K+1-j}$$

[2]Namely, the ciphertext of the rewards of the arm i at round j.

Proof. We have $data_{\mathsf{Comp}} = se_{\sigma_j} = [\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, sum_{\sigma_j(\alpha_1)}), \dots, \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, sum_{\sigma_j(\alpha_{K+1-j})})]$, which can be decrypted by Comp to $s_{\sigma_j} = [sum_{\sigma_j(\alpha_1)}, \dots, sum_{\sigma_j(\alpha_{K+1-j})}]$ where the α_i are the arms still present at round j. From this list of scores, Comp can infer the ranking of the permuted arms, *i.e.*, compute the ranking any $A_{\sigma_j(i)}$ in polynomial time.

Assume there exists a PPT adversary $\mathcal{A}_{\text{rank}}$ capable of breaking the above equality. If $\mathcal{A}$ is able to predict the ranking of the arm i with advantage better than random, then $\mathcal{A}$ knows the ranking of A_i, namely $ranking(A_i)$. Knowing the ranking of all $A_{\sigma_j(i)}$, with probability better than random, $\mathcal{A}$ is then able to compute $\sigma_j^{-1}(i)$ with advantage better than random by identifying which $A_{\sigma_j(i)}$ matches A_i. Hence a contradiction with Lemma 6.3.1. ■

Theorem 6.3.2. *Let $j \in [\![K-1]\!]$. From the data received until the round j, A honest-but-curious* Comp *cannot infer the ranking of any arm. More specifically,*

$$P\left[\mathcal{A}^{rank}(data_{\mathsf{Comp}^{\leq j}}) \in \{i, ranking(i)\}_{i\in[\![K]\!]}\right] = \frac{1}{K+1-j}$$

Proof. The proof is based on the proof of Lemma 6.3.4, with additional arguments similar to the ones of the proof of Theorem 6.3.1: because of Lemma 6.3.2, we can assume that we have j independent sets of unrelated permuted data. If an adversary $\mathcal{A}$ can break the above equality with non-negligible advantage in PPT, then we can construct an adversary who breaks the equality of Lemma 6.3.4 with non-negligible advantage, in PPT, which breaks Lemma 6.3.1, so we get a contradiction. ■

Lemma 6.3.5. *Let $j \in [\![K-1]\!]$. From the data received at round j, a honest-but-curious* Comp *does not know the correspondence between sums of rewards and arms. More specifically,*

$$P\left[\mathcal{A}^{rwd}(data_{\mathsf{Comp}^j}) \in \{i, reward(i)\}_{i\in[\![K+1-j]\!]}\right] = \frac{1}{K+1-j}$$

Proof. Assume that Comp is able, from s_{σ_j}, to infer the sum of rewards of the arm A_k with probability better than $\frac{1}{K+1-j}$. Because Comp knows the sum of rewards of the permuted arms $A_{\sigma_j(\alpha_i)}$ for all $i \in [\![K+1-j]\!]$, then by matching these rewards with the sum of the rewards of A_i, Comp is able to compute $\sigma_j(i)$ with a probability better than random. Hence, Comp breaks Lemma 6.3.1. ■

Theorem 6.3.3. *Let $j \in [\![K-1]\!]$. From the data received until round j, a honest-but-curious* Comp *does not know the correspondence between sums of rewards and arms. More specifically,*

$$P\left[\mathcal{A}^{rwd}(data_{\mathsf{Comp}^{\leq j}}) \in \{i, reward(i)\}_{i\in[\![K]\!]}\right] = \frac{1}{K}$$

Proof. Similar to the proof of Theorem 6.3.2. ■

6.3.4 Security Proofs for the RP_j

Theorem 6.3.4. *A honest-but-curious RP_j does not know the ranking of the $K-j+1$ best ranking arms, for $j \in [\![K-1]\!]$. More specifically, $\forall j \in [\![K-1]\!], \forall i \in [\![K+1-j]\!]$, and $ranking_j(i)$ is the ranking of the i-th arm at round j,*

$$\left| P\left[\mathcal{A}^{rank}(data_{\mathsf{RP}_j}) \in \{i, ranking_j(i)\}_{i\in[\![K+1-j]\!]}\right] - \frac{1}{K+1-j}\right| < negl(\lambda)$$

Proof. We have $data_{\mathsf{RP}_j} = [se_{\sigma_j}, \mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \sigma_j), n_j]$, where $se_{\sigma_j} = \sigma_j([\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}} sum_{\alpha_1}), \ldots, \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, sum_{\alpha_{K-j}})])$, the permuted list of encrypted sums of rewards. RP_j can further 'un-permute' se_{σ_j} to $se = [\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, sum_{\alpha_1}), \ldots, \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, sum_{\alpha_{K-j}})]$, the list of encrypted sums of rewards. Note that n_j does not carry any information about the partial sum, as one can simulate any se with the same n_j, so does not carry significant information to our problem.

Assume that RP_j can guess the ranking of one element with advantage x+negl(λ): there exist a PPT oracle $\mathcal{A}$ taking as input se, and outputs $(i, \hat{v}(i))$, with $i \in [\![K+1-j]\!]$. Furthermore, we have $\widehat{v}(i) = ranking_j(i)$ with probability $\frac{1}{K+1-j} + x + \text{negl}(\lambda)$. Note that, on average, $i = 1$ with probability $\frac{1}{K+1-j}$.

Let us consider an IND-CPA game, in which the strategy of $\mathcal{B}$ is the same as the one in the proof of Lemma 6.3.3 (*i.e.*, simulate a blind bandit execution, and call $\mathcal{A}$ with the relevant data). Then, following the same reasoning we get that $\mathcal{B}$ wins the IND-CPA game with probability $\frac{1}{K+1-j}\frac{1}{2}\left(\frac{1}{K+1-j} + x\right) + \frac{1}{K+1-j}\frac{1}{2}\left(1 - \frac{1}{K+1-j} - x\right)\frac{1}{2} + \frac{1}{K+1-j}\frac{1}{2}\left(1 - \frac{1}{K+1-j}\right)\frac{1}{2} + \left(1 - \frac{1}{K+1-j}\right)\frac{1}{2} = \frac{1}{2} + \frac{1}{4(K+1-j)}x$, hence $\mathcal{B}$ has an advantage of $\frac{1}{4(K+1-j)}x + \text{negl}(\lambda)$, which is a contradiction with the IND-CPA property of Paillier's.

■

Theorem 6.3.5. *A honest-but-curious RP_j does not know the ranking of the $j-1$ lowest ranking arms. More specifically, $\forall j \in \{3, \ldots, K-1\}, \forall i \in [\![j-1]\!]$, and $ranking(i)$ the ranking of the i-th arm,*

$$\left| P\left[\mathcal{A}^{rank}(data_{\mathsf{RP}_j}) \in \{i, ranking(i)\}_{i\in[\![K+1-j]\!]}\right] - \frac{1}{j-1}\right| < negl(\lambda)$$

Proof. This is straightforward as RP_j does not receive any information about the sums of the j lowest ranking arms. Furthermore, we must impose $j \geq 3$ because it is clear that RP_1 and RP_2 know the ranking of the lowest ranking arm. ■

Theorem 6.3.6. *A honest-but-curious RP_j does not know the sums of rewards at step j. More precisely, for R the set of possible rewards, $\forall i \in [\![K+1-j]\!]$,*

$$\left| P\left[\mathcal{A}^{reward}(data_{\mathsf{RP}_j}) \in \{i, reward_j(i)\}_{i\in[\![K+1-j]\!]}\right] - \frac{1}{|R|}\right| < negl(\lambda)$$

Proof. Assume that a PPT adversary $\mathcal{A}$ breaks the above inequality: there exists a PPT oracle $\mathcal{A}(c_1, \ldots, c_K)$, that returns the tuple (i, m_i) where m_i is the cleartext

of c_i with advantage $x + \text{negl}(\lambda)$. Then we prove that the adversary breaks the IND-CPA property of Paillier's cryptosystem. Note that, on average, $i = 1$ with probability $\frac{1}{K}$, and that a decryption is correct with probability $\frac{1}{|R|} + x + \text{negl}(\lambda)$.

If we consider an IND-CPA game where the strategy of $\mathcal{B}$ is the same as in the proof of Lemma 6.3.3 (*i.e.*, simulate a blind bandit execution so they can call $\mathcal{A}$), we get that $\mathcal{B}$ has an advantage of $\frac{1}{4K}x$ in the IND-CPA game, which is a contradiction with Paillier being IND-CPA secure. ■

6.3.5 Security Proof for an External Observer

Theorem 6.3.7. *An external observer, having access to the set M of all the messages exchanged during the protocol, cannot infer anything about the sum of rewards of any arm. More specifically, any such observer is bound by the inequality mentioned in Theorem 6.3.1, with $data_{\mathsf{BAI}\leq j}$ being replaced by M.*

Proof. Assume that there exists an adversary $\mathcal{A}$ able to break the above inequality, given M, in PPT. We then prove that an adversary $\mathcal{B}$ is able to break IND-CPA security of Paillier's scheme in PPT.

Let us consider a classical IND-CPA challenge, in which $\mathcal{B}$ choses two rewards r_0, r_1 and sends them to the challenge. The challenger returns $\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, r_b)$, where b is a uniformly random bit. Then, $\mathcal{B}$ simulates a secure multi-armed bandit protocol, with 2 arms, so that at the end of round 1, one of the arms has for encrypted sum of rewards the value $\mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, r_b)$, the other being random. This is possible because in this simulation, $\mathcal{B}$ can set herself the rewards x_i of each arm, as well as the budget for round 1. Furthermore, knowing the cleartext of every encrypted value at any time, $\mathcal{B}$ can simulate the full protocol by herself (especially, she can simulate Comp execution). This simulation yields a set of messages M.

Now, calling $\mathcal{A}(M)$, $\mathcal{B}$ will retrieve in PPT, with some non-negligible advantage, some information about the sums of rewards of one of the arms. With probability $\frac{1}{2}$, this information will be about the arms of r_b, thus giving, in PPT, a non-negligible advantage in the IND-CPA game, as $\mathcal{B}$ is able to find the value of b with some advantage. This is a contradiction with the fact that Paillier is IND-CPA secure. ■

Theorem 6.3.8. *An external observer, having access to the set M of all the messages exchanged during the protocol, cannot infer anything about the ranking of any arm:*

$$\left| P\left[\mathcal{A}^{rwd}(M) \in \{i, ranking(i)\}_{i\in[\![K]\!]}\right] - \frac{1}{K} \right| < negl(\lambda)$$

Proof. It is obvious that such an observer can deduce the permuted list of rankings by listening to data exchanged at step 4. However, from the data of one round, it is impossible to know more: the data from one round is an encrypted permuted sum of rewards S, the lowest permuted index i, and the same sum, with the lowest element removed S' (steps 3,4,2). This is equivalent of having knowledge of S and i only. If an adversary $\mathcal{A}$ breaks the inequality with S and i, then we can break IND-CPA.

Let $\mathcal{B}$ be the IND-CPA adversary, picking $K+1$ messages such that $m_0 < m'_i < m_1$, and a permutation σ. Sending m_0 and m_1, they receive $c_b = \mathcal{E}(\mathsf{pk}, m_b)$, and

also compute $c'_i = \mathcal{E}(\mathsf{pk}, m'_i)$. Then, if $\mathcal{A}(\sigma([c_b, c'_2, \ldots, c'_k]), \sigma(0)) = 0$, $\mathcal{B}$ returns 0, else 1. If $\mathcal{A}$ has a non-negligible advantage x, we prove similarly to the other proofs that $\mathcal{B}$ has a advantage of $\frac{x}{2}$ in the IND-CPA game, which is a contradiction.

Now, because of Lemma 6.3.2, having access to all messages does not change anything. This is because each new round is indistinguishable from a simulation run by $\mathcal{B}$, so an advantage in the "all-rounds" game would yield an advantage in the "one-round" game. ■

6.4 Experiments

6.4.1 Setup

We report on a proof-of-concept experimental study of our proposed protocol. We implemented and compared:

- SR: the successive rejects algorithm, adapted from [ABM10]. We give the pseudocode of SR in Algorithm 1.
- SR-secured: our proposed protocol, which adds security guarantees to SR. We describe SR-secured in Section 6.2.3 and we outline its workflow in Figure 6.2.

We implemented the algorithms in Python 3. Our code is available on a public git repository[3]. For SR-secured, we used *phe*[4], an open-source Python 3 library for partially homomorphic encryption using the Paillier's cryptosystem.

We summarize the results in Figure 6.3 and we discuss them next. We carried out these experiments on a laptop with Intel Core i5 3.10GHz and 8GB of RAM. We used 2048 bits keys. The results are averaged over 100 runs.

6.4.2 Run time comparison SR vs SR-secured

In each half of Figure 6.3a, we have 12 points, corresponding to the pairwise combinations between 4 budget values N (100000, 200000, 300000, 400000) and 3 values for the number of arms K (5, 10, 15). We split the figure in two plots with different Y axis because the observed times are in the order of tens of milliseconds for SR and tens of seconds for SR-secured.

For SR, we observe that the time varies more on N and less on K, which makes sense because the operations depending on N (*i.e.*, picking random numbers in the rewards generation) are more expensive than the operations depending on K (*i.e.*, additions and multiplications).

On the other hand, for SR-secured, the slight run time increase depending on N is barely visible (hence the curves look rather constant) because of the three-orders-of-magnitude overhead that is a natural consequence of the high number of encryptions and decryptions performed by SR-secured.

[3] https://gitlab-sds.insa-cvl.fr/vciucanu/secure-bai-in-mab-public-code
[4] https://python-paillier.readthedocs.io/en/develop/

As explained in Section 6.2.3, each participant and encryption/decryption from SR-secured is useful for the protocol in order to guarantee the desired security properties. We stress that the time of SR-secured barely grows when increasing the budget N, which confirms the essential property that we outlined in the complexity discussion: the number of cryptographic primitives does not depend on N. Hence, we were easily able to run SR-secured for large budgets as we show in the figure.

We conclude from this experiment that SR-secured retains the scalability of SR while adding an overhead (depending on K and not on N) due to the security primitives. Obviously, both algorithms compute exactly the same result *i.e.*, the best arm. Moreover, before running this time comparison study, we carefully checked that all intermediate sums and arm rankings are identical for SR and SR-secured, despite the encryptions and decryptions that the latter algorithm performs.

6.4.3 Zoom on SR-secured

In Figure 6.3b we highlight how the total time taken by SR-secured is split among the participants. We obtained this figure for N=400000 and K=5, hence there are 4 phases, thus 4 participants RP_1, RP_2, RP_3, RP_4 in addition to Comp, BAI, the data owner DO, and the user U.

First, notice that the shares of U and DO of the total time are relatively small, which is a desired property. Indeed, we require the DO only to encrypt her knowledge of the arms before outsourcing such encrypted data to the cloud (step 0 in Figure 6.2). This could be actually done only once at the beginning and then all runs of the best-arm identification algorithm can be done using the same encrypted data, regardless of the user that pays for such a run.

Moreover, we require U to not do any computation effort other than decrypting the result of the best-arm identification algorithm that the cloud returns to her (step 5 in Figure 6.2).

Among the cloud participants, we observe that BAI takes the lion's share of the total running time. This is expected because the role of BAI is similar to a controller that interacts with all other cloud participants.

In what concerns Comp and the RP_j, their shares are quite similar. We observed the same behaviour regardless of the chosen N and K on which we zoom.

6.5 Related work

To the best of our knowledge, our work is the first that relies on public-key encryption in order to add privacy guarantees to best-arm identification algorithms for multi-armed bandits.

There is a recent line of research on multi-armed bandits using differential privacy techniques [Dwo06; DR14], which are based on adding an amount of noise to the data to ensure that the removal or addition of a single data item does not affect the outcome of any data analysis. These works have either focused on strategies to obtain: (i) privacy-preserving input guarantees *i.e.*, make the observed rewards un-

intelligible to an outside user [GUK18], or (ii) privacy-preserving output guarantees *i.e.*, protect the chosen actions and their associated rewards from revealing private information [MT15; TD16].

There are some fundamental differences between this line of work based on differential privacy and our work based on public-key encryption. First, the considered multi-armed bandit problems are different. Indeed, we focus on identifying the best arm, which is equivalent to minimizing the *simple regret*, that is the difference between the values associated to the arm that is actually the best and the best arm identified by the algorithm. On the other hand, the aforementioned works consider the *cumulative regret* minimization that roughly consists of minimizing the difference between the rewards observed after pulling N times the best arm and the rewards observed during the N pulls done by the algorithm.

A second difference is as follows. On the one hand, our secured algorithm based on *public-key encryption* is guaranteed to return exactly the same result as the (non-secured) SR algorithm [ABM10] on which we rely as a building block in our protocol. On the other hand, the result of a *differentially-private* algorithm contains by definition some noise, hence it is different from the result of the algorithm without privacy guarantees.

Third, by construction, the performance measure (the regret) of our secure algorithm remains the same as for the non-secured version, since both versions use the same arm-pulling strategies (that is, the performed encryptions/decryptions have no influence on the choice of arms to be pulled). The price we pay for making the algorithm secure comes only in the form of additional time needed for the encryptions and decryptions. In contrast, in the differential privacy approach, noise is introduced in the inputs/outputs in order to guarantee that the algorithms are differentially private and this has a direct impact on the arm-selection strategies. Therefore, the performance of the differential private versions of the algorithms suffers an increased regret with respect to their non-secured versions by an additive [TD16] or a multiplicative factor [GUK18; MT15].

6.6 Conclusions and Future Work

We studied the problem of best-arm identification in multi-armed bandits and we addressed the inherent privacy concerns that occur when outsourcing the data and computations to a public cloud. Our main contribution is a distributed protocol that computes the best arm while guaranteeing that (i) no cloud node can learn at the same time information about the rewards and about the ranking of the arms and (ii) by analyzing the messages communicated between the different cloud nodes, no information can be learned about the rewards or about the ranking. To do so, we relied on the partially homomorphic property of Paillier's cryptosystem. The overhead due to the security primitives of our protocol depends only on the number of arms K and not on the budget N. Our experiments confirmed this property.

Looking ahead to the future work, there are many directions for further investigation. For example, we plan to investigate whether we can leverage an addition-

homomorphic cryptosystem other than Paillier's, which may be more efficient in practice and could help us reduce the run time gap between the secured and the non-secured algorithms that we observed in our proof-of-concept experimental study. Additionally, we plan to add privacy guarantees to other multi-armed bandit settings *e.g.*, cumulative regret minimization [ACBF02] or best-arm identification in linear bandits [SLM14], where the rewards of the arms depend linearly on some unknown parameter.

Algorithm 7 Secure SR algorithms

function Setup_BAI(N) ▷ *Step 1* ▷ *j tracks the round number, sum contains the sum of rewards of each competing arm. Both are stored in* BAI *state.*
  for j from 1 to K-1 **do**
    $n_j \leftarrow \left\lceil \frac{1}{\overline{log}(K)} \frac{N-K}{K+1-j} \right\rceil - \sum_{l=0}^{j-1} n_l$
  for all $i \in [\![K]\!]$ **do**
    $sum[i] \leftarrow \mathcal{E}(\mathsf{pk}_{\mathsf{Comp}}, 0)$
  $j \leftarrow 1$

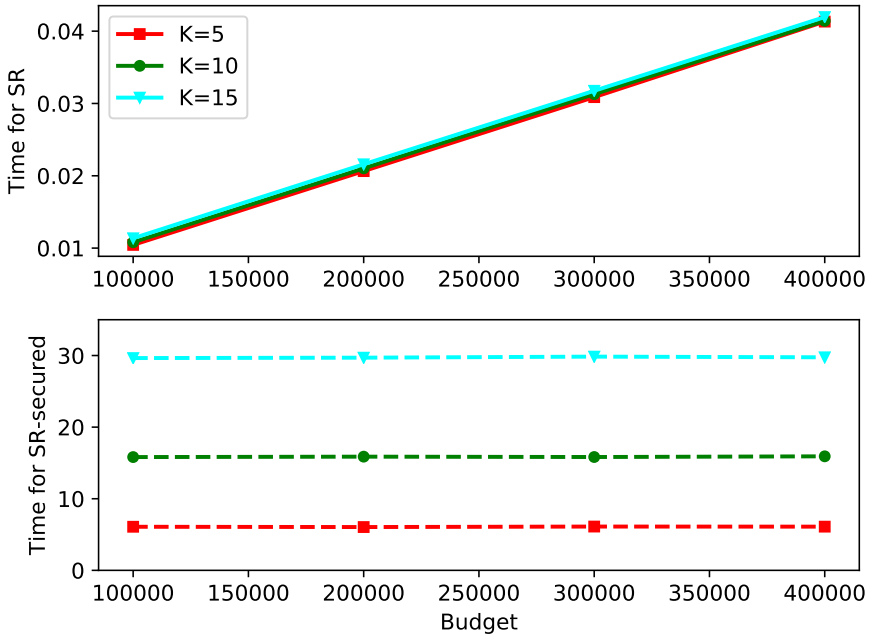
function Start_Round_BAI ▷ *Step* 2^*
  $\sigma_j \leftarrow$ random permutation of $[\![K-j+1]\!]$
  save σ_j in BAI state
  return $\sigma_j(sum), \mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \sigma_j), n_j$

function Round_RP_j($\sigma_j(sum), \mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \sigma_j), n_j$) ▷ *Step* 3^*
  Decrypt $\mathcal{E}(\mathsf{pk}_{\mathsf{RP}_j}, \sigma_j)$, retrieve σ_j and un-permute $\sigma_j(sum)$ to get sum
  for each arm in sum **do**
    Homomorphically add to sum[arm] the rewards from n_j pulls of the arm
  return $\sigma_j(sum)$

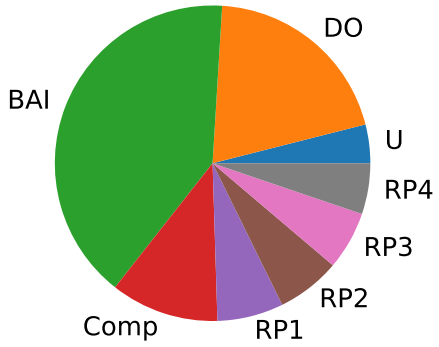
function Round_Comp($\sigma_j(sum)$) ▷ *Step* 4^*
  Decrypt each element of $\sigma_j(sum)$
  $x_{min} \leftarrow$ the index of a lowest element of the decrypted list, randomly chosen amongst all lowest elements
  return x_{min}

function End_Round_BAI($\sigma_j(sum), x_{min}$) ▷ *After Step* 4^**, before next round*
  $u_{min} \leftarrow \sigma_j^{-1}(x_{min})$
  Remove arm u_{min} from the list of participants, and from *sum* to reflect so
  $j++$

function Result ▷ *Step 5*
  Get the only remaining competing arm $\widehat{i^*}$ from *sum* in BAI state
  return $\mathcal{E}(\mathsf{pk}_{\mathsf{U}}, \widehat{i^*})$



(a) Run time (in seconds) for SR (full lines), and for SR-secured (dashed).



(b) Time share during SR-secured execution, for N=400000 and K=5.

Figure 6.3: Experimental results.

7

Secure Outsourcing of Multi-Armed Bandits

Contents

Abstract

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

Our implementation confirms the linear-time behavior and the practical feasibility of our protocol, on both synthetic and real-world data.

This work is joint with Radu Ciucanu, Pascal Lafourcade and Marta Soare. It has been accepted presented at the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2020), and published as [CLLP+20].

7.1 Introduction

7.1.1 Stochastic Multi-armed Bandit Game

The *stochastic multi-armed bandit* game is a sequential learning framework where a learning agent aims at maximizing its *cumulative reward* while successively interacting with an uncertain environment. At each time step, the agent chooses an action *(a bandit arm)* from among a fixed set of actions with unknown associated values. The environment responds with a stochastic feedback *(reward)* drawn from the distribution associated with the chosen action. The learning agent uses the received feedback to update its estimate of the values for the chosen action and to decide which action to choose next. The agent has to continuously face the so-called exploration-exploitation dilemma: it can decide to *explore* actions with more uncertain associated values, or to *exploit* the information already acquired by selecting the action with the seemingly largest associated value. Cumulative reward maximization has been already extensively studied for several multi-armed bandit settings (see [BC12] for a survey). In this chapter, we address the security concerns that occur when outsourcing the cumulative reward maximization data and computations to the cloud.

Our scenario is inspired by the *machine learning as a service* cloud computing model, for which security is known as a major concern [BMM+18]. As a motivating example, assume:

- A *data owner i.e.*, a company that wants to monetize some collected data that can be input for bandit algorithms. For instance, the data owner may be a large company (*e.g.*, Netflix) that collected a large quantity of customer preferences on TV series.

- A *user i.e.*, a company that wants to spend some budget to obtain the output of a bandit algorithm run on data owner's data. For instance, the user may be a smaller company that considers acquiring streaming rights for a package of TV series and wants to estimate, given the user's budget, what is the maximal reward that the package of TV series could generate.

The cumulative reward captures such an income for a package of series (outsourced arms) because it sums up the customer preferences (rewards) produced by each series in the package. This scenario allows the data owner to monetize some data that it collected and that the user wants to reuse. However, the data owner keeps ownership of the data, hence users are incited to spend a larger budget to obtain a more accurate estimate of the cumulative reward. We assume that the interaction between the data owner and the user is done using the *cloud* (as shown in Figure 7.1), where both data and computations are *outsourced*.

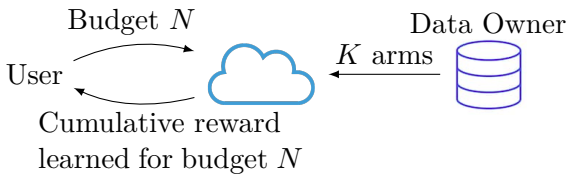


Figure 7.1: Outsourcing data and computations.

Thus, by relying on a standard cloud provider, the data owner does not have to build its own infrastructure. The data owner does the data outsourcing, and the user interacts directly with the cloud, by sending the budget and receiving the cumulative reward that can be learned for that budget. Such outsourced data may be sensitive (for instance, economical, personal or medical data) and we address the security concerns that occur when outsourcing data and computations. In other words, we want the outsourced learning algorithm to be run while protecting data against unauthorized access.

The problem that we address is how to allow the user to obtain precisely the same cumulative reward as with a standard algorithm, *Upper Confidence Bound (UCB)* [ACBF02; BC12], within a reasonable computation time and while preserving the data privacy. Indeed, the outsourced data can be communicated over an untrustworthy network and processed on some untrustworthy machines, where malicious cloud users may learn sensitive data that belongs only to the data owner. The privacy-preserving cumulative reward maximization is a hard problem. In [GUK18; MT15; TD16] the authors use *differential privacy* [Dwo06] to solve it. However,

	Differential privacy	Cryptography
Cumulative reward maximization a.k.a. cumulative regret minimization	Gajane et al. [GUK18] Mishra, Thakurta [MT15] Tossou, Dimitrakakis [TD16]	**This work**
Best arm identification a.k.a. simple regret minimization	Not yet studied to the best of our knowledge	Ciucanu et al. [CLLP+19]

Table 7.1: Summary of related work and positioning of our contribution.

in these approaches, the reward returned to the user is not the exact reward that can be learned for the user's budget (this happens because noise is injected in the input/output). It is unavoidable in any privacy-preserving approach that uses differential privacy.

We take a complementary look by relying on cryptography instead of differential privacy. To the best of our knowledge, our approach is original and its goal is to have an exact reward. The security for obtaining an exact reward has a price, since the computation time may increase because of cryptographic primitives that are time-consuming in practice. More precisely, we require that the data owner (which can be seen as an oracle knowing the reward functions associated with each arm) encrypts her data before outsourcing it to the cloud.

Then, the cumulative reward maximization algorithm is run directly in the encrypted domain, and by using the right primitives, the (encrypted) output should be exactly the same as for standard UCB, at the price of an increased computation time. From a theoretical point of view, the problem could be straightforwardly solved by using a fully homomorphic encryption scheme [Gen09], which allows to compute any function directly in the encrypted domain. However, it remains an open question how to make such a scheme work fast and be accurate in practice.

Indeed, the state-of-the-art fully homomorphic systems either yield only approximate results when they work with real numbers as those that we need in the UCB algorithm (this is the case for Microsoft SEAL [Sea] using CKKS scheme [CKK+17] for real numbers) or do not support real numbers at all (this is the case for IBM HElib [HS14]). Hence, it is not currently possible to program an algorithm such as UCB in a fully homomorphic system and obtain exactly the same result as in the standard, non-encrypted UCB.

Consequently, our challenge is to rely on simpler cryptographic schemes and design a distributed protocol with several cloud node participants such that each of them can only learn the specific data needed for performing its task and nothing else *e.g.*, if a participant does in clear computations on real numbers, these computations concern data of only one arm, and no other participant has access to this piece of data.

Our distributed algorithm returns exactly the same cumulative reward as UCB, while satisfying desirable security properties such as: only the user can see the

cumulative reward, which cannot be learned by any cloud node participant nor by an external observer. We precisely characterize our security model and security guarantees later on in the chapter.

To achieve our goals, we rely on *indistinguishable under chosen-plaintext attack* (IND-CPA) cryptographic schemes: symmetric encryption AES-CBC [NIS01; BDJ+97] and asymmetric partially homomorphic Paillier's scheme [Pai99]. We formally prove the security of our protocol and we precisely characterize the number of needed cryptographic operations.

7.1.2 Related work

We summarize related work in Table 7.1, where each line corresponds to a standard problem in stochastic multi-armed bandits. The most popular problem is *cumulative reward maximization* [BC12], where the agent uses its limited budget to maximize the sum of observed rewards.

To fulfil this goal, the agent decides which arm to pull next based on previously observed rewards: the agent can either choose to *exploit* the information acquired thus far and keep selecting the arm with the seemingly largest reward, or to *explore* by pulling one of the most uncertain arms. UCB is a standard algorithm for cumulative reward maximization [ACBF02; BC12].

From the privacy-preserving point of view, there is a recent line of research on enhancing algorithms such as UCB with differential privacy. These works have either focused on strategies to obtain: (i) privacy-preserving input guarantees *i.e.*, making the observed rewards unintelligible to an outside user [GUK18], or (ii) privacy-preserving output guarantees *i.e.*, protecting the chosen actions and their associated rewards from revealing private information [MT15; TD16]. Differential privacy algorithms have been also proposed for different bandit models than the standard stochastic one that we consider in this chapter *e.g.*, for contextual bandits [SS18].

There are some fundamental differences between this line of work based on differential privacy [GUK18; MT15; TD16] (top left box in Table 7.1) and our work based on cryptography (top right box in Table 7.1), which to the best of our knowledge is a novel approach in the community. On the positive side, the running time overhead of differentially-private algorithms is negligible, but on the other side the returned cumulative reward is different from the output of standard UCB. Indeed, to obtain differentially-private guarantees for a bandit algorithm, noise is added to the input or the output of the algorithm. Thus, the cumulative reward obtained using a differentially-private algorithm is different from that obtained by the algorithm without privacy guarantees. This is reflected in the *regret analysis* of the algorithms (where the *regret* is given by the difference in the cumulative reward obtained by a learning agent and the best cumulative reward possible obtained by always playing the best arm): the regret of differentially-private bandit algorithms have as overhead an additive [TD16] or a multiplicative factor [GUK18; MT15] with respect to the regret of their non-private version. In contrast, our cryptography-based algorithm is guaranteed to return exactly the same cumulative reward as the standard UCB.

The second line in Table 7.1 corresponds to a different bandit problem that is

best arm identification [ABM10], equivalent to minimizing the simple regret, that is the difference between the values associated with the arm that is actually the best and the best arm identified by the algorithm. To the best of our knowledge, this problem has not yet been studied from a differential-privacy point of view. From the cryptography point of view, there already exists a distributed algorithm [CLLP+19] that enhances the Successive rejects algorithm [ABM10] for best arm identification with security guarantees that are similar to the ones from this chapter. However, the standard algorithms that are secured (Successive rejects [ABM10] in [CLLP+19] and UCB [ACBF02] in this chapter) solve different problems and the corresponding distributed protocols cannot be reduced to one another.

7.1.3 Summary of contributions and Chapter organization

In Section 7.2, we introduce some basic notions: standard UCB algorithm and some cryptographic tools. Then, Section 7.3 is the core of our contribution:

- We propose UCB-DS, a secure and distributed protocol for cumulative reward maximization that guarantees the same cumulative reward as standard UCB.
- We show that UCB-DS satisfies desirable security properties that we precisely characterize, and prove.
- We analyze the theoretical complexity of UCB-DS, by quantifying the number of needed cryptographic primitives: $O(NK)$ AES-CBC encryptions/decryptions, K Paillier encryptions, and one Paillier decryption.
- We run a proof-of-concept empirical evaluation that confirms the theoretical complexity, and shows the scalability and practical feasibility of our protocols, on synthetic and real-world data.
- We propose the UCB-DS2 refinement, with stronger security guarantees at the price of K more AES-CBC keys and $O(NK)$ more AES-CBC encryptions/decryptions, and the same number of Paillier encryptions/decryptions as UCB-DS.

Finally, we conclude our chapter and outline directions for future work in Section 7.5.

The aforementioned evaluation suggests that it is practical to deploy our protocol on a real cloud and with real users, where parallelism between nodes could be leveraged to improve the system's throughput if multiple users are concurrently submitting budgets. We leave such an extended cloud system empirical evaluation as future work.

7.2 Preliminaries

We first recall the *UCB* algorithm [ACBF02]. Then, we present two cryptographic schemes that we use to build our protocols: *Paillier asymmetric encryption* scheme

and *AES-CBC symmetric encryption* scheme. We also recall the notion of *IND-CPA security*, held by both aforementioned schemes and useful for our security proofs.

7.2.1 Upper Confidence Bound (UCB)

We already defined the UCB algorithm in Section 2.2.3, page 15. We quickly give here a small reminder, that the UCB aims at maximising the cumulative reward sum, while the exact reward given by each machine is unknown and the budget limited. An algorithm has been proposed by [ACBF02] which identifies, at each turn, an upper bound of the estimation of reward of each arm. Given this parameter, named B_i for each arm i, one can select the next arm in order to maximize the expected total reward.

7.2.2 Paillier and AES cryptosystems

We have already introduced and defined Paillier and AES-CBC cryptosystems in Section 2.3, that the reader may confer to if needed, page 17.

We simply mention in this section by pointing out that all theoretical security properties of our protocols also hold if we choose any other IND-CPA symmetric scheme instead of AES-CBC, and any other additive homomorphic IND-CPA asymmetric scheme instead of Paillier. Our choice to rely on the aforementioned schemes is due to practical reasons. AES-CBC is very efficient in practice and implemented in standard libraries for modern programming languages. Paillier is also supported by a number of libraries that can be used in practice. In our protocol, we rely on Paillier whenever we need its additive homomorphic property, and on AES-CBC in all other situations when we need to symmetrically encrypt data.

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

We first define the security model and the desired security properties (Section 7.3.1). Then, we propose our secure protocol UCB-DS (Section 7.3.2), and we analyze its correctness, security, and complexity (Section 7.4). We introduce a refinement of UCB-DS in Section 7.4.4.

7.3.1 Security Model

As outlined in the Introduction and in Figure 7.1, we assume that the data (*i.e.*, the reward functions associated to K bandit arms) and the computations (*i.e.*, the cumulative reward maximization algorithm) are outsourced to an *honest-but-curious* cloud. This means that the cloud executes tasks dutifully, but tries to extract as much information as possible from the data that it sees. Our model follows the classical formulation in [Ode09] (Chapter 7.5, where *honest-but-curious* is denoted *semi-honest*), in particular:

1. each cloud node is trusted: it correctly does the required computations, it does not sniff the network and it does not collude with other nodes,

2. an external observer has access to all messages exchanged over the network.

The user indicates to the cloud her budget N and receives the cumulative reward R that the cloud computes using the K arms outsourced by the data owner and the user's budget N. The user does not have to do any computation, except for decrypting R when the user receives this information encrypted from the cloud. We expect the following *security properties*:

1. No cloud node can learn the cumulative reward.

2. The user cannot learn information about the rewards produced by each arm or which arm has been pulled at some round.

3. By analyzing the messages exchanged between different cloud nodes, an external observer cannot learn the cumulative reward, the sum of rewards produced by some arm, or which arm has been pulled at some round.

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

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

7.3.2 Overview of UCB-DS

In Figure 7.2, we present an overview of UCB-DS. There are $K+1$ cloud participants: K arm nodes R_i and a node AS (Arm Selector) that is the controller of the protocol.

We assume that the data owner and all cloud participants share the same symmetric AES-CBC key, which is used for the encryption function Enc in Figure 7.2. We also assume that the user U generates a key pair $(\mathsf{pk}, \mathsf{sk})$ using the Paillier cryptosystem, and this pk is used for the encryption function $\mathcal{E}_{\mathsf{U}}$ in Figure 7.2. In the sequel, by $[\![x]\!]$ we denote the set $\{1, \ldots, x\}$, and by $y||z$ we denote the concatenation of y and z.

In a nutshell, UCB-DS works as follows:

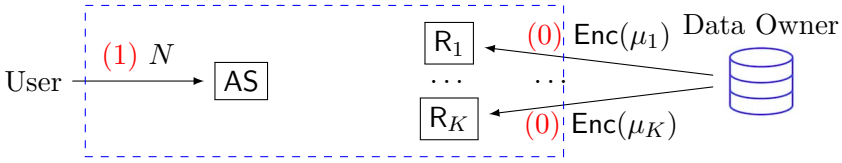
- Figure 7.2a (steps 0 and 1). For $i \in [\![K]\!]$, the data owner outsources to arm node R_i the reward function (encrypted with Enc) associated to arm i. The user sends to the cloud her budget N.
- Figure 7.2b (steps 2, 3, and 4). This is the core of the protocol, being done during $1+N-K$ iterations: once for the initialization phase of UCB and $N-K$ times for the exploration-exploitation phase of UCB cf. Algorithm 2. For each iteration, all arm nodes interact to decide which arm should be pulled next and communicate this information to AS. The arm nodes communicate in a random order, which changes at each iteration. All messages exchanged between nodes are encrypted with Enc. Although each arm node stores information about its rewards, it never reveals this information to other nodes.
- Figure 7.2c (steps 5 and 6). After spending the user's budget, each arm node sends to AS the sum of rewards that it produced, encrypted with $\mathcal{E}_{\mathsf{U}}$. Due to the additive homomorphic property of Paillier cryptosystem, AS is able to sum up the K partial rewards to compute the cumulative reward $\mathcal{E}_{\mathsf{U}}(R)$ directly in the encrypted domain. Only the user can decrypt this information.

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

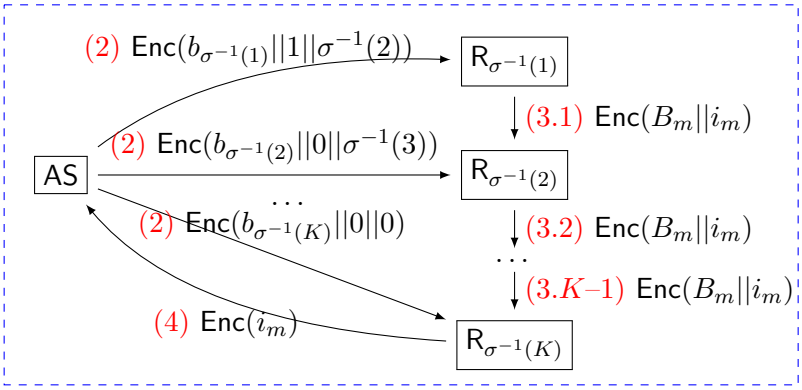
Algorithm 8 Pseudocode of AS during steps 2, 3, and 4 cf. Figure 7.2b

$i_m \leftarrow 0$
for $j \in [\![N-K+1]\!]$ **do**
  $\sigma \leftarrow$ random permutation of $[\![K]\!]$
  for $i \in [\![K]\!]$ **do** ▷ *b_i is a bit indicating whether arm i should be pulled or not*
    if $i_m = 0$ **or** $i_m = i$ **then**
      $b_i \leftarrow 1$
    else
      $b_i \leftarrow 0$
    if $\sigma(i) \neq K$ **then** ▷ *$next_i$ indicates the next arm node in the ring, or 0 if i is the last cf. σ*
      $next_i \leftarrow \sigma^{-1}(\sigma(i)+1)$
    else
      $next_i \leftarrow 0$
    Send $\mathsf{Enc}(b_i||first_i||next_i)$ to arm node R_i
  Receive ciphertext from arm node $\mathsf{R}_{\sigma^{-1}(K)}$ ▷ *ciphertext is* $\mathsf{Enc}(i_m)$
  $i_m \leftarrow \mathsf{Dec}$(ciphertext)

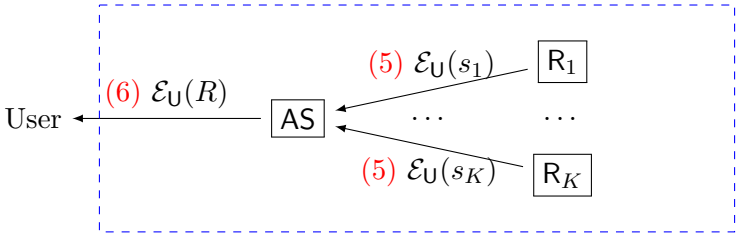
Step 0. We recall (cf. Algorithm 2) that the data owner knows $\mu_1, \ldots, \mu_K$ defining K Bernoulli distributions associated to the K arms. The data owner sends to each arm node R_i the encrypted value $\mathsf{Enc}(\mu_i)$, for $i \in [\![K]\!]$. Since the data owner and the



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



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



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

Figure 7.2: Overview of UCB-DS. The dashed rectangle is the cloud.

Algorithm 9 Pseudocode of R_i, for $i \in [\![K]\!]$, during steps 2, 3, and 4 cf. Figure 7.2b

Receive ciphertext1 from AS ▷ *ciphertext1 is* $\mathsf{Enc}(b_i||first_i||next_i)$
$(b_i, first_i, next_i) \leftarrow \mathsf{Dec}(\text{ciphertext1})$
$t \leftarrow t + 1$
if $b_i = 1$ **then** ▷ *Pull arm i and update its variables*
    $r \leftarrow pull(i)$
    $s_i \leftarrow s_i + r$
    $n_i \leftarrow n_i = 1$
$B_i \leftarrow \frac{s_i}{n_i} + \sqrt{\frac{2\ln(t)}{n_i}}$ ▷ *Always update B_i because t changes*
if $first_i = 0$ **then**
    Receive ciphertext2 from preceding arm node in ring ▷ *ciphertext2 is* $\mathsf{Enc}(B_m||i_m)$
    $(B_m, i_m) = \mathsf{Dec}(\text{ciphertext2})$
if $first_i = 1$ **or** $B_m < B_i$ **then**
    $i_m \leftarrow i$
    $B_m \leftarrow B_i$
if $next_i \neq 0$ **then**
    Send $\mathsf{Enc}(B_m||i_m)$ to R_{next_i}
else
    Send $\mathsf{Enc}(i_m)$ to AS

cloud nodes have shared the key used for Enc, then each arm node R_i can decrypt and obtain μ_i. Moreover, each node R_i initializes to 0 the following two variables that it later on updates during the protocol: s_i (*i.e.*, sum of rewards for arm i) and n_i (*i.e.*, number of times the arm i has been pulled). Additionally, each arm node R_i initializes a variable $t = K - 1$, which is later on updated and needed for the computation of B_i.

Step 1. This step is trivial: the user sends her budget N to AS, which receives N.

Steps 2, 3, and 4 are the most technical pieces of our protocol. We present the pseudocode of these steps in Algorithms 8 and 9 and we explain them next.

Step 2. This step corresponds to everything except the last two lines in Algorithm 8 and has $1 + N - K$ iterations. At each iteration, AS sends three pieces of information to the R_i nodes. The first one is a bit b_i indicating whether the arm i should be pulled or not. At the first iteration (that corresponds to the initialization phase of UCB cf. Algorithm 2), AS sends $b_i = 1$ to each arm, and at the next $N - K$ iterations (that correspond to the exploration-exploitation phase of UCB cf. Algorithm 2), AS sends $b_i = 1$ only to a chosen arm i_m and sends $b_i = 0$ to all other arms. Moreover, at each iteration, AS generates a permutation $\sigma : [\![K]\!] \rightarrow [\![K]\!]$ (*i.e.*, a function for which every element occurs exactly once as an image value), based on which AS computes two more components that it sends to R_i: $first_i$ that indicates whether the arm node is the first of the ring hence it should initialize B_m and i_m,

and $next_i$ that indicates to which node the updated B_m and i_m should be sent during Step 3. The arm node that receives 0 on the *next* component is the last one of the ring and sends i_m to AS, which thus knows which arm should be pulled next. All three components that AS sends to R_i are thus useful for the ring computation of i_m in Step 3. The permutation changes at each AS iteration because it is important to have a random order during the ring communication. Without a random order, it may happen that the last arm is much better than all others and it is almost always pulled, hence it has a very good estimate of the cumulative reward.

Step 3. This step corresponds to everything except the last two lines in Algorithm 9. Note that the variable t stores how many arm pulls have been done in total since the beginning of the protocol. As discussed for Step 0, each arm initialized $t = K - 1$, hence $t = K$ after the first iteration of AS, which allows to compute the first B_i values at the end of the initialization phase. Then, during the next $N - K$ iterations of AS, the variable t is incremented, which allows to compute B_i values during the exploration-exploitation phase. To decide which arm has the highest B_i and should be pulled at the next iteration, the arm nodes R_i do a distributed ring computation, where the first arm node according to permutation σ (*i.e.*, the only arm node that received $first_i$=1) initializes max value B_m and argmax i_m. At each ring iteration (Steps 3.1, ..., 3.K-1, cf. Figure 7.2b), the current arm node sends updated B_m and i_m to the next arm node cf. σ. Even though B_m and i_m do not change, it is important to re-encrypt $\mathsf{Enc}(B_m||i_m)$ before sending it to the next node to prevent an external observer from knowing when there is a change in the max and argmax (and hence learn information about which arms are pulled more often). Finally, once the ring computation reaches the last arm node relative to σ (*i.e.*, the only one that received $next_i = 0$), we go to Step 4.

Step 4. This step corresponds to the last two lines in Algorithm 9 (the last arm node in the ring sends $\mathsf{Enc}(i_m)$ to AS), followed by the last two lines in Algorithm 8 (AS receives and decrypts the index of the arm to be pulled at the next iteration).

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

Step 6. The node AS takes the K ciphertexts $\mathcal{E}_{\mathsf{U}}(s_i)$ received at Step 5, and computes $\mathcal{E}_{\mathsf{U}}(R)=\mathcal{E}_{\mathsf{U}}\left(\sum_{i=1}^{K} s_i\right)=\prod_{i=1}^{K}(\mathcal{E}_{\mathsf{U}}(s_i))$. This identity follows from the additive homomorphic property of Paillier cryptosystem, cf. Section 7.2. Then, AS sends $\mathcal{E}_{\mathsf{U}}(R)$ to the user, who is able to decrypt using her sk and hence the user obtains R.

7.4 Analysis of UCB-DS

Next, we analyze the correctness (Section 7.4.1), the security properties (Section 7.4.2), and the complexity (Section 7.4.3) of UCB-DS.

7.4.1 Correctness

We point out that UCB-DS outputs exactly the same cumulative reward as UCB. The computations done in Algorithms 8 and 9 to maximize the reward are the same as the one done in Algorithm 2. Indeed, if we take UCB-DS and remove all encryptions/decryptions (both symmetric and asymmetric), and all messages are communicated in clear between participants, then we obtain a protocol that we call UCB-D, which outputs exactly the same result as UCB-DS. This happens because of the consistency property of the chosen cryptographic schemes *i.e.*, if we encrypt a message M using Enc (or $\mathcal{E}_{\mathsf{U}}$, respectively) to obtain a ciphertext C, then if we decrypt C using Dec (or $\mathcal{D}_{\mathsf{U}}$, respectively), then we obtain exactly M.

Next, to reduce UCB-D to UCB, we simply remove all distributions of tasks among participants and rewrite UCB-D as a sequential algorithm to obtain exactly UCB. In particular, the random permutation σ (that is generated at each round to decide in which order to iterate over arms) reduces to the randomness in the argmax function used in standard UCB (cf. Algorithm 2) when, if several arms have maximal B_i-value, then the argmax should be randomly picked among those arms.

7.4.2 Security

In Table 7.2, we summarize what each participant in UCB-DS knows/does not know, while referencing the relevant theorems. The main properties of our protocol are:

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

These properties subsume the list of desirable security properties listed in Section 7.3.1. Before formally stating the theorems, we point out some assumptions. First, we recall (cf. Section 7.3.1) that the participants are honest-but-curious and do not collude. We discuss the impact of collusions at the end of this section. Second, during the ring computation (cf. Step 3 in Section 7.3.2), each arm learns an intermediate max value B_m, together with intermediate arm argmax i_m; we assume that the

Participant	Knows	Does not know
AS	• Arm pulled at each round	• Sum of rewards for some arm and cumulative reward (Theorem 7.4.1)
R_i	• Sum of rewards for arm i • Arm pulled at each round, with average probability $\frac{1}{2} + \frac{1}{2K}$ (Theorem 7.4.2)	• Sum of rewards of other arm $j \neq i$ and cumulative reward (Theorem 7.4.3)
U	• Cumulative reward	• Arm pulled at each round (Theorem 7.4.4) • Sum of rewards for some arm (Theorem 7.4.5)
External observer	• Nothing	• Arm pulled at each round (Theorem 7.4.6) • Sum of rewards for some arm and cumulative reward (Theorem 7.4.7)

Table 7.2: What each participant of UCB-DS knows and does not know.

knowledge on intermediate B_m and i_m by each arm does not leak significant information on the sum of rewards. We show in Section 7.4.4 our refinement UCB-DS2, which hides i_m during the ring computation to relax the second hypothesis.

Before discussing the security properties for each participant, we introduce some notations needed for the theorem statements:

- $n_{i,t}$ = the number of times arm i has been pulled until round t.
- $s_{i,t}$ = the sum of rewards obtained by arm i until round t.
- $data^t_{\mathsf{A}}$ = the data to which participant A has access until round t, where A can be a participant from Figure 7.2 or the external observer (*ext*). If t is omitted, this denotes the data to which A has access at the end of the protocol.
- $\mathcal{A}^{pb(.)}(d)$ = the answer of a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ that knows d and tries to solve the problem pb. Depending on the problem, pb can also take some *input*.
- By *negligible in* λ, we denote that our security theorems are always asymptotic *i.e.*, they describe the behavior when the security parameter λ of the cryptographic schemes becomes infinitely large. More precisely, a function $f : \mathbb{N} \to \mathbb{R}$ is negligible in λ if for any $c \in \mathbb{N}$, and for any λ large enough, $f(\lambda) < \lambda^{-c}$.

We next provide theorems that state each non-trivial property from Table 7.2. We first state an useful lemma, which intuitively says that guessing the cumulative reward with probability better than random is equivalent to guessing the sum of rewards of some arm with probability better than random.

Lemma 7.4.1. *Let $\mathcal{A}$ be a PPT adversary trying to find the cumulative reward R, and let $\mathcal{B}$ be a PPT adversary trying to find the sum of rewards of some arm. Let d be some data, $cr(.)$ be the problem of guessing the cumulative reward, and $sum(.)$ be the problem of guessing the sum of rewards of some arm. We have the following statement:*

$\mathcal{A}^{cr(.)}(d)$ has a non-negligible advantage $\Leftrightarrow$ $\mathcal{B}^{sum(.)}(d)$ has a non-negligible advantage.

Proof. $\Leftarrow$ Assume that $\mathcal{B}$ can guess the sum of rewards of some arm with probability better than random. Then, $\mathcal{A}$ can call $\mathcal{B}$, and hence get the sum of rewards of one arm with probability better than random. From this sum, $\mathcal{A}$ can guess a lower bound on the cumulative reward, hence eliminating some possibilities, and thus guessing the cumulative reward with probability better than random.

$\Rightarrow$ If $\mathcal{A}$ can guess the cumulative reward with probability better than random, then $\mathcal{B}$ can use this cumulative reward as an upper bound on the sum of rewards of some arm, thus having a probability better than random of guessing the sum of rewards of some arm. ■

Security of AS. By construction of UCB-DS, AS knows the arm pulled at each round. We state that AS cannot learn the sum of rewards produced by some arm.

Theorem 7.4.1. *For an arm $i \in [\![K]\!]$ and a round $t \in [\![N-K+1]\!]$, an honest-but-curious AS cannot learn $s_{i,t}$, given $data^t_{\mathsf{AS}}$, with a probability better than random. More precisely, for all PPT adversaries $\mathcal{A}$,*

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

is negligible in λ, where $\mathcal{A}^{sum(t)}(data^t_{\mathsf{AS}})$ returns $(i,\hat{s}_{i,t})$ in which $\hat{s}_{i,t}$ is $\mathcal{A}$'s guess on $s_{i,t}$ for the arm i (chosen by $\mathcal{A}$), and $p_S(n_{i,t}, s_{i,t})$ is the probability of obtaining a sum of rewards $s_{i,t}$ from $n_{i,t}$ pulls of arm i until round t.

Proof. Before Step 5 of UCB-DS, AS has access at each round to the indices of the pulled arms. Thus, AS knows $n_{i,t}$ *i.e.*, the number of times the arm i has been pulled until round t. The set of all possible sums of rewards for arm i until round t is $\{0,1,\ldots,n_{i,t}\}$. We denote by $p_S(n_{i,t}, s_{i,t})$ the probability of obtaining the sum of rewards $s_{i,t}$ from $n_{i,t}$ pulls of arm i until round t. Next, we show that the advantage of AS based of $data^t_{\mathsf{AS}}$ is $p_S(n_{i,t}, s_{i,t})$ plus an amount negligible in λ.

Since AS has no knowledge on μ_i, the property stated in the theorem is respected at each round before Step 5, *i.e.*, for all $t < N-K+1$.

We next prove the property for the last round *i.e.*, $t = N-K+1$. At the end of UCB-DS, at Step 5, AS receives the values $\mathcal{E}_{\mathsf{U}}(s_{1,t}),\ldots,\mathcal{E}_{\mathsf{U}}(s_{K,t})$. We prove that retrieving any information about any $s_{i,t}$ from these ciphertexts breaks the IND-CPA property of Paillier's cryptosystem [Pai99]. At this point of UCB-DS, $data^t_{\mathsf{AS}}$ consists of $\mathcal{E}_{\mathsf{U}}(s_{1,t}),\ldots,\mathcal{E}_{\mathsf{U}}(s_{K,t})$ and the list of arms that have been pulled at each round. Assume there exists a PPT adversary $\mathcal{A}$ able, from $data^t_{\mathsf{AS}}$ to find $s_{i,t}$ for

some i with non negligible advantage x:

$$\left|\Pr\left[(i,\hat{s}_{i,t}) \leftarrow \mathcal{A}^{sum(t)}(data^t_{\mathsf{AS}}); \hat{s}_{i,t} = s_{i,t}\right] - p_S(n_{i,t}, s_{i,t})\right| = x + \text{negl}(\lambda).$$

In the worst case, each $i \in [\![K]\!]$ has an equal probability of being chosen by $\mathcal{A}$. We also assume that if $data^t_{\mathsf{AS}}$ does not correspond to the data collected by AS during a run of UCB-DS (for instance, if one piece of $data^t_{\mathsf{AS}}$ has been replaced by another unrelated message), then $\mathcal{A}$ does not give any advantage. If such an adversary $\mathcal{A}$ exists, then we show how to construct an adversary $\mathcal{B}$ able to break the IND-CPA property of Paillier.

Let us build an IND-CPA game, in which $\mathcal{B}$ chooses two values m_0, m_1, and sends them to the challenger. The challenger randomly selects $b \in \{0,1\}$ and answers with $\mathcal{E}_{\mathsf{U}}(m_b)$. $\mathcal{B}$ wins the IND-CPA game if $\mathcal{B}$ guesses b with a non-negligible advantage.

To do so, $\mathcal{B}$ first creates a simulation of an UCB-DS execution *i.e.*, $\mathcal{B}$ creates nodes U', AS', R'_i, and DO', with Bernoulli distributions defined by μ'_i of its choice. Then, $\mathcal{B}$ runs an execution of UCB-DS on these nodes. Because $\mathcal{B}$ controls all the nodes, it knows the sums of rewards $s'_{1,t}, \ldots, s'_{K,t}$, as well as a list L of arms pulled at each round.

As input for the IND-CPA game, $\mathcal{B}$ chooses $m_1 = s'_{1,t}$ and another value m_0, different from all $s'_{i,t}$, sends both values to the challenger, and receives $\mathcal{E}_{\mathsf{U}}(m_b)$. Then, $\mathcal{B}$ computes $\mathcal{E}_{\mathsf{U}}(s'_{i,t})$ for each i, and calls $\mathcal{A}^{sum(t)}([\mathcal{E}_{\mathsf{U}}(m_b), \mathcal{E}_{\mathsf{U}}(s'_{2,t}), \ldots, \mathcal{E}_{\mathsf{U}}(s'_{K,t}), L])$.

The strategy of $\mathcal{B}$ is as follows: if $\mathcal{A}$ returns $(1, m_1)$, then $\mathcal{B}$ answers 1. Otherwise, $\mathcal{B}$ answers randomly. We next derive the probability of a correct answer by $\mathcal{B}$.

- If $i \neq 1$ (probability $1 - \frac{1}{K}$), then $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. Hence this branch offers a probability of success of $(1 - \frac{1}{K})\frac{1}{2}$.
- If $i = 1$ (probability $\frac{1}{K}$), let us consider the value of b.
  - If $b = 0$ (probability $\frac{1}{2}$), then we have two cases:
    - If the output of $\mathcal{A}$ is $(1, m_1)$ (probability $p_S(n_{1,t}, s_{1,t})$), then $\mathcal{B}$ answers 1 and it is wrong, hence the probability of success is 0.
    - Otherwise (probability $1 - p_S(n_{1,t}, s_{1,t})$), $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. The probability of success of this branch is $\frac{1}{K}\frac{1}{2}(1 - p_S(n_{1,t}, s_{1,t}))\frac{1}{2}$.
  - If $b = 1$ (probability $\frac{1}{2}$), then we have two cases:
    - If the output of $\mathcal{A}$ is $(1, m_1)$ (probability $p_S(n_{1,t}, s_{1,t}) + x + \text{negl}(\lambda)$), then $\mathcal{B}$ correctly answers 1. The probability of success of this branch is $\frac{1}{K}\frac{1}{2}(p_S(n_{1,t}, s_{1,t}) + x + \text{negl}(\lambda))$.
    - Otherwise (probability $1 - p_S(n_{1,t}, s_{1,t}) - x - \text{negl}(\lambda)$), $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. The probability of success of this branch is $\frac{1}{K}\frac{1}{2}(1 - p_S(n_{1,t}, s_{1,t}) - x - \text{negl}(\lambda))\frac{1}{2}$.

By aggregating the aforementioned cases, the probability α of success of $\mathcal{B}$ is:

$$\begin{aligned}\alpha =& \left(1-\frac{1}{K}\right)\frac{1}{2} + \frac{1}{K}\frac{1}{2}\left(1-p_S(n_{1,t},s_{1,t})\right)\frac{1}{2} + \frac{1}{K}\frac{1}{2}\left(p_S(n_{1,t},s_{1,t})+x+\text{negl}(\lambda)\right)\\ &+\frac{1}{K}\frac{1}{2}\left(1-p_S(n_{1,t},s_{1,t})-x-\text{negl}(\lambda)\right)\frac{1}{2}\\ =&\frac{1}{2}-\frac{1}{2K}+\frac{1}{4K}-\frac{p_S(n_{1,t},s_{1,t})}{4K}+\frac{p_S(n_{1,t},s_{1,t})}{2K}+\frac{x}{2K}+\frac{1}{4K}-\frac{p_S(n_{1,t},s_{1,t})}{4K}\\ &-\frac{x}{4K}+\text{negl}(\lambda)\\ =&\frac{1}{2}+\frac{x}{4K}+\text{negl}(\lambda)\end{aligned}$$

Hence, $\mathcal{B}$ has an advantage of $\frac{x}{4K}$ in the IND-CPA game, which is non negligible. This is a contradiction with the fact that Paillier is IND-CPA secure. Consequently, there does not exist any PPT adversary $\mathcal{A}$ that violates the property stated in the theorem. ■

As a corollary, by Lemma 7.4.1 and Theorem 7.4.1, we infer that AS cannot learn the cumulative reward with probability better than random.

Security of R_i. By construction of UCB-DS, each arm node R_i knows its sum of rewards. Moreover, due to the properties of the ring computation, R_i knows with average probability $\frac{1}{2}+\frac{1}{2K}$ the arm to be pulled at the next round (Theorem 7.4.2), but it cannot learn the sum of rewards of any other arm (Theorem 7.4.3).

Theorem 7.4.2. *At the end of round $t \in [\![N-K]\!]$ and before the start of round $t+1$, given data$^t_{\mathsf{R}_i}$, the average probability that an honest-but-curious R_i guesses the arm to be pulled at round $t+1$ is $\frac{1}{2}+\frac{1}{2K}$.*

Proof. After round t, an arm i can either guess randomly (with a success probability of $\frac{1}{K}$), or use the data to which it has access: the partial max B_m, the partial argmax index i_m, and the next arm in the ring communication. The knowledge of the next arm is useless, as it does not bring any information about any B value. Similarly, the knowledge of B_m does not leak more information than i_m. Hence, the only useful piece is i_m. Based on this only useful piece of data and on the assumption cf. Section 7.4.2 that any information derived from partial argmax data from the previous rounds is negligible, we infer that the best policy for the arm is to bet that arm i_m is the arm to be pulled at the next round. Let us consider an arm at position $\sigma^{-1}(i)$, where σ is the ring permutation used at the round t. Its guess is correct if and only if the next arm to be selected, say j, has position $\sigma^{-1}(j) \leq \sigma^{-1}(i)$. Hence, the arm at position $\sigma^{-1}(i)$ has a success probability of $\frac{\sigma^{-1}(i)}{K}$. On average, an arm has a success probability of

$$\frac{1}{K}\sum_{i=1}^{K}\frac{\sigma^{-1}(i)}{K} = \frac{1}{K^2}\frac{K(K+1)}{2} = \frac{K+1}{2K} = \frac{1}{2}+\frac{1}{2K}$$

which concludes the proof. ■

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

Table 7.3: Number of cryptographic operations used in UCB-DS.

Theorem 7.4.3. *For an arm $i \in [\![K]\!]$ and a round $t \in [\![N-K+1]\!]$, an honest-but-curious R_i cannot learn $s_{j,t}$ for some other arm $j \neq i$, given $data^t_{\mathsf{R}_i}$, with a probability better than random. More precisely, for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr\left[(j, \hat{s}_{j,t}) \leftarrow \mathcal{A}^{sum(t)}(data^t_{\mathsf{R}_i}); \hat{s}_{j,t} = s_{j,t}\right] - p_R(n_{i,t}, t, s_{j,t})\right|$$

is negligible in λ, where $\mathcal{A}^{sum(t)}(data^t_{\mathsf{R}_i})$ returns a tuple $(j, \hat{s}_{j,t})$ in which $j \neq i$ is chosen by $\mathcal{A}$ and $\hat{s}_{j,t}$ is $\mathcal{A}$'s guess of the sum of rewards for arm j, and $p_R(n_{i,t}, t, s_{j,t})$ is the probability of arm j to have sum of rewards $s_{j,t}$ at round t seen that arm i has been pulled $n_{i,t}$ times.

Proof. If an arm i has been pulled $n_{i,t}$ times until round t, then another arm j has been pulled at most $t - n_{i,t}$ times. Hence, a baseline probability of R_i to guess the sum of rewards of any other arm j is the $p_R(n_{i,t}, t, s_{j,t})$ defined in the theorem statement. The arm node R_i cannot possibly guess the sum of rewards for arm j with a better probability because it does not see any useful information that it can leverage. In particular, the only information that R_i receives about the rewards of any other arm is the partial max value B_m (derived from the sum of arm i_m using the number of pulls of i_m, to which R_i does not have access) received during Step 3. As mentioned in Section 7.4.2, we assume that the information that one arm can derive from one such random B value does not provide any advantage. ■

As a corollary, by Lemma 7.4.1 and Theorem 7.4.3, we infer that R_i cannot learn the cumulative reward with probability better than random.

Security of U. The user knows the cumulative reward that she can decrypt after Step 6. Moreover, the user cannot learn the arms selected at some round (Theorem 7.4.4) or the sum of rewards for some arm (Theorem 7.4.5).

Theorem 7.4.4. *For each round $t \in \{2, \ldots, N-K+1\}$, the user U cannot guess which arm is pulled at round t with probability better than random.*

Proof. The user does not receive any message until the end of UCB-DS (Step 6). By construction of UCB-DS, all arms are pulled at the first round, then from round 2 and until the end of UCB-DS *i.e.*, round $N-K+1$, there is a single arm pulled at each round. In particular, the user does not receive any information on which arm is pulled at some round, hence her best strategy is to answer randomly, with a probability of success of $\frac{1}{K}$. ■

Theorem 7.4.5. *For an arm $i \in [\![K]\!]$, the user U cannot guess the sum s_i of rewards for the arm i with probability better that random.*

Proof. Similarly to the previous proof, we observe that the user U does not receive any message until the end of UCB-DS (Step 6). In particular, U does not get any information about which arm is selected at some round. Because all arm probability distributions are equiprobable to U, it is also true that all partitions of the cumulative reward R are equiprobable to U, thus U has no advantage in guessing the partition of rewards. Hence, the the probability of U guessing a correct partition of the rewards is equal to $\frac{1}{p(R)}$, where $p(R)$ is the number of partitions of R. This observation also proves that U cannot guess the individual sum of rewards of some arm i. If it was the case, then U would know that some of the partitions are more likely e.g., if U can guess the sum of rewards s_i of the arm i, then all partitions not having s_i as the value for arm i would be discarded, which is a contradiction. ■

External observer. An external observer sees all messages exchanged between nodes, from which we show that she cannot learn which arm is pulled at some round (Theorem 7.4.6) or the sum of rewards for some arm (Theorem 7.4.7).

Theorem 7.4.6. *For each round $t \in \{2, \ldots, N-K+1\}$, an honest-but-curious external observer cannot learn which arm is pulled at round t, given $data^t_{ext}$, with probability better than random. More precisely, for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathcal{A}^{pa(t)}(data^t_{ext}) = i^t_m] - \frac{1}{K}\right| \text{ is negligible in } \lambda,$$

where $\mathcal{A}^{pa(t)}(data^t_{ext})$ returns the guess of $\mathcal{A}$ on which arm is pulled at round t, and i^t_m is the true arm pulled at round t.

Proof. By construction of UCB-DS, all arms are pulled at the first round, then from round 2 and until the end of UCB-DS *i.e.*, round $N-K+1$, there is a single arm pulled at each round. We next show that if there exists a PPT adversary with a non negligible advantage in guessing the arm pulled at some round $2 \leq t \leq N-K+1$, then this would break the IND-CPA property of AES-CBC.

An external observer (denoted *ext* in the sequel) sees all encrypted messages that are exchanged among UCB-DS participants. We denote by $data^t_{ext}$ this collection of data after round t. We assume, toward a contradiction, that there exists a PPT adversary $\mathcal{A}$ able from $data_{ext}$ to find the arm i^t_m pulled at some round t with a non negligible advantage x:

$$\left|\Pr[\mathcal{A}^{pa(t)}(data^t_{ext}) = i^t_m] - \frac{1}{K}\right| = x + \text{negl}(\lambda).$$

We also assume that if $data^t_{ext}$ does not correspond to an actual collection of encrypted messages that *ext* sees, then the advantage for such an input is negligible.

We next show that by using the adversary $\mathcal{A}$, we can construct an adversary $\mathcal{B}$ able to break the IND-CPA property of AES-CBC. To do so, $\mathcal{B}$ creates a simulation

of an UCB-DS execution, similarly to the proof of Theorem 7.4.1. Even though the messages of such a simulation are encrypted, $\mathcal{B}$ knows the keys hence the state of each arm. In particular, $\mathcal{B}$ knows in plain text the message sent by AS to the arm pulled at round t. This message is of the form $m_1 = (1||first_{i,t}||next_{i,t})$, with 1 being the Boolean value saying the arm has to be pulled.

As input for the IND-CPA game, $\mathcal{B}$ sends the aforementioned m_1 and another message $m_0 = (0||first_{i,t}||next_{i,t})$ that it generates based on m_1. Then, $\mathcal{B}$ receives back $\mathsf{Enc}(m_b)$, where b is a random bit selected uniformly by the challenger. Next, $\mathcal{B}$ calls $\mathcal{A}^{pa(t)}(data'_{ext})$, where $data'_{ext}$ is the collection of encrypted messages from the $\mathcal{B}$'s simulation, except that it replaces $\mathsf{Enc}(m_1)$ by $\mathsf{Enc}(m_b)$. The strategy of $\mathcal{B}$ is: if $\mathcal{A}$ returns the correct i_m^t, then $\mathcal{B}$ returns 1, otherwise answer randomly.

- If $b = 0$ (probability $\frac{1}{2}$), then $\mathcal{A}$ does not receive a correct simulation because no arm is pulled at round t. According to our assumption, $\mathcal{A}$ does not give any advantage.
  - If $\mathcal{A}$ returns the correct i_m^t (probability $\frac{1}{K}$), then $\mathcal{B}$ answers 1 and is wrong.
  - Otherwise (probability $1 - \frac{1}{K}$), then $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. This branch yields a probability of success of $\frac{1}{2}(1 - \frac{1}{K})\frac{1}{2}$.
- If $b = 1$ (probability $\frac{1}{2}$), then the advantage given by $\mathcal{A}$ can be leveraged by $\mathcal{B}$.
  - If $\mathcal{A}$ returns the correct i_m^t (probability $\frac{1}{K} + x + \text{negl}(\lambda)$), then $\mathcal{B}$ correctly answers 1. The probability of success of this branch is $\frac{1}{2}(\frac{1}{K} + x + \text{negl}(\lambda))$.
  - Otherwise (probability $1 - \frac{1}{K} - x - \text{negl}(\lambda)$), $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. This branch yields a probability of success of $\frac{1}{2}(1 - \frac{1}{K} - x - \text{negl}(\lambda))\frac{1}{2}$.

By aggregating the aforementioned cases, the probability α of success of $\mathcal{B}$ is:

$$\begin{aligned}\alpha &= \frac{1}{2}(1 - \frac{1}{K})\frac{1}{2} + \frac{1}{2}(\frac{1}{K} + x + \text{negl}(\lambda)) + \frac{1}{2}(1 - \frac{1}{K} - x - \text{negl}(\lambda))\frac{1}{2} \\ &= \frac{1}{4} - \frac{1}{4K} + \frac{1}{2K} + \frac{x}{2} + \frac{1}{4} - \frac{1}{4K} - \frac{x}{4} + \text{negl}(\lambda) \\ &= \frac{1}{2} + \frac{x}{4} + \text{negl}(\lambda)\end{aligned}$$

Hence, $\mathcal{B}$ has an advantage of $\frac{x}{4}$ in the IND-CPA game, which is non negligible. This contradicts the fact that AES-CBC is IND-CPA secure. Hence, we conclude that there does not exist any PPT adversary $\mathcal{A}$ that violates the property stated in the theorem. ■

Before proving the next theorem, we add a new hypothesis.

Hypothesis 2. *We assume that the messages exchanged by the nodes during the protocol are uniformly random among possible messages, or at least that, with overwhelming confidence, they are not related to the secret keys of the encryption schemes.*

This hypothesis is required, because specific messages such as key-dependant messages [BRS03] can alter the security of the protocol. For instance, it is known that IND-CPA does not imply circular security [CL01] (which states that, for a collection of encryption algorithms E_i or secret key sk_i, it is impossible to distinguish the encryptions $E_i(\mathsf{sk}_{i+1})$ from encryptions of 0. Especially, [CGH12] shows that there exists IND-CPA schemes in which circular encryption leads to a complete breakup (key recovery) of both schemes.

However, if we assume that messages are random, then key-related messages become a negligible event, and as such, all the related weaknesses vanish.

Theorem 7.4.7. *For an arm $i \in [\![K]\!]$ and a round $t \in [\![N-K+1]\!]$, an honest-but-curious external observer cannot learn $s_{i,t}$, given $data_{ext}^t$, with a probability better than random. More precisely, for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr\left[(i,\hat{s}_{i,t}) \leftarrow \mathcal{A}^{sum(t)}(data_{ext}^t); \hat{s}_{i,t} = s_{i,t}\right] - p_Q(t, s_{i,t})\right|$$

is negligible in λ, where $\mathcal{A}^{sum(t)}(data_{ext}^t)$ returns $(i,\hat{s}_{i,t})$ in which $\hat{s}_{i,t}$ is $\mathcal{A}$'s guess on $s_{i,t}$ for the arm i (chosen by $\mathcal{A}$), and $p_Q(t,s_{i,t})$ is the probability of obtaining a sum of rewards $s_{i,t}$ from at most t pulls of arm i until round t.

Proof. The external observer collects $data_{ext}^t$, which consists of several encrypted messages, some of them being encrypted with Enc (AES-CBC) and some other being encrypted with $\mathcal{E}_{\mathsf{U}}$ (Paillier). We prove that these messages do not provide an advantage bigger than the advantage of an adversary in a classical IND-CPA game on Enc or $\mathcal{E}_{\mathsf{U}}$. For simplicity, we assume that the $data_{ext}^t$ only contains two encrypted messages, $\mathsf{Enc}(m)$ and $\mathcal{E}_{\mathsf{U}}(n)$. The proof can naturally be adapted if $data_{ext}^t$ consists of more than two messages.

The goal of the adversary is to extract at least a bit of information from either m or n, with probability better than random. The entropy of this system is minimal when $m = n$. Hence, when $m = n$, the adversary has the highest probability of guessing at least a bit from either m or n (which are the same in this case). As a consequence, in the general case, the advantage of an adversary having to guess a bit about m or n, knowing $\mathsf{Enc}(m)$ or $\mathcal{E}_{\mathsf{U}}(n)$ is bounded above by the advantage of an adversary having to guess a bit about m, knowing $\mathsf{Enc}(m)$ and $\mathcal{E}_{\mathsf{U}}(m)$.

Let us prove that the advantage of a PPT adversary in this latter case (having to guess a bit about m from $\mathsf{Enc}(m)$ and $\mathcal{E}_{\mathsf{U}}(m)$) is negligible.

We assume, toward a contradiction, that there exists a PPT adversary $\mathcal{A}$ able to win the game where, given $\mathsf{Enc}(m)$ and $\mathcal{E}_{\mathsf{U}}(m)$, $\mathcal{A}$ recovers a bit of information about m with a non-negligible advantage x: given $\mathsf{Enc}(m)$ and $\mathcal{E}_{\mathsf{U}}(m)$, the probability that $\mathcal{A}$ outputs a correct guess about a bit of m is equal to $\frac{1}{2} + x + \mathrm{negl}(\lambda)$.

We use this adversary to create another adversary $\mathcal{B}$ able to break the IND-CPA property of the encryption schemes Enc (or $\mathcal{E}_{\mathsf{U}}$, respectively). As usually in

the IND-CPA game, $\mathcal{B}$ chooses two messages m_0 and m_1, and sends them to the challenger. Then, $\mathcal{B}$ receives the challenge $\mathsf{Enc}(m_b)$ (or $\mathcal{E}_{\mathsf{U}}(m_b)$, respectively), and calls $\mathcal{A}(\mathsf{Enc}(m_b), \mathcal{E}_{\mathsf{U}}(m_0))$ (or $\mathcal{A}(\mathsf{Enc}(m_0), \mathcal{E}_{\mathsf{U}}(m_b))$, respectively). If $\mathcal{A}$ returns a correct guess about m_0, then $\mathcal{B}$ returns 0. Otherwise, it returns 1.

- If $b = 0$ (happens with probability $\frac{1}{2}$), then $\mathcal{A}$ has a non negligible advantage in guessing a bit about m.
  - $\mathcal{A}$ outputs a correct guess about one bit of m_0 with probability $\frac{1}{2} + x + \text{negl}(\lambda)$. In this case, $\mathcal{B}$ is correct. This branch happens with probability $\frac{1}{2}(\frac{1}{2} + x + \text{negl}(\lambda))$.
  - If $\mathcal{A}$ does not answer correctly (happens with probability $\frac{1}{2} - x - \text{negl}(\lambda)$), then $\mathcal{B}$ is correct with probability $\frac{1}{2}$. This branch happens with probability $\frac{1}{2}(\frac{1}{2} - x - \text{negl}(\lambda))\frac{1}{2}$.
- If $b = 1$ (happens with probability $\frac{1}{2}$), then $\mathcal{A}$ has no advantage.
  - If $\mathcal{A}$ returns a correct guess about one bit of m_0 (happens with probability $\frac{1}{2}$), then $\mathcal{B}$ is wrong.
  - If not (happens with probability $\frac{1}{2}$), then $\mathcal{B}$ returns a random guess and is correct with probability $\frac{1}{2}$. This branch of events happen with probability $\frac{1}{2^3}$.

By aggregating these cases, the probability α of success of $\mathcal{B}$ is:

$$\begin{aligned}\alpha &= \frac{1}{2}(\frac{1}{2} + x + \text{negl}(\lambda)) + \frac{1}{2}(\frac{1}{2} - x - \text{negl}(\lambda))\frac{1}{2} + \frac{1}{8} \\ &= \frac{1}{4} + \frac{1}{2}x + \frac{1}{8} - \frac{1}{4}x + \frac{1}{8} + \text{negl}(\lambda) \\ &= \frac{1}{2} + \frac{1}{4}x + \text{negl}(\lambda)\end{aligned}$$

Hence, $\mathcal{B}$ has a non-negligible advantage of $\frac{1}{4}x$ in the IND-CPA game against Enc (or $\mathcal{E}_{\mathsf{U}}$, respectively), which is a contradiction with its IND-CPA property. Guessing a bit about the encrypted message is equivalent to guessing the reward with a probability better than random (*i.e.*, better than $p_Q(t, s_{i,t})$ cf. our theorem statement), which concludes our proof.

■

As a corollary, by Lemma 7.4.1 and Theorem 7.4.7, we infer that the external observer cannot learn the cumulative reward with probability better than random.

Impact of collusions. As pointed out earlier, an hypothesis behind our security theorems is that cloud nodes do not collude. By collusion we mean that cloud nodes put together all their data. If at least 2 of the R_i nodes collude, they could learn their respective algorithm inputs (*i.e.*, bandit arm values that only the data owner is supposed to know at the same time) and outputs (*i.e.*, cloud nodes could sum

up the partial sums of rewards known by each node), hence UCB-DS would not satisfy the desirable security properties. In addition to following a standard security model (cf. discussion in Section 7.3.1), we believe that the no-collusion hypothesis is necessary if we want a secure cumulative reward maximization algorithm that produces exactly the same output as standard UCB, which manipulates real numbers *i.e.*, B_i needs average, ln, and sqrt. Indeed, as already mentioned in the introduction, it is not currently possible in practice to use fully-homomorphic encryption on real numbers without result approximation. Hence, to minimize data leakage, our choice is to do computations on real numbers in clear, and to distribute reward functions and B_i-value computations among K cloud nodes (one per arm), each of them having access in clear only to data pertaining to its arm.

7.4.3 Complexity

We detail in Table 7.3 the number of cryptographic operations used in each step of UCB-DS. By summing up, we obtain $O(NK)$ AES-CBC encryptions/decryptions, K Paillier encryptions, and one Paillier decryption. Hence, we have a number of AES-CBC operations linear in N, whereas the number of Paillier operations does not depend on N. These are desirable complexity properties. In particular, the number of Paillier operations (which are quite slow to evaluate in practice) depends only on K that is typically much smaller than N in bandit scenarios.

Our implementation follows the aforementioned theoretical analysis and confirms the linear time behaviour and the scalability of UCB-DS. We include details on our proof-of-concept experimental study, following synthetic [GUK18; TD16] and real-world scenarios [KSS13; GRG+01; HK15].

7.4.4 Refinement: UCB-DS2

We propose next the UCB-DS2 refinement, which adds slightly stronger security guarantees to UCB-DS, for few more cryptographic operations. A property of UCB-DS, stated in Theorem 7.4.2, is that an arm node R_i knows with average probability of $\frac{1}{2} + \frac{1}{2K}$ what arm is pulled at the next round. This happens because during the ring computation, every arm sees in clear the partial argmax i_m. We propose the UCB-DS2 refinement of UCB-DS, which removes the aforementioned leakage and hence allows relaxing the second hypothesis from Section 7.4.2.

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

Algorithm 10 Modifications to the pseudocode of UCB-DS, cf. Algorithms 8 and 9, to obtain UCB-DS2.

For AS, take everything but the last 2 lines as in Algorithm 8, then take this pseudocode:

14: **Receive** ciphertext from arm node $\mathsf{R}_{\sigma^{-1}(K)}$ ▷ *ciphertext is now* $\mathsf{Enc}(\mathsf{Enc}_{i_m}(i_m))$

15: ciphertext2 $\leftarrow \mathsf{Dec}$(ciphertext)
16: **for** $i \in [\![K]\!]$ **do**
17: **if** Dec_i(ciphertext2) $= i$ **then** ▷ *ciphertext2 decrypts as a correct arm index*
18: $i_m \leftarrow i$
19: **Break**

For R_i (with $i \in [\![K]\!]$), take the first 8 lines as in Algorithm 9, then take the following pseudocode.

9: **if** $first_i = 0$ **then**
10: **Receive** ciphertext2 from preceding arm node in ring ▷ *ciphertext2 is now* $\mathsf{Enc}(B_m||\mathsf{Enc}_{i_m}(i_m))$
11: $(B_m, \mathsf{Enc}_{i_m}(i_m)) = \mathsf{Dec}$(ciphertext2)
12: **if** $next_i \neq 0$ **then**
13: **Send** $\mathsf{Enc}(B_m||\mathsf{Enc}_{i_m}(i_m))$ to R_{next_i}
14: **else**
15: **Send** $\mathsf{Enc}(\mathsf{Enc}_{i_m}(i_m))$ to AS

We show in Algorithm 10 the modifications to Step 3 and 4 of UCB-DS ()cf. Algorithms 8 and 9) that allow to obtain UCB-DS2. In the worst case, these modifications cost $(N-K+1)(K-1)$ encryptions at Step 3 and $(N-K+1)K$ decryptions at Step 4, which does not change the overall asymptotic behavior outlined in Section 7.4.3.

All theorems from Section 7.4.2 also hold for UCB-DS2, except Theorem 7.4.2 that is replaced by the next theorem, which formally states the stronger security guarantees of UCB-DS2.

Theorem 7.4.8. *In UCB-DS2, at the end of round $t \in [\![N-K]\!]$ and before the start of round $t+1$, given $data^t_{\mathsf{R}_i}$, an honest-but-curious arm node R_i cannot learn the arm to be pulled at round $t+1$ with probability better than random.*

Proof. At each round t, the arm node R_i receives $\mathsf{Enc}(B_m||\mathsf{Enc}_{i_m}(i_m))$ and decrypts into $B_m||\mathsf{Enc}_{i_m}(i_m)$. By hypothesis, B_m does not leak any information about the next arm to be pulled. The only way for R_i to guess the next arm with probability better than random is to use some information contained in $\mathsf{Enc}_{i_m}(i_m)$. However, since Enc_{i_m} is IND-CPA, it is impossible to learn any information on i_m with non negligible advantage. Hence, the strategy of R_i to guess the arm pulled at round $t+1$ is not better than random. ■

7.4.5 Experiments

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

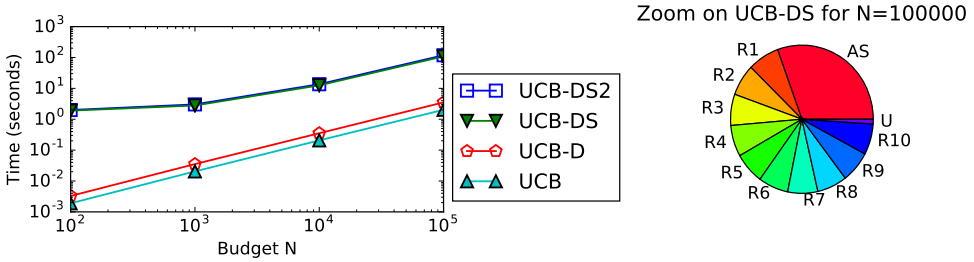
- UCB = Standard UCB algorithm [ACBF02; BC12], outlined in Algorithm 2.
- UCB-D = UCB with distribution of tasks among participants in the spirit of UCB-DS cf. Section 7.3.2, but with all messages exchanged in clear among participants (that is, UCB-D does not use any cryptographic primitive). The only overhead compared to UCB comes from the distribution of tasks.
- UCB-DS = Distributed Secure UCB cf. Section 7.3.2.
- UCB-DS2 = The refinement of UCB-DS cf. Section 7.4.4.

We implemented the algorithms in Python 3. For AES-CBC we used the *PyCryptodome* library[1] and keys of 256 bits. For additive homomorphic encryption using the Paillier's cryptosystem, we used the *phe* library[2] in the default configuration with keys of 2048 bits. We carried out our experiments on a laptop with CPU Intel Core i7 of 2.80GHz and 16GB of RAM, running Ubuntu. Each result that we report is averaged over 100 runs. In each run, we executed all four algorithms using the same random seeds, needed for drawing the arm rewards and for generating the permutation used to iterate in a random order over the arms when choosing the argmax arm to be pulled at the next round.

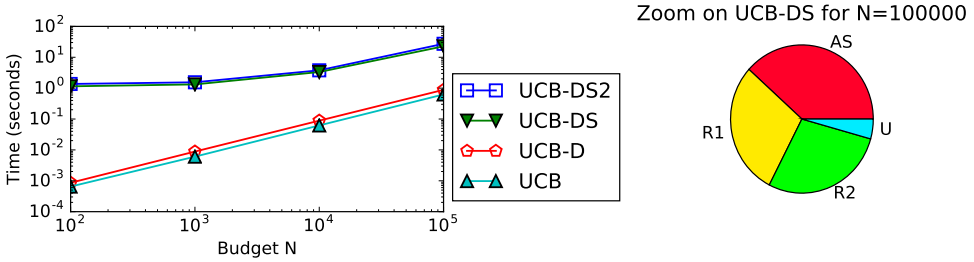
For reproducibility reasons, we make available on a public GitHub repository[3] our source code, together with the data that we used, the generated results from which we obtained our plots, and scripts that allow to install the needed libraries and reproduce our plots.

Before discussing our experimental results, we would like to stress that, as expected, in each experiment, all four algorithms output exactly the same cumulative reward. The property that our secure algorithms return exactly the same cumulative reward as standard UCB is in contrast with differentially-private multi-armed bandit algorithms [GUK18; MT15; TD16], where the returned cumulative rewards are different from that of standard UCB. Consequently, a shallow empirical comparison between these works and ours boils down to comparing apples and oranges: (i) on the one hand, the running time of differentially-private bandit algorithms is roughly the same as for standard UCB and is never reported in their experiments, whereas (ii) on the other hand, for our algorithms the cumulative reward is always the same as for standard UCB and consequently there is no point for us in doing any plot on the cumulative reward. Nevertheless, we carefully analyzed all experimental settings

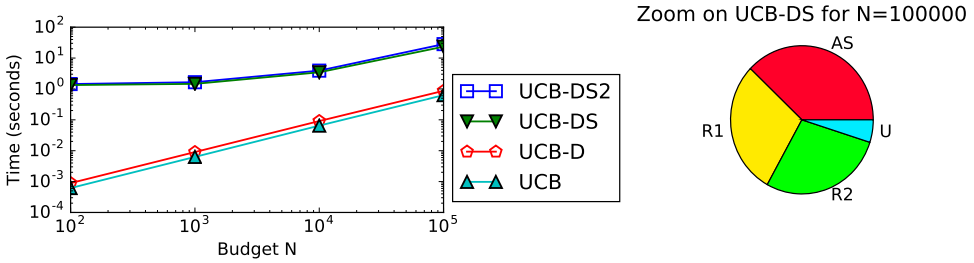
[1] https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html
[2] https://python-paillier.readthedocs.io/en/develop/
[3] https://github.com/radu1/secure-ucb



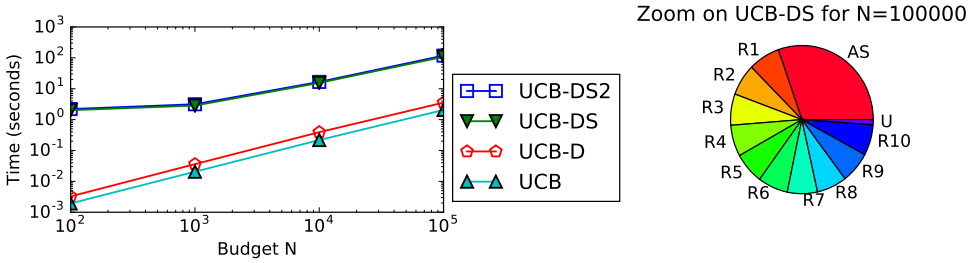
(a) Scenario 1 [GUK18]: $K = 10$, $\mu_1 = 0.9$, $\mu_2 = \ldots = \mu_{10} = 0.8$.



(b) Scenario 2 [GUK18; TD16]: $K = 2$, $\mu_1 = 0.9$, $\mu_2 = 0.6$.

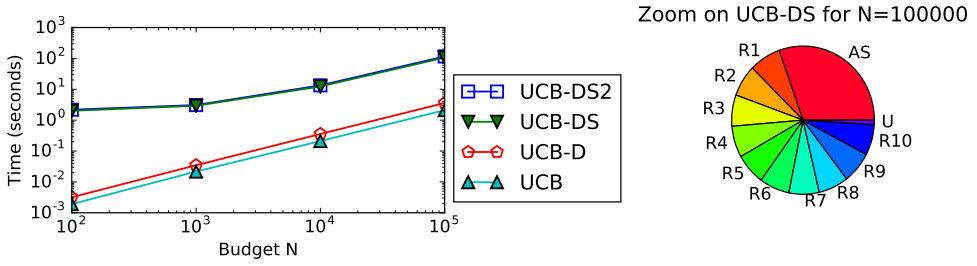


(c) Scenario 3 [GUK18] $K = 2$ $\mu_1 = 0.9$, $\mu_2 = 0.8$

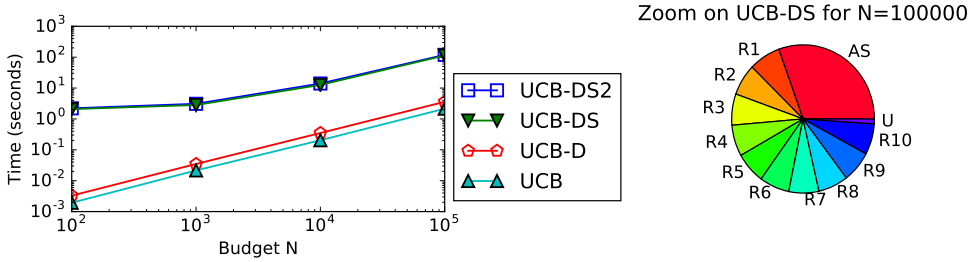


(d) Scenario 4 [GUK18] $K = 10$: $\mu_1 = 0.9$, $\mu_2 = \mu_3 = \mu_4 = 0.8$, $\mu_5 = \mu_6 = \mu_7 = 0.7$, $\mu_8 = \mu_9 = \mu_{10} = 0.6$.

Figure 7.3: Scalability with respect to N. In the zoom, we do not show DO (its share is close to 0).



(e) Scenario 5 [GUK18] $K = 10$: $\mu_1 = 0.9$, $\mu_2 = \ldots = \mu_{10} = 0.6$.



(f) Scenario 6 [TD16] $K = 10$: $\mu_1 = 0.55$, $\mu_2 = 0.2$, $\mu_3 = \ldots = \mu_{10} = 0.1$.

Figure 7.3: (cont.) Scalability with respect to N. In the zoom, we do not show DO (its share is close to 0).

(N, K, μ) used in the related work, that we adapt for our scalability experiments, as we detail next.

Scalability with respect to N. In this experiment, we fix K and μ, and we vary N. To fix K and μ, we rely on six scenarios from the related work, as detailed in Figure 7.3, where we show the comparison of the four algorithms for these scenarios. We vary N from 10^2 to 10^5 that is also the maximum budget considered in [GUK18; TD16]. The running times for UCB and UCB-D are very close, and up to two orders of magnitude smaller than the times of UCB-DS and UCB-DS2, which are also very close. All algorithms have a similar linear time behavior. The overhead between the secure and non-secure algorithms comes naturally from the cryptographic primitives. Moreover, the two lines corresponding to the secure algorithms are not parallel with the other two lines because, cf. Section 7.4.3, the overhead due to Paillier encryptions depends only on K (that is fixed in the figure) and not on N (that varies in the figure), hence the Paillier overhead is more visible for small N. The running times of UCB-DS/UCB-DS2 for the largest considered budget $N = 10^5$ is of ∼100 seconds, which remains practical. In Figure 7.3, we also zoom on the time taken by each participant of UCB-DS for $N = 10^5$. We observe that AS takes the lion's share, which is expected because at each round AS sends encrypted messages to all R_i participants, whereas each R_i sends an encrypted message only to one other participant. As expected, all R_i take roughly the same time. Moreover, the shares taken by the data owner and the user are the smallest among all participants,

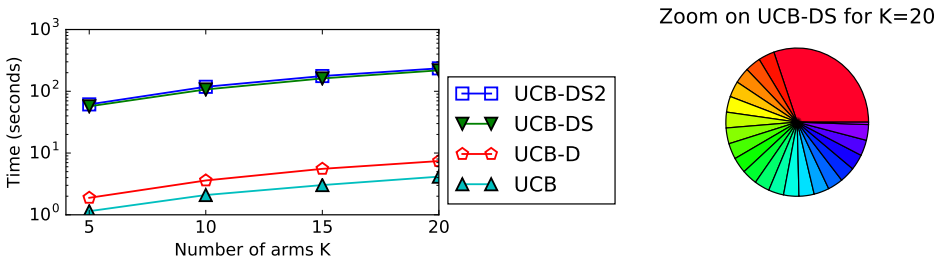


Figure 7.4: Scalability with respect to K for fixed $N = 10^5$. In the zoom (labels not shown because they would be colliding): the AS takes the lion's share, $\mathsf{R}_{1 \leq i \leq 20}$ take the 20 equal shares, U is barely visible, and DO is not shown since its share is close to 0.

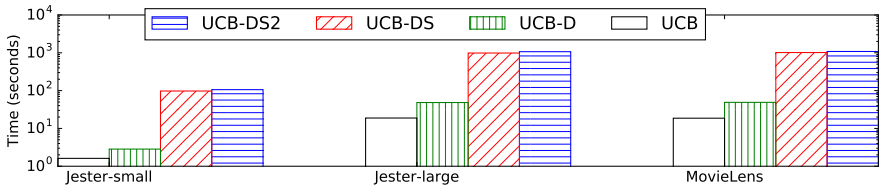


Figure 7.5: Running times on three real world data scenarios from [KSS13].

which is a desirable property because we require them to do as few computations as possible, whereas the bulk of the computation is outsourced to the cloud.

Scalability with respect to K. In this experiment, we fix $N = 10^5$, and we vary $K \in [5, 10, 15, 20]$ and implicitly μ with $\mu_1 = 0.9$ and all other $\mu_{2 \leq i \leq K} = 0.8$. We present results in Figure 7.4, where we observe, as in the previous experiment, a similar linear time behavior and a similar zoom on the time taken by each participant.

Real-world data. In addition to the two aforementioned scalability experiments on synthetic data, we think it is important to also stress test our algorithms on some real-world data. For this experiment, we use the same data and experimental setup as [KSS13]. More precisely, we use data from Jester[4][GRG+01], a collection of ratings ranging from -10 (very not funny) to 10 (very funny), given by 25K users on 100 jokes. Exactly as [KSS13], we pre-process this dataset by assigning the lowest score to the unrated jokes, and then we extract two bandit scenarios:

- Jester-small: $K = 10$, corresponding to the 10 most rated jokes, where $\mu_i =$ (# of ratings $\geq$ threshold 3.5 for joke i) / (# of users).
- Jester-large: $K = 100$, corresponding to all 100 jokes, where μ_i is computed similarly as for Jester-small, except that the threshold here is set to 7.

[4] http://eigentaste.berkeley.edu/dataset/

Moreover, we use data from MovieLens[5] [HK15], more precisely the "MovieLens 100K Dataset" that contains ratings ranging from 1 (bad) to 5 (very good) given by 1K users on a set movies, from which, exactly as [KSS13], we look only at the first 100 movies and derive the following bandit scenario:

- MovieLens: $K = 100$, corresponding to the first 100 movies, where μ_i = (# of ratings $\geq$ threshold 4 for movie i) / (# of users).

We ran each of the three aforementioned scenarios with budget $N = 10^5$, which is the largest budget considered in [KSS13]. We show results in Figure 7.5, which essentially confirms the behavior observed in the synthetic experiments *i.e.*, there are roughly two orders of magnitude between non-secure and secure algorithms. In the largest considered scenarios (Jester-large and MovieLens, both with $K = 100$), where standard UCB takes around twenty seconds, both UCB-DS and UCB-DS2 take around one thousand seconds, that we believe acceptable as waiting time for the user before getting the cumulative reward result for which she pays.

7.5 Conclusions and Future Work

We tackled the problem of cumulative reward maximization in multi-armed bandits in a setting where data and computations are outsourced to some honest-but-curious cloud. We proposed UCB-DS, a distributed and secure protocol based on UCB, which yields exactly the same cumulative reward as UCB while enjoying desirable security properties that we precisely characterize. In particular, no cloud node or external observer can learn the cumulative reward, which can be seen only by the user who pays a budget. We rely on distribution of tasks and on cryptographic schemes to achieve the security properties of UCB-DS, and we characterize the overhead of cryptography from both theoretical and empirical points of view. Our experiments show the scalability and practical feasibility of UCB-DS, and of its refinement UCB-DS2.

As future work, we plan to extend our scenario such that multiple users can concurrently submit budgets to the cloud and receive in return corresponding learned cumulative rewards. Such a scenario would imply heavier computations. We believe that the different tasks of our secure distributed algorithms could be pipelined among cloud nodes in order to increase the system's throughput. We also plan to develop protocols for adding security guarantees to other types of bandit algorithms (such as contextual and linear bandits), and more in general, for other learning settings.

[5] https://grouplens.org/datasets/movielens/

Conclusion

The aim of this thesis was to fill in some gaps between theoretical cryptography and everyday software, by proposing new applications of existing cryptographic primitives with guarantees on the security properties, so that the user can confidently know what they are doing and how secure the framework they are using actually is. In this context, we have taken a look on a special topic of computing, namely duplicate detection, and started by improving the current state, and comparing our solution to an optimal bound we derived. After that, we compared our solutions to the state of the art, and showed a net improvement of our solution. After that, we drove into a general study of the security of each filter, in the context where an adversary only has access to the filter's output (and not its internal state). These results help understanding how filters may behave in adversarial context, and if they are efficient enough for the use case (for instance, some applications require very low false positive when others require very low false negative, when both low error rates usually cannot be obtained). As such, we have opened a new field of study on the optimization of such filters in adversarial settings.

Then, we took a look into blockchain and tried to formalize one of its properties, namely interoperability. We acknowledge that blockchain interoperability was already reached before the writing of our paper, thus our work only focuses on the formal definition of what interoperability is, and how it can be studied on blockchain. For this, we created a formal framework for studying blockchain, which might be reused for other purposes.

Finally, we proposed new exchange protocols for decentralizing machine learning algorithms. In this chapter, the idea was to decentralise machine learning algorithms, which might have an elevated computing cost for the user, who might prefer to delegate this computation to a cloud. However, naive implementations would leak data to the cloud nodes, hence our use of cryptographic primitives for securing data exchanges. Furthermore, we quantified the amount of information each node learns, thus formally guaranteeing the security of our schemes.

Images Credits

The images under each part title belong to their respective authors.

- Part II: CC BY-NC 2.5 Randall Munroe, cropped from https://xkcd.com/2030/
- Part III: ©Alicia Filipak, reproduced with the author permission

List of Figures

List of Tables

List of Algorithms

Bibliography

[ABM10] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. "Best Arm Identification in Multi-Armed Bandits". In: *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*. Ed. by Adam Tauman Kalai and Mehryar Mohri. Omnipress, 2010, pp. 41–53. URL: https://hal-enpc.archives-ouvertes.fr/hal-00654404 (*cited on pages* 13–15, 96, 109, 111, 120, 153).

[ACBF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-Time Analysis of the Multiarmed Bandit Problem". In: *Mach. Learn.* 47.2–3 (May 2002), pp. 235–256. ISSN: 0885-6125. DOI: 10.1023/A:1013689704352 (*cited on pages* 15, 16, 112, 117, 119–121, 139, 153).

[Agr95] Rajeev Agrawal. "Sample Mean Based Index Policies with $O(\log(n))$ Regret for the Multi-Armed Bandit Problem". In: *Advances in Applied Probability* 27.4 (1995), pp. 1054–1078. DOI: 10.2307/1427934 (*cited on page* 15).

[AKG+18] Antonio Fernández Anta, Kishori Konwar, Chryssis Georgiou, and Nicolas Nicolaou. "Formalizing and Implementing Distributed Ledger Objects". In: *SIGACT News* 49.2 (June 2018), pp. 58–76. ISSN: 0163-5700. DOI: 10.1145/3232679.3232691 (*cited on page* 86).

[BC12] Sébastien Bubeck and Nicolò Cesa-Bianchi. "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems". In: *Found. Trends Mach. Learn.* 5.1 (2012), pp. 1–122. DOI: 10.1561/2200000024 (*cited on pages* 116, 117, 119, 139).

[BCD+14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. *Enabling blockchain innovations with pegged sidechains*. 2014. URL: http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains (*cited on page* 85).

[BDJ+97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. FOCS '97. USA: IEEE Computer Society, 1997, pp. 394–

403. ISBN: 0818681977. DOI: 10.1109/SFCS.1997.646128 (*cited on pages* 19, 119).

[BDM04] B. Babcock, M. Datar, and R. Motwani. "Load shedding for aggregation queries over data streams". In: *Proceedings. 20th International Conference on Data Engineering*. IEEE Computer Society, Mar. 2004, pp. 350–361. DOI: 10.1109/ICDE.2004.1320010 (*cited on page* 27).

[Blo70] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692 (*cited on pages* 27, 60).

[BMM+18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. "Fast Homomorphic Evaluation of Deep Discretized Neural Networks". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 483–512. DOI: 10.1007/978-3-319-96878-0_17 (*cited on page* 116).

[BRJ+14] Markus Borg, Per Runeson, Jens Johansson, and Mika V. Mäntylä. "A Replicated Study on Duplicate Detection: Using Apache Lucene to Search Among Android Defects". In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '14. Torino, Italy: ACM, 2014, 8:1–8:4. ISBN: 978-1-4503-2774-9. DOI: 10.1145/2652524.2652556 (*cited on page* 27).

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. "Encryption-Scheme Security in the Presence of Key-Dependent Messages". In: *Selected Areas in Cryptography*. Ed. by Kaisa Nyberg and Howard Heys. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 62–75. ISBN: 978-3-540-36492-4. DOI: 10.1007/3-540-36492-7_6 (*cited on page* 135).

[But14] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform*. 2014. URL: https://github.com/ethereum/wiki/wiki/White-Paper (*cited on pages* 84, 86).

[But16] Vitalik Buterin. *Chain Interoperability*. 2016. URL: https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf (*cited on pages* 84, 85).

[CCL+19] Yao Cheng, Cheng-Kang Chu, Hsiao-Ying Lin, Marius Lombard-Platet, and David Naccache. "Keyed Non-parametric Hypothesis Tests". In: *Network and System Security - 13th International Conference, NSS 2019, Sapporo, Japan, December 15-18, 2019, Proceedings*. Ed. by Joseph K. Liu and Xinyi Huang. Vol. 11928. Lecture Notes in

Computer Science. Springer, 2019, pp. 632–645. DOI: 10.1007/978-3-030-36938-5_39 (*cited on page* 5).

[CDPLB+17] Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. "SCARFF: a Scalable Framework for Streaming Credit Card Fraud Detection with Spark". In: *CoRR* abs/1709.08920 (2017). arXiv: 1709.08920 (*cited on pages* 12, 60).

[CFP+00] Corinna Cortes, Kathleen Fisher, Daryl Pregibon, and Anne Rogers. "Hancock: A Language for Extracting Signatures from Data Streams". In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Boston, Massachusetts, USA: Association for Computing Machinery, 2000, pp. 9–17. ISBN: 1581132336. DOI: 10.1145/347090.347094 (*cited on page* 60).

[CGH12] David Cash, Matthew Green, and Susan Hohenberger. "New Definitions and Separations for Circular Security". In: *Public Key Cryptography – PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 540–557. ISBN: 978-3-642-30057-8. DOI: 10.1007/978-3-642-30057-8_32 (*cited on page* 135).

[CHV+03] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. "Password Interception in a SSL/TLS Channel". In: *Advances in Cryptology - CRYPTO 2003*. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 583–599. ISBN: 978-3-540-45146-4. DOI: 10.1007/978-3-540-45146-4_34 (*cited on page* 26).

[CKK+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 409–437. ISBN: 978-3-319-70694-8. DOI: 10.1007/978-3-319-70694-8_15 (*cited on page* 118).

[CL01] Jan Camenisch and Anna Lysyanskaya. "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation". In: *Advances in Cryptology — EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 93–118. ISBN: 978-3-540-44987-4. DOI: 10.1007/3-540-44987-6_7 (*cited on page* 135).

[CLJ+17] Hanhua Chen, Lianglyi Liao, Hai Jin, and Jie Wu. "The dynamic cuckoo filter". In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. Oct. 2017, pp. 1–10. DOI: 10.1109/ICNP.2017.8117563 (*cited on page* 28).

[CLK+14] Shouyuan Chen, Tian Lin, Irwin King, Michael R. Lyu, and Wei Chen. "Combinatorial Pure Exploration of Multi-Armed Bandits". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 379–387. DOI: 10.5555/2968826.2968869 (*cited on page* 96).

[CLLP+19] Radu Ciucanu, Pascal Lafourcade, Marius Lombard-Platet, and Marta Soare. "Secure Best Arm Identification in Multi-armed Bandits". In: *Information Security Practice and Experience*. Ed. by Swee-Huay Heng and Javier Lopez. Cham: Springer International Publishing, 2019, pp. 152–171. ISBN: 978-3-030-34339-2. DOI: 10.1007/978-3-030-34339-2_9 (*cited on pages* 4, 96, 118, 120).

[CLLP+20] Radu Ciucanu, Pascal Lafourcade, Marius Lombard-Platet, and Marta Soare. "Secure Outsourcing of Multi-armed Bandits". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020, pp. 202–209. DOI: 10.1109/TrustCom50675.2020.00038 (*cited on pages* 5, 116).

[CM03] Saar Cohen and Yossi Matias. "Spectral Bloom Filters". In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD '03. San Diego, California: ACM, 2003, pp. 241–252. ISBN: 1-58113-634-X. DOI: 10.1145/872757.872787 (*cited on page* 27).

[CM05] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58 –75. ISSN: 0196-6774. DOI: 10.1016/j.jalgor.2003.12.001 (*cited on page* 27).

[CM07] Pierre-Arnaud Coquelin and Rémi Munos. "Bandit Algorithms for Tree Search". In: *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*. UAI'07. Vancouver, BC, Canada: AUAI Press, 2007, pp. 67–74. ISBN: 0974903930. DOI: 10.5555/3020488.3020497 (*cited on page* 96).

[Cry19] CryptoID. *Crypto-currency blockchain explorers*. 2019. URL: https://chainz.cryptoid.info/ (*cited on page* 84).

[DD06] Fan Deng and Rafiei Davood. "Approximately detecting duplicates for streaming data using stable bloom filters". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*. Ed. by Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis. ACM, 2006, pp. 25–36. DOI: 10.1145/1142473.1142477 (*cited on pages* 27, 31, 34, 49, 62, 66).

[DLOM02] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. "Frequency Estimation of Internet Packet Streams with Limited Space". In: *Proceedings of the 10th Annual European Symposium on Algorithms*. ESA '02. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 348–360. ISBN: 3540441808. DOI: 10.1007/3-540-45749-6_33 (*cited on page* 60).

[DNB13] Sourav Dutta, Ankur Narang, and Suman K. Bera. "Streaming Quotient Filter: A Near Optimal Approximate Duplicate Detection Approach for Data Streams". In: *Proc. VLDB Endow.* 6.8 (June 2013), pp. 589–600. ISSN: 2150-8097. DOI: 10.14778/2536354.2536359 (*cited on pages* 27–29, 31, 33, 40, 45, 49, 62, 65).

[DP08] Martin Dietzfelbinger and Rasmus Pagh. "Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract)". In: *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*. Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz. Vol. 5125. Lecture Notes in Computer Science. Springer, 2008, pp. 385–396. DOI: 10.1007/978-3-540-70575-8_32 (*cited on pages* 10, 27).

[DR14] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3–4 (Aug. 2014), pp. 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042 (*cited on page* 110).

[DS81] Dorothy E Denning and Giovanni Maria Sacco. "Timestamps in key distribution protocols". In: *Communications of the ACM* 24.8 (1981), pp. 533–536. DOI: 10.1145/358722.358740 (*cited on page* 26).

[Dwo06] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Vol. 4052. Lecture Notes in Computer Science. Springer, 2006, pp. 1–12. DOI: 10.1007/11787006_1 (*cited on pages* 110, 117).

[EDMM06] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. "Action Elimination and Stopping Conditions for the Multi-Armed Bandit and Reinforcement Learning Problems". In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 1079–1105. ISSN: 1532-4435. DOI: 10.5555/1248547.1248586 (*cited on page* 96).

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. "Duplicate Record Detection: A Survey". In: *IEEE Trans. Knowl. Data Eng.* 19.1 (2007), pp. 1–16. DOI: 10.1109/TKDE.2007.250581 (*cited on page* 27).

[ELL+20] Rose Esmander, Pascal Lafourcade, Marius Lombard-Platet, and Claudia Negri Ribalta. "A silver bullet?: a comparison of accountants and developers mental models in the raise of blockchain". In: *ARES 2020: The 15th International Conference on Availability, Reliability and Security, Virtual Event, Ireland, August 25-28, 2020*. Ed. by Melanie Volkamer and Christian Wressnegger. ACM, 2020, 83:1–83:10. DOI: 10.1145/3407023.3409193 (*cited on page* 7).

[FAK+14] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. "Cuckoo Filter: Practically Better Than Bloom". In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Sydney, Australia: ACM, 2014, pp. 75–88. ISBN: 978-1-4503-3279-8. DOI: 10.1145/2674005.2674994 (*cited on pages* 27, 28, 34, 35, 46, 49, 62, 66).

[FCA+00] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol". In: *IEEE/ACM Trans. Netw.* 8.3 (June 2000), pp. 281–293. ISSN: 1063-6692. DOI: 10.1109/90.851975 (*cited on pages* 12, 27).

[FFH+15] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Wen Xia, Yucheng Zhang, and Yujuan Tan. "Design Tradeoffs for Data Deduplication Performance in Backup Workloads". In: *Proceedings of the 13th USENIX Conference on File and Storage Technologies*. FAST'15. Santa Clara, CA: USENIX Association, 2015, pp. 331–344. ISBN: 978-1-931971-201. URL: http://dl.acm.org/citation.cfm?id=2750482.2750507 (*cited on page* 27).

[Gam84] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2 (*cited on page* 61).

[Gen09] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 169–178. ISBN: 9781605585062. DOI: 10.1145/1536414.1536440 (*cited on page* 118).

[GGL12] Victor Gabillon, Mohammad Ghavamzadeh, and Alessandro Lazaric. "Best Arm Identification: A Unified Approach to Fixed Budget and Fixed Confidence". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 3212–3220. DOI: 10.5555/2999325.2999493 (*cited on page* 96).

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *Advances in Cryptology - EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 281–310. ISBN: 978-3-662-46803-6. DOI: 10.1007/978-3-662-46803-6_10 (*cited on page* 86).

[GLPN19] Rémi Géraud, Marius Lombard-Platet, and David Naccache. "Quotient Hash Tables: Efficiently Detecting Duplicates in Streaming Data". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: Association for Computing Machinery, 2019, pp. 582–589. ISBN: 9781450359337. DOI: 10.1145/3297280.3297335 (*cited on pages* 3, 26, 31, 60–62, 65–67, 73).

[GO03] Lukasz Golab and M. Tamer Özsu. "Issues in Data Stream Management". In: *SIGMOD Rec.* 32.2 (June 2003), pp. 5–14. ISSN: 0163-5808. DOI: 10.1145/776985.776986 (*cited on page* 60).

[GRG+01] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. "Eigentaste: A Constant Time Collaborative Filtering Algorithm". In: *Inf. Retr.* 4.2 (July 2001), pp. 133–151. ISSN: 1386-4564. DOI: 10.1023/A:1011419012209 (*cited on pages* 137, 142).

[GSLPN20] Rémi Géraud-Stewart, Marius Lombard-Platet, and David Naccache. "Approaching Optimal Duplicate Detection in a Sliding Window". In: *Computing and Combinatorics*. Ed. by Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee. Cham: Springer International Publishing, 2020, pp. 64–84. ISBN: 978-3-030-58150-3. DOI: 10.1007/978-3-030-58150-3_6 (*cited on pages* 3, 60).

[GUK18] Pratik Gajane, Tanguy Urvoy, and Emilie Kaufmann. "Corrupt Bandits for Preserving Local Privacy". In: *Algorithmic Learning Theory, ALT 2018, 7-9 April 2018, Lanzarote, Canary Islands, Spain*. Ed. by Firdaus Janoos, Mehryar Mohri, and Karthik Sridharan. Vol. 83. Proceedings of Machine Learning Research. PMLR, 2018, pp. 387–412 (*cited on pages* 111, 117–119, 137, 139–141).

[GWC+10] Deke Guo, Jie Wu, Honghui Chen, Ye Yuan, and Xueshan Luo. "The Dynamic Bloom Filters". In: *IEEE Transactions on Knowledge and Data Engineering* 22.1 (Jan. 2010), pp. 120–133. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.57 (*cited on page* 28).

[HK15] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015). ISSN: 2160-6455. DOI: 10.1145/2827872 (*cited on pages* 137, 143).

[HS14] Shai Halevi and Victor Shoup. "Algorithms in HElib". In: *Advances in Cryptology – CRYPTO 2014*. Ed. by Juan A. Garay and Rosario Gennaro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 554–571. ISBN: 978-3-662-44371-2. DOI: 10.1007/978-3-662-44371-2_31 (*cited on page* 118).

[IBM17] IBM Hyperledger Consortium. *Hyperledger Sawtooth*. 2017. URL: https://sawtooth.hyperledger.org/docs/core/releases/latest/index.html (*cited on page* 86).

[JRB19] Sandra Johnson, Peter Robinson, and John Brainard. "Sidechains and interoperability". In: *arXiv e-prints*, arXiv:1903.04077 (Mar. 2019), arXiv:1903.04077. arXiv: 1903.04077 [cs.CR] (*cited on pages* 84, 89).

[JST10] Hossein Jowhari, Mert Saglam, and Gábor Tardos. "Tight Bounds for Lp Samplers, Finding Duplicates in Streams, and Related Problems". In: *CoRR* abs/1012.4889 (2010), pp. 49–58. arXiv: 1012.4889 (*cited on page* 27).

[KCG16] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. "On the Complexity of Best-Arm Identification in Multi-Armed Bandit Models". In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 1–42. ISSN: 1532-4435. DOI: 10.5555/2946645.2946646 (*cited on page* 96).

[KLC19] Mirko Koscina, Marius Lombard-Platet, and Pierre Cluchet. "PlasticCoin: an ERC20 Implementation on Hyperledger Fabric for Circular Economy and Plastic Reuse". In: *2019 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019, Thessaloniki, Greece, October 14-17, 2019 - Companion Volume*. Ed. by Payam M. Barnaghi, Georg Gottlob, Dimitrios Katsaros, Yannis Manolopoulos, Rahul Pandey, Theodoros Tzouramanis, and Athena Vakali. ACM, 2019, pp. 223–230. DOI: 10.1145/3358695.3361107 (*cited on page* 6).

[KLNR21] Mirko Koscina, Marius Lombard-Platet, and Claudia Negri Ribalta. "A blockchain-based marketplace platform for circular economy". In: *SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021*. Ed. by Chih-Cheng Hung, Jiman Hong, Alessio Bechini, and Eunjee Song. ACM, 2021, pp. 1746–1749. DOI: 10.1145/3412841.3442136. URL: https://doi.org/10.1145/3412841.3442136 (*cited on page* 7).

[KNP+17] Michael Kapralov, Jelani Nelson, Zhengyu Pachocki Jakuband Wang, David P. Woodruff, and Mobin Yahyazadeh. "Optimal Lower Bounds for Universal Relation, and for Samplers and Finding Duplicates in Streams". In: *FOCS*. IEEE Computer Society, 2017, pp. 475–486. DOI: 10.1109/FOCS.2017.50 (*cited on page* 27).

[KS06] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293. ISBN: 978-3-540-46056-5. DOI: 10.1007/11871842_29 (*cited on page* 96).

[KS08] Marios Kleanthous and Yiannakis Sazeides. "CATCH: A Mechanism for Dynamically Detecting Cache-Content-Duplication and its Application to Instruction Caches". In: *2008 Design, Automation and Test in Europe*. Mar. 2008, pp. 1426–1431. DOI: 10.1109/DATE.2008.4484874 (*cited on page* 61).

[KSS13] Pushmeet Kohli, Mahyar Salek, and Greg Stoddard. "A Fast Bandit Algorithm for Recommendations to Users with Heterogeneous Tastes". In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI'13. Bellevue, Washington: AAAI Press, 2013, pp. 1135–1141. DOI: 10.5555/2891460.2891618 (*cited on pages* 137, 142, 143).

[LCL+10] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 661–670. ISBN: 9781605587998. DOI: 10.1145/1772690.1772758 (*cited on page* 96).

[LL19] Pascal Lafourcade and Marius Lombard-Platet. "Get-your-ID: Decentralized Proof of Identity". In: *Foundations and Practice of Security - 12th International Symposium, FPS 2019, Toulouse, France, November 5-7, 2019, Revised Selected Papers*. Ed. by Abdelmalek Benzekri, Michel Barbeau, Guang Gong, Romain Laborde, and Joaquín García-Alfaro. Vol. 12056. Lecture Notes in Computer Science. Springer, 2019, pp. 327–336. DOI: 10.1007/978-3-030-45371-8_20 (*cited on page* 6).

[LLP20] Pascal Lafourcade and Marius Lombard-Platet. "About blockchain interoperability". In: *Information Processing Letters* 161 (2020), p. 105976. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2020.105976 (*cited on pages* 4, 84).

[MAEA05] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. "Duplicate Detection in Click Streams". In: *Proceedings of the 14th International Conference on World Wide Web*. WWW '05. Chiba, Japan: ACM, 2005, pp. 12–21. ISBN: 1-59593-046-9. DOI: 10.1145/1060745.1060753 (*cited on pages* 12, 27, 61, 62).

[ME97] Alvaro E. Monge and Charles Elkan. "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records". In: *Workshop on Research Issues on Data Mining and Knowledge Dis-*

covery, DMKD 1997 in cooperation with ACM SIGMOD'97, Tucson, Arizona, USA, May 11, 1997. 1997 (*cited on page* 61).

[Mil14] Bartosz Milewski. *Pure Functions, Laziness, I/O, and Monads*. 2014. URL: https://www.schoolofhaskell.com/school/starting-with-haskell/basics-of-haskell/3-pure-functions-laziness-io (*cited on page* 86).

[MQP18] Will Martino, Monica Quaintance, and Stuart Popejoy. *Chainweb Whitepaper*. 2018. URL: http://kadena2.novadesign.io/wp-content/uploads/2018/08/chainweb-v15.pdf (*cited on pages* 85, 89).

[MT15] Nikita Mishra and Abhradeep Thakurta. "(Nearly) Optimal Differentially Private Stochastic Multi-Arm Bandits". In: *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*. UAI'15. Amsterdam, Netherlands: AUAI Press, 2015, pp. 592–601. ISBN: 9780996643108. DOI: 10.5555/3020847.3020909 (*cited on pages* 111, 117–119, 139).

[Mun14] Rémi Munos. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning". In: *Foundations and Trends in Machine Learning* 7.1 (2014), pp. 1–129. DOI: 10.1561/2200000038 (*cited on pages* 16, 96).

[Nag18] Sebastian Nagel. *April 2018 Crawl Archive Now Available*. http://commoncrawl.org/2018/05/april-2018-crawl-archive-now-available/. 2018 (*cited on page* 49).

[Nak09] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf (*cited on pages* 84, 86).

[NIS01] NIST. *Advanced Encryption Standard (AES)*. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. FIPS Publication 197. 2001 (*cited on pages* 17, 18, 119).

[NRLS+21] Claudia Negri Ribalta, Marius Lombard-Platet, Camille Salinesi, and Pascal Lafourcade. "Blockchain Mirage or Silver Bullet? A Requirements-driven Comparative Analysis of Business and Developers' Perceptions in the Accountancy Domain". In: *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 12.1 (2021), pp. 85–110. DOI: 10.22667/JOWUA.2021.03.31.085. URL: https://doi.org/10.22667/JOWUA.2021.03.31.085 (*cited on page* 7).

[NY15] Moni Naor and Eylon Yogev. "Bloom Filters in Adversarial Environments". In: *Advances in Cryptology – CRYPTO 2015*. Ed. by Rosario Gennaro and Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 565–584. ISBN: 978-3-662-48000-7. DOI: 10.1007/978-3-662-48000-7_28 (*cited on pages* 12, 13, 57).

[Ode09] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. 1st. USA: Cambridge University Press, 2009. ISBN: 052111991X. DOI: 10.1017/CBO9780511721656 (*cited on page* 121).

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238. ISBN: 978-3-540-48910-8. DOI: 10.1007/3-540-48910-X_16 (*cited on pages* 19, 98–100, 119, 129).

[SBA20] Ariel Shtul, Carlos Baquero, and Paulo Sérgio Almeida. *Age-Partitioned Bloom Filters*. 2020. arXiv: 2001.03147 [cs.DS] (*cited on page* 62).

[Sea] *Microsoft SEAL (release 3.3)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. 2019 (*cited on page* 118).

[SLM14] Marta Soare, Alessandro Lazaric, and Rémi Munos. "Best-Arm Identification in Linear Bandits". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 828–836. DOI: 10.5555/2968826.2968919 (*cited on pages* 96, 112).

[SS18] Roshan Shariff and Or Sheffet. "Differentially Private Contextual Linear Bandits". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 4301–4311. DOI: 10.5555/3327144.3327342 (*cited on page* 119).

[SYB14] David Schwartz, Noah Youngs, and Arthur Britto. *The Ripple Protocol Consensus Algorithm*. 2014. URL: https://ripple.com/files/ripple_consensus_whitepaper.pdf (*cited on page* 84).

[SZ08] Hong Shen and Yu Zhang. "Improved Approximate Detection of Duplicates for Data Streams Over Sliding Windows". In: *Journal of Computer Science and Technology* 23.6 (2008), pp. 973–987. ISSN: 1860-4749. DOI: 10.1007/s11390-008-9192-1 (*cited on pages* 3, 28, 34, 49, 60, 62, 66).

[Tan11] Ole Tange. "GNU Parallel - The Command-Line Power Tool". In: *The USENIX Magazine* 36 (Jan. 2011), pp. 42–47. DOI: 10.5281/zenodo.16303 (*cited on page* 49).

[TD16] Aristide C. Y. Tossou and Christos Dimitrakakis. "Algorithms for Differentially Private Multi-Armed Bandits". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 2087–2093. DOI: 10.5555/3016100.3016190 (*cited on pages* 111, 117–119, 137, 139–141).

[Tho33] William R Thompson. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3/4 (1933), pp. 285–294. DOI: 10.2307/2332286. arXiv: 1708.05033 (*cited on page* 96).

[TRL12] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. "Theory and Practice of Bloom Filters for Distributed Systems". In: *IEEE Communications Surveys Tutorials* 14.1 (Jan. 2012), pp. 131–155. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.031611.00024 (*cited on page* 27).

[TS15] Stefan Thomas and Evan Schwartz. *A Protocol for Interledger Payments*. 2015. URL: https://interledger.org/interledger.pdf (*cited on pages* 84, 85).

[Vau02] Serge Vaudenay. "Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS..." In: *Advances in Cryptology — EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 534–545. ISBN: 978-3-540-46035-0. DOI: 10.1007/3-540-46035-7_35 (*cited on page* 26).

[Wag02] David Wagner. "A Generalized Birthday Problem". In: *Advances in Cryptology — CRYPTO 2002*. Ed. by Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 288–304. ISBN: 978-3-540-45708-4. DOI: 10.1007/3-540-45708-9_19 (*cited on page* 63).

[WL93] Pierre Wolper and Denis Leroy. "Reliable hashing without collision detection". In: *Computer Aided Verification*. Ed. by Costas Courcoubetis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 59–70. ISBN: 978-3-540-47787-7. DOI: 10.1007/3-540-56922-7_6 (*cited on page* 36).

[Yoo10] MyungKeun Yoon. "Aging Bloom Filter with Two Active Buffers for Dynamic Sets". In: *IEEE Trans. on Knowl. and Data Eng.* 22.1 (Jan. 2010), pp. 134–138. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.136 (*cited on pages* 3, 28, 31, 49, 60, 62, 67).

[You50] William J Youden. "Index for rating diagnostic tests". In: *Cancer* 3.1 (1950), pp. 32–35. DOI: 10.1002/1097-0142(1950)3:1<32::aid-cncr2820030106>3.0.co;2-3 (*cited on page* 62).

RÉSUMÉ

Dans cette thèse, nous nous intéresserons à deux thématiques majeures : la sécurité des données dans des protocoles d'échange cryptographiques, ainsi que l'analyse et la sécurisation des structures de données. Pour cela, nous appliquons des analyses formelles, reposant autant sur des outils cryptographiques existants que des modélisations ad-hoc pour le problème envisagé. En premier lieu, nous étudierons la sécurité de détecteurs de doublons, ainsi qu'une optimisation de l'utilisation de l'espace de stockage disponible. Ensuite, nous nous tournerons sur un cas pratique de modélisation, proposant un nouveau modèle formel pour la blockchain. Enfin, nous nous pencherons sur la problématique d'externalisation de protocoles de bandit manchot. Nous verrons comment externaliser les calculs, tout en prouvant que les nœuds du réseau apprennent aussi peu d'information que possible.

MOTS CLÉS

sécurité de l'information, modélisation, protocoles d'échange, optimisation, cryptographie appliquée

ABSTRACT

In this thesis, we are interested in two major themes: data security in exchange protocols, and the analysis and securing of data structures. To do so, we apply formal analyses, based on existing cryptographic tools as well as ad-hoc modelling for the problem under consideration. Firstly, we will study the security of duplicate detectors, as well as an optimisation of the use of available storage space. Then, we will turn to a practical modelling case, proposing a new formal model for blockchain. Finally, we will study the problem of outsourcing bandit learning protocols. We will see how to outsource calculations, while proving that the network learns as little as possible.

KEYWORDS

information security, modelisation, exchange protocols, optimisation, applied cryptography