# Smart contracts for auctions : from experimental assessment to privacy

Lucas Massoni Sguerra

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Mines Paris-PSL

# SMART CONTRACTS FOR AUCTIONS: FROM EXPERIMENTAL ASSESSMENT TO PRIVACY

*Contrats intelligents pour les enchères : de l'évaluation expérimentale à la confidentialité*


Soutenue par

**Lucas MASSONI SGUERRA**


Le 19 avril 2023

Ecole doctorale n° 621

**Ingénierie des systèmes, matériaux, mécanique, énergétique**

Spécialité

**Informatique temps-réel, robotique et automatique**

Composition du jury :

| | |
|---|---|
| Sylvain CONCHON<br>Professeur, Université Paris-Saclay | *Président* |
| Rachid GUERRAOUI<br>Professeur, EPFL | *Rapporteur* |
| Maria POTOP-BUTUCARU<br>Professeur, Sorbonne Université | *Examinateur* |
| Nour EL MADHOUN<br>Professeur associé, ISEP | *Examinateur* |
| Emilio Jesus GALLEGO ARIAS<br>Jeune chercheur, IRIF Inria | *Examinateur* |
| Pierre JOUVELOT<br>Directeur de recherche, Mines Paris | *Examinateur* |
| Gérard MEMMI<br>Professeur, Télécom Paris | *Co-directeur* |
| Fabien COELHO<br>Professeur, Mines Paris | *Directeur de thèse* |

MINES PARIS | PSL

*Distrust and caution*
*are the parents of security.*

BENJAMIN FRANKLIN

# ACKNOWLEDGEMENTS

*This work couldn't have materialized without the help and support of my family, friends and colleagues. Here I name a few of the people that inspired, encouraged and supported me during my period as a PhD student.*

*First, I'd like to thank my supervisors, Emilio J. Gallego Arias, Fabien Coelho, Gérard Memmi, and Pierre Jouvelot. Thank you for your guidance, support, rigor, and inspiring conversations. Special thanks to Pierre Jouvelot: thank you for all the Skype calls and time you took to work with me.*

*I also would like to thank the professors and staff from CRI (Centre de recherche en informatique), François Irigoin, Corinne Ancourt, Claude Tadonki, Claire Medrala, Olivier Hermant and Laurent Daverio; thank you for your support and aid.*

*Special thanks to my PhD colleagues Bruno Massoni Sguerra, Luc Perera, Adila Susungi, Patryk Kiepas, Robin Le Conte des Floris, Maksim Berezov and Dongmin Son: thank you for sharing your PhD experiences with me, and for the good times.*

*To my colleagues from Eniblock and The Sandbox, Maxime Vanmeerbeck, Yoann Thomas, Christophe Convert, Vincent Herbert, Alexis Figel, Andres Adjimann and Felipe Faria, thanks you for sharing your knowledge and enthusiasm for cryptography and blockchain systems.*

*Also, my friends Giuliano Tadeo Rodrigues Varela e Silva, Felipe Massao Kitanaka Matsuoka, Nelson Gomes, Amanda Fernandes and Juan Jeronimo Fuentes, thanks for your encouragement and support.*

*And finally, a special thanks to my family, specially my parents Ricardo Giaretta Sguerra and Christina Maria Massoni Sguerra: thank you for infusing me with a passion for knowledge and science and for always pushing me to reach my potential.*

# ABSTRACT

The advent of auctions for the allocation of sponsored search results on the Internet has brought auctions to the foreground of e-commerce. For instance, for each Google query submitted on the Internet, there are associated sponsored links displayed to the user, automatically sold by auctions, resulting in billions of auctions happening online everyday. Google sale of sponsored links alone is responsible for the movement of hundreds of billions of dollars per year.

Presently, the vast majority of auctions take place in centralized services, which requires auction participants to relinquish total control to the auctioneer, leaving the participants at the mercy of the system, with no choice other than to trust the service, hoping that the process is efficient and honest, i.e., that it makes good use of computer resources and won't disclose private information.

In this thesis, we study whether blockchain technologies, designed for providing trusted environments for the execution of programs (known in this context as "smart contracts"), provide an efficient and secure environment for auctions. To make this analysis more concrete, we selected a so-called "truthful" auction, known as Vickrey–Clarke–Groves for sponsored search (VCG for search), as the basis to assess the impacts of such environments on the auction process.

The first step for our research is the formalization of VCG algorithms, also called "mechanisms"; we give the specifications of the general VCG mechanism and of its VCG for search variant using the proof assistant Coq, together with the proofs of some of their key properties.

Following up on this sound basis for the definition of auctions, we describe our rationale for the reasoned choice of a sample of blockchain systems to be used as the basis for experimental research. We restricted our systematic and scientifically grounded study of blockchain systems to Ethereum and Tezos, Ethereum being the industry standard for smart contracts development and Tezos, a newer blockchain that intends to tackle some of Ethereum's performance issues by selecting a more efficient consensus mechanism, proof-of-stake, instead of the proof-of-work approach used in the original Ethereum. Our performance-comparison methodology focuses on the experimental evaluation of both system in terms of programmability, performance and cost.

Our tests indicate that, at the time of study, Ethereum surpassed Tezos in terms of programmability and community support, which led us to select it as the blockchain of choice to base the rest of our research upon. In addition, our benchmarking provides numerical evidence for the existence of significant scalability limitations for Ethereum and its proof-of-work consensus algorithm; such shortcomings were also noted by the Ethereum community, with scalability solutions being developed in the recent years. Due to their relevancy for the industry, we perform another benchmark comparison between two promising solutions: Layer-2 Polygon proof-of-stake extension and Ethereum's proof-of-stake Merge update. We used our VCG for search smart contract to perform a comparison between standard Ethereum and the scalability updates. This comparison reveals the

advantage of both upgrades, but in the case of Polygon, put to light the fact that its dependency on Ethereum's pricing remains a limitation.

Even though performance issues are clearly seen by the community as key to the acceptance of the blockchain technology, the transparency of public blockchains, though it is what ultimately makes the technology secure, presents another severe handicap for the execution of programs that expect a certain level of privacy, such as VCG for search. In the final chapter of this thesis, we analyze the effects of this lack of privacy on the VCG for search ecosystem. We explore some of the industry proposals for privacy, present three new proof-of-concepts variants of VCG for search that increase the privacy of this type of VCG contracts and analyze their effects on the privacy of the auction process as well as their efficiency and monetary impacts on the participants.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

CHAPTER 1

# INTRODUCTION

## 1.1 Contexte

*Les enchères sont des méthodes polyvalentes pour vendre des biens, et elles peuvent être adaptées à différents objectifs. Elles se déroulent tous les jours dans une grande variété de contextes, que ce soit pour vendre des antiquités, des produits alimentaires ou des annonces en ligne. L'impact économique de l'utilisation de ces processus est énorme et a été reconnu par le Prix de la Banque de Suède en sciences économiques en mémoire d'Alfred Nobel, considéré comme le prix "Nobel" en économie, décerné à Milgrom et Wilson en 2020 [1].*

*Certaines enchères ont des propriétés qui peuvent forcer les potentiels acheteurs, appelés "enchérisseurs", à adopter des stratégies d'enchères spécifiques. Notre intérêt dans cette thèse réside dans les enchères qui demandent aux enchérisseurs participants de révéler des informations jugées sensibles et/ou privées pour eux. Une enchère de cette variété est l'enchère dite Vickrey-Clarke-Groves (VCG) pour la recherche sponsorisée (nous utilisons également le nom de"VCG pour la recherche", en abrégé), qui est le principal objectif de ce travail. Cette technique d'enchère est une variante du mécanisme d'enchère VCG général, qui est considéré comme l'une des bases de la théorie des enchères ; nous discutons de l'importance de cette technique d'enchère dans la vie quotidienne, en particulier pour les moteurs de recherche sur Internet tels que Google, dans le Chapitre 4.*

*Actuellement, la majorité des enchères qui ont lieu dans le monde sont en ligne et utilisent des services centralisés. Dans ces services centralisés, il y a une autorité centrale qui applique les règles et contrôle les interactions entre les enchérisseurs et le vendeur (appelé "commissaire-priseur", dans ce contexte). Pour participer, les enchérisseurs doivent avoir confiance dans le système qui fournit les services ; et l'abus de cette confiance a déjà conduit à des enchères malhonnêtes [2] [3]. Afin d'encourager les enchérisseurs à participer aux enchères en évitant qu'elles ne soient injustes, une approche prometteuse peut consister à profiter de la transparence et de la confiance des systèmes décentralisés, c'est-à-dire les blockchains.*

*Les blockchains sont des systèmes distribués et décentralisés, d'abord introduits avec Bitcoin au travers de l'article de 2009 intitulé "Bitcoin: A Peer-to-Peer Electronic Cash System" [4], rédigé par un auteur mystérieux, Satoshi Nakamoto. Les blockchains sont des registres distribués composés de plusieurs machines, appelées "nœuds", qui communiquent entre elles et exécutent le protocole de la blockchain, qui soutient le réseau de la blockchain. Ces nœuds prennent en charge une base de données, en constante expansion, d'identifiants de compte et de soldes. Afin*

*de faciliter la réplication de l'état interne de la blockchain entre toutes les machines, la base de données est divisée en blocs qui sont cryptographiquement "chaînés" ensemble, d'où le nom "blockchain"(chaîne de blocs). Pour maintenir la cohérence des données entre les différents nœuds, ces blocs doivent être transparents, c'est-à-dire accessibles et lisibles par tous les participants de la blockchain, et leur contenu doit être facilement vérifiable pour éviter les mauvaises conduites des nœuds malveillants. Ces caractéristiques rendent les systèmes de blockchain particulièrement sécurisés.*

*La deuxième génération de blockchains, introduite par Ethereum [5], a ajouté la notion de "contrats intelligents" (*smart contract*) aux systèmes décentralisés. Les contrats intelligents sont des programmes indépendants qui résident à l'intérieur d'une blockchain et qui bénéficient des mêmes fonctionnalités de sécurité et de confiance que les données qui y sont stockées. Par conséquent, notre objectif initial dans cette thèse tourne autour de la tâche de mettre en œuvre VCG pour la recherche en tant que contrat intelligent et d'analyser les implications pratiques d'une telle implémentation.*

*La première étape est de sélectionner une plateforme blockchain pour notre implémentation. Dès le début de nos recherches, il est devenu évident que, lors de la comparaison des systèmes blockchain, l'industrie se concentre sur les mots à la mode et la publicité ; il y a un manque évident de rigueur scientifique dans la manière dont les comparaisons existantes (voir Chapitre 5) sont gérées. Nous avons donc décidé de généraliser notre recherche d'une plateforme blockchain en une comparaison de référence qui, nous l'espérons, ouvrira la voie à une définition plus appropriée et plus scientifique de la manière de comparer différents systèmes blockchain, du moins du point de vue des contrats intelligents.*

*Notre première comparaison de référence se concentre sur les systèmes blockchain Ethereum et Tezos. Ethereum est le système qui a introduit les contrats intelligents et reste toujours comme la plateforme la plus populaire pour eux, tandis que Tezos, un système blockchain plus récent, a fait des choix de conception intéressants pour résoudre certains problèmes présents dans les systèmes précédents, tels que Ethereum. Les deux systèmes ont des approches différentes en ce qui concerne les contrats intelligents, mais la différence la plus radicale réside dans le protocole de consensus qu'ils ont adopté, c'est-à-dire les algorithmes par lesquels les systèmes distribués s'accordent sur leur état actuel. Ethereum, à l'époque*[1]*, adoptait le mécanisme de consensus de preuve de travail (*proof-of-work *ou PoW, en anglais) d'abord adopté par Bitcoin, utilisé par les différents nœuds composant le réseau pour s'accorder sur un état commun par le biais de calculs intensifs. Bien que robuste, cette approche limite la scalabilité du système et utilise une énorme quantité d'énergie, ce qui est clairement un inconvénient de nos jours. Tezos adopte un mécanisme de preuve d'enjeu (*proof-of-stake *ou PoS, en anglais), une alternative à la preuve de travail, dans lequel les utilisateurs bloquent des jetons comme garantie pour avoir le droit de produire des blocs pour la blockchain. Notre comparaison de référence a utilisé des implémentations de VCG pour la recherche pour ces deux systèmes comme base de comparaison. Nous avons évalué les deux*

[1] Ethereum a mis à jour son consensus sur la preuve d'enjeu le 15 septembre 2022

*systèmes en termes de programmabilité et de performances.*

*Cette première comparaison a révélé des lacunes dans la plateforme Ethereum, en termes de scalabilité et de coût d'utilisation, quelque chose qui a également été remarqué par les membres de la dite "crypto-sphère" et qui a poussé l'industrie à rechercher des solutions à ces problèmes tout en profitant de la part de marché et de la fiabilité d'Ethereum. Une solution préconisée par la communauté était de mettre à jour le protocole Ethereum, ce qui nous a logiquement conduits à réaliser une deuxième étude comparative, cette fois en se concentrant uniquement sur Ethereum et sur la proposition la plus prometteuse pour résoudre ses problèmes de scalabilité, la "couche-2" intégrée à Polygon PoS et la mise à jour du consensus d'Ethereum connue sous le nom de "Merge" (voir chapitre 5). Polygon PoS est une blockchain de preuve d'enjeu caractérisée comme une "couche-2" pour Ethereum, car elle fonctionne en parallèle du réseau Ethereum et permet aux utilisateurs d'Ethereum d'utiliser la chaîne de Polygon pour gérer les données d'Ethereum. D'autre part, la mise à jour Merge d'Ethereum est la réponse d'Ethereum à ses problèmes de scalabilité, une proposition visant à "fusionner" la chaîne d'Ethereum avec une autre chaîne de preuve d'enjeu, afin de fonctionner comme la couche de consensus pour Ethereum, en permettant la transition du consensus de la preuve de travail vers la preuve d'enjeu* [2].

*Si l'approche de référence introduite ici répond aux exigences d'efficacité des contrats intelligents d'enchères, les enchères telles que VCG pour la recherche nécessitent également, en plus d'un certain niveau d'efficacité, un certain degré de confidentialité. Comme indiqué précédemment, ce type d'enchères exige en effet des enchérisseurs qu'ils révèlent des informations sensibles. Notre dernière étape dans cette thèse consiste donc en une analyse de cette "exigence de confidentialité", que nous considérons comme d'une importance capitale si l'on souhaite que les enchères soient utilisées dans la pratique sur les blockchains. Nous proposons ici l'adoption de certaines techniques de cryptographie afin de compenser ce manque de confidentialité, tout en préservant une certaine transparence qui incitera les enchérisseurs à participer.*

## 1.2 Principaux résultats

*Les principaux résultats de ce travail de recherche sont les suivants.*

### 1.2.1 Spécification et implémentation de contrats intelligents pour VCG pour la recherche sponsorisée

*Pour fournir un point de départ solide à notre travail, nous donnons une spécification générale de VCG et VCG pour la recherche en Coq, ainsi que les preuves de certaines de ses propriétés clés. Nous fournissons ensuite des implémentations de VCG (Vickrey-Clarke-Groves) pour la recherche sponsorisée sous forme de contrats intelligents, dans les langages Solidity et SmartPy, en ciblant les blockchains Ethereum et Tezos.*

[2] La fusion d'Ethereum a eu lieu le 15 septembre 2022 [6]

### 1.2.2 Comparaison de référence entre Ethereum et Tezos

*Nous concevons et réalisons une expérience de comparaison de référence pour deux systèmes blockchain, Ethereum et Tezos, avec deux algorithmes de consensus différents, preuve de travail et preuve d'enjeu. Notre comparaison se concentre sur les contrats intelligents, ce qui distingue notre travail des comparaisons axées sur les performances présentes dans l'industrie et les articles académiques.*

*Les résultats de la comparaison de référence ont été rassemblés dans un article intitulé "*Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos*" (Évaluation des performances des blockchains : un cas d'utilisation de contrat intelligent d'enchère VCG pour Ethereum et Tezos) [7] et présenté lors du quatrième Symposium international sur les fondements et les applications de la blockchain en 2021 (*Fourth International Symposium on Foundations and Applications of Blockchain, FAB '21).

### 1.2.3 Comparaison de référence pour les solutions de scalabilité d'Ethereum

*Nous concevons et réalisons une deuxième expérience de comparaison de référence, une fois encore du point de vue des contrats intelligents et fondée sur le contrat VCG pour la recherche, mais cette fois en comparant le réseau Ethereum avec deux propositions de solution de scalabilité : le protocole Polygon PoS de la couche-2 et la mise à jour "Merge" de la preuve d'enjeu d'Ethereum.*

## 1.3 Structure de la thèse

*Cette thèse est composée de 6 parties principales. Après cette introduction initiale, qui pose le contexte de notre travail, le chapitre 3 commence par présenter les bases techniques et théoriques des technologies de la blockchain. Il commence par une présentation générale des registres distribués avec Bitcoin, suivie d'une introduction à d'autres systèmes tels que Ethereum, Polygon PoS et Tezos. Le chapitre se concentre principalement sur Ethereum, car c'est la blockchain dont les contrats intelligents ont servi de base à une grande partie du travail de cette thèse.*

*Le chapitre 4 présente l'algorithme de l'enchère VCG pour la recherche, et certaines notions de théorie des jeux et des enchères sont introduites afin d'illustrer au mieux les caractéristiques de VCG pour la recherche. Nous présentons également une modélisation du mécanisme VCG général dans l'assistant de preuve Coq ; l'enchère VCG pour la recherche est une instance de ce mécanisme général dont elle hérite toutes les propriétés utiles. Le chapitre se termine par une contextualisation de l'utilisation de VCG dans l'industrie aujourd'hui.*

*Le chapitre 5 présente des implémentations de VCG pour la recherche sous forme de contrats intelligents ciblant les blockchains Ethereum et Tezos. Ces contrats sont utilisés pour réaliser des comparaisons de référence entre différentes plates-formes pour les applications décentralisées. La première comparaison met en confrontation Ethereum et Tezos, tandis que la seconde aborde les solutions de mise à l'échelle d'Ethereum, notamment Polygon en tant que couche-2 et la mise à jour*

*“Merge” d’Ethereum. Les deux comparaisons de référence se concentrent sur la programmabilité, les performances et les coûts financiers.*

*Les considérations concernant la confidentialité de notre implémentation de VCG sont présentées dans le chapitre 6, où nous discutons des conséquences du manque inhérent de confidentialité de la blockchain sur les enchères telles que VCG. Nous présentons des travaux connexes traitant de la confidentialité des enchères à l’intérieur et à l’extérieur de la blockchain et clôturons le chapitre en présentant trois solutions de preuve de concept qui tentent d’augmenter la confidentialité des systèmes basés sur la blockchain.*

*Enfin, le chapitre 7 conclut notre document en récapitulant les principales contributions et en présentant nos intentions pour les travaux futurs.*

CHAPTER 2

# INTRODUCTION

## 2.1 Context

Auctions are versatile methods for selling goods, and they can be tailored for different purposes. They occur everyday in a wide variety of settings, from selling antiques or food products to placing advertisements. The economic impact of the use of such processes is huge, and has been recognized by the Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel, considered as the ”Nobel” prize in economics, granted to Milgrom and Wilson in 2020 [1].

Some auctions have properties that can force potential buyers, known as “bidders”, to adopt specific bidding strategies. Our interest in this thesis lies in auctions that demand participating bidders to reveal information deemed sensitive and/or private to them. One auction of this variety is the so called Vickrey–Clarke–Groves (VCG) auction for sponsored search (we also use the name “VCG for search”, for short), which is the main focus of this work. This auction technique is a variant of the General VCG auction mechanism, which is considered as one of the cornerstones of auction theory; we discuss the importance of this auction technique in everyday life, in particular the search engines on the Internet such as Google, in Chapter 4.

Presently, the majority of auctions happening in the world are online, and use centralized services. In centralized services, there is a central authority that enforces the rules and controls the interactions between the bidders and the seller (known as “auctioneer” in this context). In order to participate, bidders have to trust the system providing the services; this has already led to some dishonest auctions [2] [3]. In order to increase the incentive for bidders to participate in auctions by preventing the latter to be unfair, a promising approach can be to take advantage of the transparency and trust of decentralized systems, i.e., blockchains.

Blockchains are distributed and decentralized systems, first introduced with Bitcoin through the 2009 paper “Bitcoin: A Peer-to-Peer Electronic Cash System” [4] by a mysterious author, Satoshi Nakamoto. Blockchains are distributed ledgers composed of multiple machines, known as “nodes”, that communicate with each other and execute the blockchain protocol, which supports the blockchain network. These nodes support an ever growing database of accounts and balances. In order to facilitate the replication of the blockchains’s internal state between all machines, the database is divided into blocks that are cryptographically “chained” together, hence the name “blockchain”. In order to maintain the data consistent between the different nodes, these blocks need to be transparent, i.e., accessible and readable by any blockchain participant, and their contents need to be easily verifiable, to avoid misbehavior from malicious nodes. These character-

istics make blockchain systems particularly secure.

The second generation of blockchains, introduced by Ethereum [5], added the notion of "smart contracts" into decentralized systems. Smart contracts are independent programs that live inside a blockchain and enjoy some of the same security and trust features as the data stored there. Therefore, our initial goal in this thesis revolves around the task of implementing VCG for search as a smart contract, and analyse the practical implications of such an implementation.

The first step for doing so is to select a blockchain platform for our implementation. Early in our research, it became apparent that when comparing blockchain systems, the industry focuses on buzzwords and advertisement; there is a clear lack of scientific rigour in the way the existing comparisons (see Chapter 5) are managed. We thus decided to develop our search for a single blockchain platform into a benchmark comparison that, we hope, paves the way towards defining a proper, more scientific manner to compare different blockchain systems, at least from a smart-contract point of view.

Our first benchmark focuses on the Ethereum and Tezos blockchains systems. Ethereum is the system that pioneered smart contracts and is still the most popular platform for them, while Tezos, a newer blockchain system, made some interesting design choices to tackle some of the issues present in previous systems, such as Ethereum. Both system have different approaches to smart contracts, but the one drastic difference is the consensus protocol they adopted, i.e., the algorithms through which distributed systems agree on their current state. Ethereum, at the time [1], adopted the Bitcoin-style proof-of-work (PoW) consensus mechanism, used by the different systems composing the network to agree on a common state through intense computation. Although robust, this approach limits the scalability of the system and uses a huge amount of energy, a clear drawback nowadays. Tezos adopts a proof-of-stake (PoS) mechanism, an alternative to proof-of-work, in which users stake coins as collateral for the right to produce blocks for the blockchain. Our benchmark comparison used implementations of VCG for search for both systems as a basis of comparison. We evaluated both systems in terms of programmability and performance.

This first comparison revealed shortcomings in the Ethereum platform, in terms of scalability and price of usage, something that was also noticed by the members of the so-called "crypto-sphere", and which has pushed the industry to look for solutions to these issues while still enjoying Ethereum's market share and reliability. One solution advocated by the community was to upgrade the Ethereum protocol, which logically led us to perform a second benchmark study, this time focusing on Ethereum only and the most promising proposal for fixing its scalability problems, the "Layer-2", embedded into Polygon PoS and the Ethereum's consensus update known as the "Merge" (see Chapter 5). Polygon PoS is a proof-of-stake blockchain characterized as a "layer-2" for Ethereum, for it runs in parallel to the Ethereum network, and enables Ethereum users the option to use Polygon's chain to handle Ethereum data. On the other hand, the Ethereum Merge update is Ethereum's own response to its scalability issues, a proposal to "merge" Ethereum's chain with another proof-of-stake chain, to function as the consensus layer for Ethereum, updating the

[1] Ethereum updated its concensus to proof-of-stake on September 15, 2022 [6]

proof-of-work consensus to proof-of-stake [2].

If the benchmark approach introduced here addresses the efficiency requirements of auction smart contracts, auctions such as VCG for search also require, in addition to a certain level of efficiency, a certain degree of privacy. As previously stated, this type of auctions indeed requires bidders to reveal sensitive information. Our last step in this thesis is thus an analysis of this "privacy requirement", which we believe to be of key importance if one wants auctions to be used in practice on blockchains. We propose here the adoption of some cryptography techniques in order to counterbalance this lack of privacy, while still preserving some of the transparency that will incentivize bidders to participate.

## 2.2 Main results

The main results of this research work are the following.

### 2.2.1 Specification and implementation of smart contracts for VCG for sponsored search

To provide a sound starting point for our work, we give a general specification of VCG and VCG for search in Coq, together with the proofs of some of its key properties. We then provide implementations of the VCG (Vickrey–Clarke–Groves) for sponsored search in smart-contract form, in both Solidity and SmartPy languages, targeting the Ethereum and Tezos blockchains.

### 2.2.2 Benchmark comparison between Ethereum and Tezos

We design and perform a benchmark-comparison experiment for two blockchain systems, Ethereum and Tezos, with two different consensus algorithms, proof-of-work and proof-of-stake. Our comparison focuses on smart contracts, which distinguishes our work from the performance-driven benchmarks present in the industry and academic papers.

The benchmark comparison results were gathered in an article titled "Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos" [7] and presented at the Fourth International Symposium on Foundations and Applications of Blockchain 2021 (FAB '21).

### 2.2.3 Benchmark comparison for Ethereum scalability solutions

We design and perform a second benchmark experiment, once again from the perspective of smart contracts and based on the VCG for search contract, though this time comparing the Ethereum network with two of its scalability solution proposals: the layer-2 Polygon PoS and Ethereum's proof-of-stake Merge update.

[2] Ethereum Merge took place on September 15, 2022 [6]

### 2.2.4 Privacy-preserving proof-of-concept solutions

We perform an in-depth analysis of the effects of the inherent lack of privacy induced by blockchain systems to our VCG for search contract, an auction that demands privacy. Based on this analysis, we present three privacy-preserving proof-of-concept solutions for our VCG for search contract. Our proof-of-concept proposals are incremental and use cryptography techniques.

## 2.3 Thesis structure

This thesis has 6 main parts. After this initial introduction, which sets the context of our work, Chapter 3 starts by presenting the technical and theoretical background of blockchain technologies. It begins with a general presentation of distributed ledgers with Bitcoin, followed by an introduction to other systems, i.e., Ethereum, Polygon PoS and Tezos. The main focus of the chapter is Ethereum, as it is the blockchain whose smart contracts were the basis of much of this thesis' work.

Chapter 4 presents the VCG for search auction algorithm, and some notions of game and auction theory are introduced in order to better illustrate VCG for search characteristics. We also present a modelization of the general VCG mechanism in the Coq proof assistant; the VCG for search auction is an instance of this general mechanism from which it inherits all the useful properties. The chapter ends with a contextualization of the usage of VCG in the industry today.

Chapter 5 presents implementations of VCG for search in smart-contract form targeting the Ethereum and Tezos blockchains. These contracts are used to perform benchmark comparisons between different platforms for decentralized applications. The first benchmark puts face-to-face Ethereum and Tezos, and the second one addresses Ethereum's scaling solutions, specifically the layer-2 Polygon and the Ethereum Merge update. Both benchmarks comparisons focus on programmability, performance and monetary cost.

Considerations over the privacy of our VCG implementation are presented in Chapter 6, where we discuss the consequences of the blockchain's inherent lack of privacy on auctions such as VCG. We present related work dealing with privacy of auctions inside and outside of the blockchain, and close the chapter by presenting three proof-of-concept solutions that attempt to increase the privacy of blockchain-based system.

Finally, Chapter 7 concludes our document, recapitulating the key findings and presenting our intentions for future work.

## CHAPTER 3

# BACKGROUND

*Dans ce chapitre, nous explorons les bases conceptuelles et techniques des systèmes de blockchain qui servent de fondement à cette thèse et qui sont nécessaires pour une compréhension adéquate de nos analyses et conclusions. Le lecteur averti peut choisir de sauter cette partie, bien que nous recommandions au moins de la survoler dans tous les cas, afin d'avoir une idée des notions sur lesquelles nous nous appuyons, étant donné leur évolution rapide en raison du manque de maturité de ce domaine en plein essor.*

*La section 3.1 introduit certaines définitions cryptographiques couramment utilisées dans les blockchains, y compris celles utilisées dans cette thèse. La section 3.2 présente certains concepts des systèmes de blockchain, avec une présentation de Bitcoin. Les aspects approfondis des systèmes de blockchain et de leur infrastructure sont présentés et discutés dans la section 3.3. Dans la section 3.4, la blockchain Ethereum et ses contrats intelligents sont discutés. La section 3.5 aborde les problèmes de passage à l'échelle (*scalability*) du mécanisme de preuve de travail (*proof-of-work *ou PoW) d'Ethereum et explore également les blockchains de preuve d'enjeu (*proof-of-stake *ou PoS) pertinentes pour cette thèse, telles que Tezos, Polygon PoS et le Merge d'Ethereum. Enfin, la section 3.6 met en évidence les outils utilisés tout au long de cette thèse.*

In this chapter we explore the conceptual and technical background of blockchain systems that serve as a basis for this thesis and are necessary for a proper understanding of our analyses and findings. The knowledgeable reader may want to skip this part, although we advise at least skimming over it in all cases, to get a feel of the notions that we build upon, since there are rapidly changing ones, given the lack of maturity of this burgeoning field.

Section 3.1 introduces some cryptographic definitions commonly used in blockchains, including those employed in this thesis. Section 3.2 presents some concepts of blockchain systems, with a presentation of Bitcoin. In-depth aspects of blockchain systems and their infrastructure are presented and discussed in Section 3.3. In Section 3.4, the Ethereum blockchain and its smart contracts are discussed. Section 3.5 addresses the scalability issues of Ethereum's proof-of-work (PoW) mechanism and also explores proof-of-stake (PoS) blockchains relevant for this thesis, i.e., Tezos, Polygon PoS, and the Ethereum Merge. Finally, Section 3.6 highlights the tools utilized throughout this thesis.

## 3.1 Cryptography

Before we delve into blockchain systems, some ideas about key notions of cryptography are needed, namely hash functions and public key cryptography.

### 3.1.1 Definition

Cryptography, from "crypt" meaning "hidden" and "graphy" meaning "writing" [8], is a group of techniques that enables the communication of information between two parties in a secure context, i.e., such that the transferred message is not shared with undesirable onlookers.

In cryptography, one defines *plaintext* as the readable information, and *ciphertext* as the encoded, hidden plaintext, that is decodable neither by untrusted humans nor machines. The process of transforming plaintext into ciphertext is called *encryption*; the reverse process is known as *decryption* [9].

### 3.1.2 Hash functions

Hashing is the process of applying a mathematical function known as a *hash function* over inputs of any arbitrary size and generating a fixed-size output. Hashes are, in general, meant to be irreversible, one-way functions, i.e., a function that is easy to compute but difficult to invert.

There are many different hash functions in the literature; our interest in this thesis is mainly focused on the hash functions adopted by Bitcoin and Ethereum, two blockchain systems (more about them in the following section). We are also interested in hash functions as a way of one-way encryption.

Bitcoin is built upon SHA-256 (Secure Hash Algorithm 256); SHA-256 generates an almost unique 256-bit signature for an input. Ethereum adopts Keccak-256; similarly to SHA-256, it also produces an 256-bit output from different-lengths inputs. Both hash functions belong to the SHA (Secure Hash Algorithm) family of cryptographic hash functions published by the U.S. National Institute of Standards and Technology.

Hash functions can be used for one-way encryption; in this type of encryption of messages, a ciphertext can be produced, but it cannot be decrypted back into plaintext. Note that, though one cannot decipher the ciphertext, it is easy to prove the knowledge of the plaintext by hashing it again and generating the same resulting ciphertext.

One drawback of hashes for one-way encryption is that these techniques are susceptible to brute-force attacks, since they are not very expensive to execute, and hash functions are deterministic; a trial-and-error approach to guess the plaintext input is, in theory, possible.

### 3.1.3 Key Cryptography

Key cryptography is a type of cryptography characterized by the usage of cryptographic "keys" to encrypt/decrypt plaintext. Keys are strings of characters, used within an encryption algorithm to

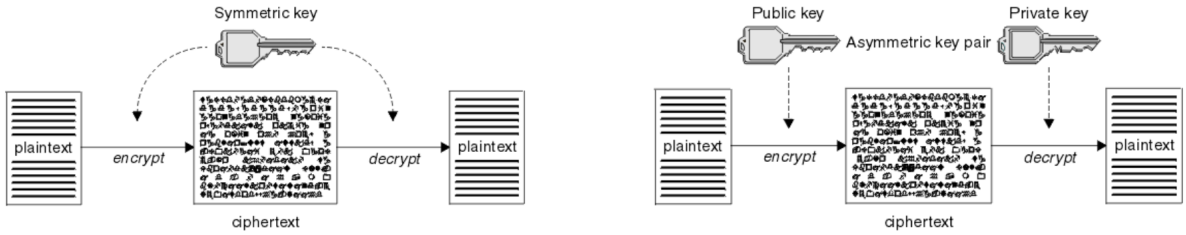


Figure 3.1: Symmetric and asymmetric key encryption (from [9])

transform plaintext into ciphertext [10]. There are two types of such an application.

**symmetric key cryptography** This technique requires both parties to use the same secret key.

**asymmetric key cryptography** The other approach is characterized by the usage of a pair of keys, one key for encryption and a different one for decryption. The key used for encryption is known as the "public key"; this key is only used for encryption, and can be shared publicly without compromising the integrity of the process. The other key, known as the "private key", can decrypt the ciphertext produced by the public key, and it needs to be kept in secret.

Figure 3.1 represents both symmetric and asymmetric key cryptographic techniques in a diagram.

## 3.2 Bitcoin and blockchain

The Blockchain technology was first introduced in Satoshi Nakamoto's paper [4], which describes an electronic payment system with no central authority. Until 2009 and the first Bitcoin transfer [11], virtual-money transfer systems were, indeed, always dependent on a central authority. Central authorities were deemed necessary to ensure the security and reliability of such systems; for instance, banks or credit card companies act for most financial transactions.

The Bitcoin system, instead of building upon a central authority, is distributed among multiple "nodes", i.e., machines that execute a dedicated blockchain protocol and communicate with each other, this way supporting the Bitcoin blockchain network. These nodes manage an ever-growing database of money transfers that is divided into blocks, all cryptographically "chained" together, hence the name "blockchain".

Nakamoto's proposal for this new distributed electronic payment system addresses two main challenges. First, it had to ensure security and fairness in the absence of a governing authority. The solution adopted to reach such a requirement was the use of specific cryptographic techniques, which allow users to verify the correctness of the information being transmitted in the system and the authenticity of the data stored there. The other challenge was the issue of coordinating the

decentralized nodes to produce and reach an agreement on the values stored in the blocks of data; the solution was the adoption of a dedicated consensus mechanism to be implemented by all nodes.

At the year of this writing, 2022, Bitcoin has become incredibly popular, with an estimated market cap, i.e., total value of existing bitcoins, of 700 billion dollars; the type of virtual money proposed initially by Nakamoto has gained its own terminology, “cryptocurrency”. The follow-up platforms such as Ethereum, Polygon or Tezos that adopt similar cryptographical strategies as Bitcoin’s have been popularized with the term “blockchain”, with new generations emerging at a rapid pace with more functionalities and improvements.

## 3.3 Blockchain infrastructure

In this section, we present the basic infrastructure and fundamentals of blockchain technologies that are present in one way or another in all platforms for currency exchange or smart contract execution.

### 3.3.1 Nodes

Blockchains are supported by a distributed network of computers, known as “nodes”. Nodes run the software application, known as “client”, necessary for the creation, transmission and verification of blocks.

Nodes are crucial for the blockchain communication, since these nodes are responsible for the Peer-to-Peer communication between clients. They expose a REST API (Representational State Transfer) that enables users to communicating with the chain and submitting transactions that, if accepted, can be included in a block and update the internal storage of the blockchain (more about transactions and block creation in Section 3.3.5)

Nodes are also responsible for keeping the blockchain records, and updating it when each new storage update operation is performed. This can be very demanding in terms of storage, so there are other options such as the so-called “light nodes”.

Finally, the nodes that take part in the block creation process, i.e., aggregate user transactions in the data structure, that is a block, which will then update the internal storage of the blockchain, are known as “miners”; the block creation process is known as “mining”.

### 3.3.2 Accounts and keys

Each user is represented by an account in a blockchain system; accounts can ”hold” coins and interact with other accounts.

In order to be able to use a blockchain, users need to hold a pair of cryptography-based private-public keys. The private key is used to ”sign” transactions, which are messages between accounts (more about transactions in Section 3.3.3). And, with the public key, users can verify the validity

of a signature. These two types of keys are what characterizes, as we already saw in section 3.1.3, asymmetric cryptography.

These keys are managed via so-called “wallets”. The word “wallet” is, in fact, misleading as it does not hold coins; crypto-coins are stored directly in the distributed ledger that is a blockchain. A (crypto) wallet is a piece of software for the creation of pairs of private-public keys; wallets can also store the keys and facilitate the user interactions with the blockchain by providing signature capabilities and querying the chain for accounts’ information.

### 3.3.3 Transactions

Transactions are the sole method that enables accounts to interact with the storage of blockchains. In the simplest case, an user constructs their desired transaction with information such as the transaction’s user target and the amount of coins to transfer; this is known as ”forging” a transaction. Transactions are not free, and updating the internal state of the whole blockchain requires payment, in form of a “fee” from users. Different blockchains have different approaches for calculating said fees; we expand on this concept in Section 3.4.4.

Once the transaction is ready, the user needs to sign it with their private key. The signed transaction is then submitted to a node; this node will validate the contents of the transaction such as verifying the signature and checking the validity of its values.

Transactions validated by a node are then added to the node’s memory pool (“mempool”). Mempools are buffer zones [12] where transactions wait to be selected and treated by a miner. Each node has their own mempool. While a transaction is waiting to be treated, the node broadcasts the existence of this ”pending” transaction to its peers, a process that replicates the transaction throughout the entire network of nodes.

Miners can select transactions as they wish, though usually giving preference to those with higher fees, and form blocks with them. Through a process called “consensus” (we detail consensus mechanisms in Section 3.3.6), one miner will be able to publish their block to the whole network; this block will be appended to the already existing chain of blocks as the new head of chain. As part of this process, the transactions belonging to this block take effect, updating the internal storage of the chain.

### 3.3.4 Blocks

Blocks are, as we just saw, batches of transactions aggregated by a mining node. A blockchain block can have different configurations, a block is usually composed of information such as:

- the identifier of the block, in form of the block hash;
- the transactions present in the block;
- a timestamp, i.e., the time of the block publication;

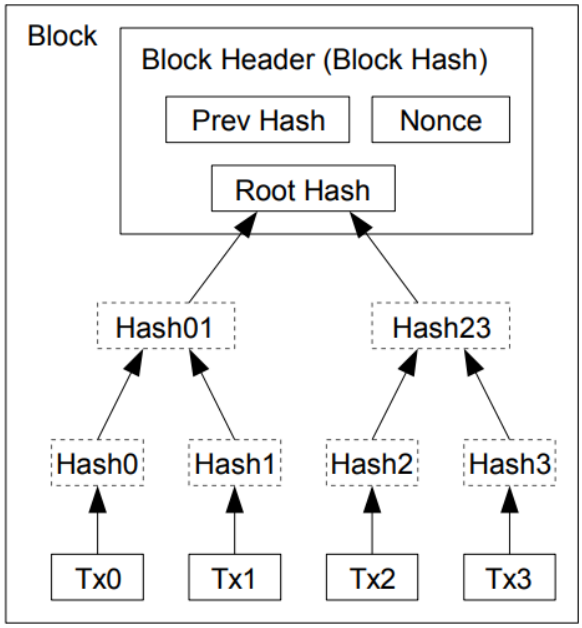


Figure 3.2: Bitcoin block with Merkle tree [4]

- a nonce, derived from the consensus mechanism;
- a Merkle tree root;
- the previous hash.

A Merkle tree, or hash tree, is a data structure used for data verification. The tree is composed of the hashes of its child nodes [13]. Figure 3.2 represents the Merkle tree behind the Merkle root of an example block. The leaves are constituted by the IDs[1] of the transactions in the block, and a tree node is (recursively) composed of the hashes of its children values. For example, in the figure, $Hash01$ is generated by $\mathrm{H}(\mathrm{Hash0}, \mathrm{Hash1})$ [2], for the transaction IDs Tx0 and Tx1.

If a malicious node changes one of the transaction of the block, it would alter the transaction ID, and the change would have an direct impact on the subsequent nodes, changing the root hash, which would indicate fraud.

Blocks are thus batches of transactions with a hash of the previous block in the chain. This links blocks together (in a chain) because hashes are cryptographically derived from the block data. This

[1] In the case of Bitcoin, transaction IDs are calculated by hashing the transaction data twice, with SHA-256 [14]

[2] For Bitcoin, the hash function $H$ is the double SHA-256 hashing of its argument: $H(i)$ = SHA-256(SHA-256($i$)) [15]

prevents fraud, because one change in any block in history would invalidate all the following blocks as all subsequent hashes would change and everyone running the blockchain would notice.

Once a block is published, it should be replicated in all nodes as the top block of the chain. Note that, and this is a key point for ensuring the validity of the blockchain at all times, blocks cannot be erased.

### 3.3.5 Mining

Mining is the process by which miner nodes publish a new ”head” block for the blockchain. Miners are constantly building blocks with transactions from the mempool, processing these into a block and then trying to publish this new formed block into the blockchain. Miners are constantly competing to be able to publish their blocks; there is a consensus mechanism that assists the nodes into reaching an agreement on which block will be the new head of the blockchain. We detail what a consensus mechanism is in Section 3.3.6 and expand on the different types adopted by the industry.

A miner that successfully publishes their block gets rewarded. In Bitcoin, rewards are divided in two parts: the transaction fees from all the users whose transactions were added to the block; and some coins that the miner rewards himself with. This self-reward is a special type of transaction, named “coinbase transaction”; it mints new Bitcoin coins into the system. The initial coinbase reward for mining a block in Bitcoin was 50 BTC; this reward is halved every 210,000 blocks, until the system reaches its maximum number of Bitcoins, that being set at 21 million. Once no more coins can be minted, the reward will be solely based on the transaction fees. This design-built limited number of coins is what is creating the scarcity of this resource, and thus its ultimate value (as would be for a physical resource such as gold).

### 3.3.6 Consensus

In a decentralized system such as a blockchain, nodes in this peer-to-peer network need help reaching agreement on the system internal global state. Consensus protocols are rules that the distributed nodes need to follow in order to reach a general agreement between the blockchain participants.

In the case of blockchains, consensus means agreeing on which is the most recent block of the blockchain, which will henceforth update the global state of the chain. Thus, for blockchain systems, consensus is a protocol to determine which of the miner nodes has the right to publish a block. The most popular forms of consensus for blockchains are proof-of-work (PoW) and proof-of-estate (PoS); the sections 3.4.6, 3.5.1.1, 3.5.2.1 and 3.5.3.1 present both protocols in more detail.

#### 3.3.6.1 Forks

A so-called “fork” appears when a blockchain splits, creating an alternative chain. For instance, there are temporary forks that occur whenever there is a time-limited communication failure between the network nodes. In this case, the subset of the disconnected nodes will select a different

last block to maintain the blockchain; this will cause two different chains to grow. Eventually, if and when the communication is reestablished, one of the chains will most likely be longer than the other; the protocol specifies that, in such a case, the shorter one will be abandoned; this design principle is known as the "longest chain rule"; Figure 3.3 represents said rule.

A fork could also occur when more than one block are created at about the same time by multiple miners. Since nodes can only validate a single new block, the spurious blocks will be abandoned by the network; these abandoned blocks are known as "uncle blocks", and the miners are nonetheless rewarded for mining them, though usually the reward is smaller than for a normal block.

Forks are also related to changes in the blockchain's consensus mechanism. Updates to the blockchain consensus algorithm that modify the production and validation of blocks are also known as forks. The type of such a fork depends on whether the change is drastic or not. Changes in which non-upgraded nodes, i.e., those that still don't conform to the updated protocol, can still validate and produce blocks lead to so-called *soft forks*. On the other hand, changes that make non-upgraded nodes unable to participate according to the new consensus are called *hard fork*. A hard fork essentially creates a new blockchain, different from the original one while keeping the block history until that point; Figure 3.4 represents the two types of forks.

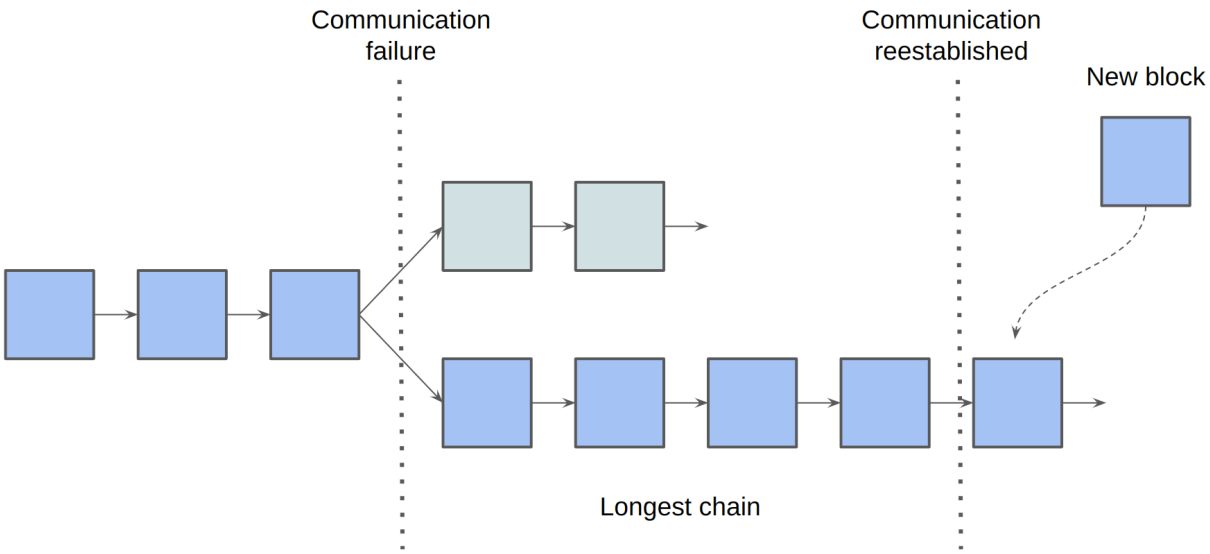


Figure 3.3: Schema describing the longest chain rule: a fork is generated following a communication failure, and, after the reestablishment of the network, a new minted block will chose the longest chain to continue, while the shorter one is abandoned.

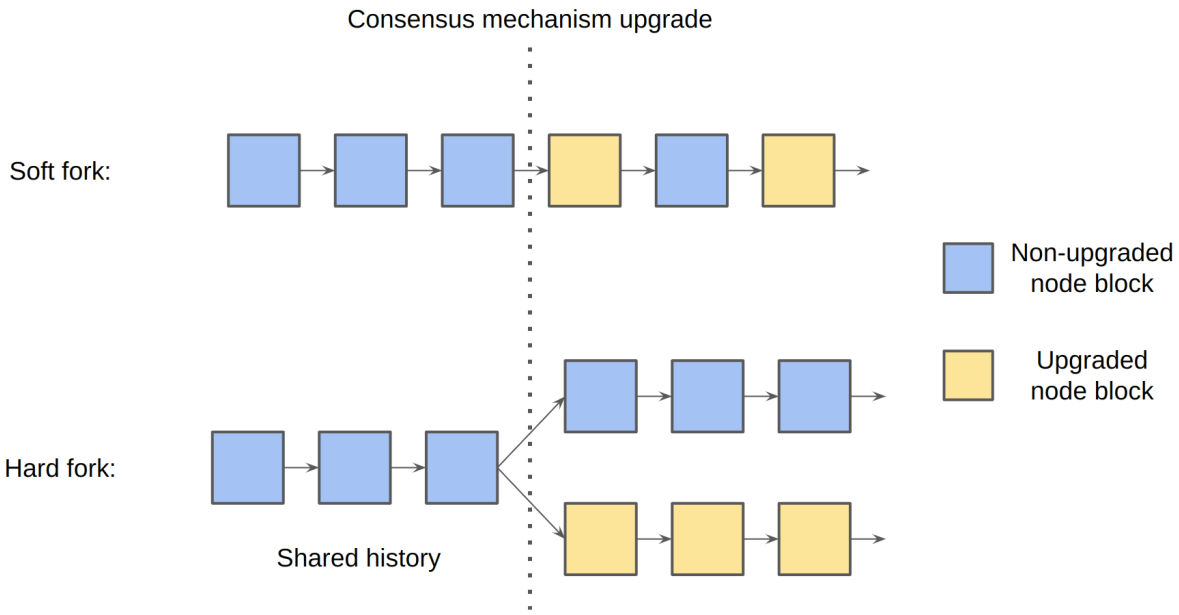


Figure 3.4: Graphical representation of the different types of forks. A soft fork will allow non-upgraded nodes to continue to publish blocks, while a hard fork won't, creating two separated chains with a shared history.

#### 3.3.6.2 Finality and Confirmation blocks

Finality is the assurance that a transaction cannot be changed, without significant loss for the blockchain, in terms of mined coins. Finality refers to the amount of time users need to wait to consider their transaction secure in the blockchain; waiting the stipulated time, or number of blocks, assures the user that their transaction isn't on a uncle block or a temporary fork [16] [17].

A common approach to measuring finality is by a number of *confirmation blocks*. Bitcoin, for example, specifies 6 confirmation blocks, which means that, after waiting for 6 new blocks to be added to the blockchain they are using, users are guaranteed that their transaction won't be reverted, as could be the case if they were on the short side of a fork.

#### 3.3.6.3 Mainnet and Testnets

Testnets are instances of a blockchain system with no real-world value linked to its assets. These networks are usually running the same software as the main network of the said blockchain, which is normally referred to as the "mainnet".

Testnets are mainly used for, as their name suggests, testing purposes and are managed by the

community to help build adhesion to an existing infrastructure.

## 3.4 Ethereum: a blockchain as a distributed computer

Ethereum was first introduced in 2014 within a white paper [5] by Vitalik Buterin. Ethereum marked the beginning of the second generation of blockchains, expanding the functionalities of bitcoins with a scripting language that allows the system to house decentralized applications. Becoming live in 2015, Ethereum is presently the second most valuable blockchain system after Bitcoin, with a market cap of 350 billion dollars.

### 3.4.1 Smart contracts

Ethereum introduced a special type of account in its chain, accounts that, instead of being controlled by a pair of public-private keys, are controlled by code. These accounts are known as "smart contracts". The presence of smart contracts enabled Ethereum to expand its functionalities over what is achieved by a distributed ledger such as Bitcoin; it made possible for the blockchain to function as a distributed computer.

The concept of smart contracts dates back to the 1990s, with early work by Szabo [18] and Miller [19], with a proposal for the algorithmic enforcement of agreements, but with no system proposed for implementation. A smart contract is defined by a collection of code (functions) and data (the contract's state). In Ethereum, the code is usually written in the Solidity programming language.

As with typical user's accounts, smart contracts are identified by a 42 character hexadecimal address. Smart contracts, as externally owned accounts, can also hold ETH (the Ethereum coin) and "tokens" (i.e., proxies for assets), and communicate with other accounts. Although there is a difference in the type of communication a contract can partake, contracts cannot initiate transactions; they can, however, respond to and send messages, to perform internal communications between accounts, though it must within a transaction started by an user.

Once a contract is created, it needs to be deployed into the blockchain, through a dedicated transaction (see Section 3.4.3).

#### 3.4.1.1 Chain computation

Within the context of smart contracts, one denotes as an "on-chain" process any computation that happens within a smart contract inside a blockchain, enjoying thus the security and trust of the cryptosystem. Its counterpart is an "off-chain" process, which is a computation being executed anywhere outside of the blockchain. On-chain programs in Ethereum are written in Solidity.

#### 3.4.1.2 Solidity

Solidity is a Turing-complete, object-oriented, high-level language for implementing smart contracts [20]. Influenced by C++, Python and JavaScript, Solidity was designed to target the Ethereum Virtual Machine (EVM), which is presented in detail in Section 3.4.2. Solidity is statically typed, supporting inheritance, libraries and complex user-defined data structures.

As presented in Section 3.4.1, a smart contract is composed of its persisting state variables and functions that can modify this state. Both are declared within a Solidity contract. A Solidty contract is thus divided in its variables, or "data", and its "functions".

Any contract data must be assigned to a location: "storage" or "memory". Storage data is kept in the persistent storage of the blockchain; updating or storing a new value is thus very costly, because the whole blockchain needs to be updated with this new value. On the other side, memory data only exists during the execution of a transaction, and thus are much cheaper to manipulate [21]. The storage data in a contract represents its state variables, which can be public, internal or private. Being internal or private only prevents other contracts from reading or updating the variable; the data is however still readable by onlookers. Public variables have the interesting feature of having their getter functions generated automatically by the compiler, which allows other contracts to read their values.

Functions can be external, public, internal or private. External and public functions are part of the contract interface, which means that they can be called from other contracts and via transaction. Public, unlike external functions, can be called internally, within the current contract, while internal and private ones are called within a transaction execution [22].

Solidity adds to this set of functions the concept of "view functions". These functions are a specific type of function, that have no "state mutability" permission; thus they do not modify the state variables. View functions can be executed from an Ethereum client locally, without the execution of transactions. An example of view function are getters [21].

In Solidity, "events" are used to facilitate the communication between a smart contract and off-chain applications. Events can be emitted within a function; they are used to communicate changes in the current state of the contract. Applications can subscribe and listen to these events through the Remote Procedure Call (RPC) interface of an Ethereum client [23].

#### 3.4.1.3 Ethereum Request for Comments (ERC)

Ethereum Improvement Proposals (EIPs) are standards for the Ethereum platform that address core protocol specifications, client APIs, and, more importantly for this thesis, contract standards [24]. There are different types of EIPs: Ethereum request for comments, or ERCs, are application-level standards and conventions, including contract standards. ERCs don't need to be adopted by all participants [25], though they are part of an effort to standardize some functionalities among smart contracts.

The two most important standards that are related to this thesis work are ERC20 and ERC721.

**ERC20** This is a standard for contracts to define and keep tract of fungible tokens [26], "fungible" meaning that all the tokens have the same value and are mutually interchangeable. ERC20 makes it possible for different currencies to live inside the Ethereum network; these are often refereed to as "tokens". ERC20 is important to allow tokens to be re-used by other applications, such as wallets and decentralized exchanges [27].

**ERC721** While ERC20 defines fungible tokens, ERC721 is a standard for non-fungible tokens [28], hence their nickname NFTs. "Non-fungible" means here that each token is unique. NFTs have different applications, from identifier-authentication tokens to real estate deeds to artwork.

### 3.4.2 Ethereum Virtual Machine (EVM)

The Ethereum virtual machine, or EVM, is the stack-based execution machine that runs bytecode instructions, known as EVM Opcodes, to update the Ethereum systems' internal state. Ethereum has one (and only one) *canonical* state, maintained by the different nodes composing the blockchain. The EVM defines what changes in this state by executing the transactions from block to block [16].

The EVM performs all computations in a data area called the "stack". All in-memory values are also stored in this stack. It has a maximum depth of 1024 elements and supports 256-bit-wide words [29].

The EVM is completely isolated from the network, and thus contracts have no access to the network, rendering them unable to read previous transaction data or access other network data, unless it is through "messages", internal calls between contracts. This restriction is necessary for safety reasons, to allow untrusted code to be executed on the Ethereum blockchain [29]. Another security measure adopted by the EVM is exception handling; upon detecting an error, or running out of gas, the machine immediately halts, reverts all changes to the previous state and returns an error to the user.

### 3.4.3 Ethereum transactions

Ethereum transactions refer to actions initiated by externally-owned accounts. These transactions are submitted via an Ethereum client, with the purpose of altering the Ethereum's internal state. A transaction is defined by the following fields [16]:

- the recipient, i.e., a target address, which can either be an externally owned account or a smart contract;
- a signature, generated via the sender's private key and used for signing the data of the transaction and validating the sender's identity;
- a value, i.e., the amount of ETH transferred from the sender to the recipient;

- data, which is an optional field to include arbitrary side data, including the code for transactions if need be;
- `gasLimit`, which is the maximum amount of gas usable for running the transaction, which will thus be cancelled and reverted if this limit is reached by the miner that executes it;
- `gasPrice`, i.e., the price of a unit of gas, in Wei (smallest denomination of Ether, 1 ETH $= 10^{18}$ Wei [30]) [3].

Depending on the purpose of the transaction, it can be classified as a "transfer", a "contract call" or a "contract deployment".

**transfer** These transactions send ETH between accounts, being either another user or a contract. If its target is a contract, it will call the contract's "fallback function" once completed.

**Contract call** These transactions target contracts, specifying one of the contract's functions, with the corresponding input data. Contract calls can only be initiated by users, but contracts can communicate between themselves via internal messages.

**Contract deployment** This special type of transaction is used to insert a smart contract into the blockchain. The transaction must contain the contracts' bytecode, generated from compiling the Solidity source code, as well as inputs for the contract's constructor function. It is customary for developers to generate the deployment transaction via a deployment script in JavaScript or TypeScript [31], through tools such as Hardhat [32] or Truffle (more about Truffle in Section 3.6.2).

### 3.4.4 Transaction fees

In order for transactions to be acknowledged by the network, certain fees need to be paid. Here, we present the rules for transaction fee calculation on the Ethereum network.

#### 3.4.4.1 Gas

Gas is the "fuel" that enables computations in the Ethereum network. The purpose of the inclusion of gas in this infrastructure is two-fold: first, preventing denial-of-service attacks by limiting how much computation can be done in a transaction; and second, rewarding miners for their work, since gas must be paid by the transaction issuer. Crypto wallets such as MetaMask [33] have functionalities to calculate the gas consumption of a transaction a user is trying to execute (more about MetaMask on Section 3.6.4). In practice, a transaction's gas cost is defined by the sum of the executed EVM Opcodes gas costs [34].

[3] The Ethereum Improvement Protocol 1559 changed the gas price into `maxPriorityFeePerGas` and `maxFeePerGas`, both are further explained in the next subsection.

During the forging of a transaction, the user needs to set a gas limit and a gas price. The gas limit is the maximum amount of gas an user is willing to consume in a transaction. A transaction will be executed until it is completed or it reaches the gas limit. If a transaction reaches the gas limit without finishing execution, the transaction will be reverted; the EVM will revert any changes caused by the transaction, and the miner will be rewarded a fee of *Gas limit* $\times$ *Gas price*. If a transaction finishes before reaching the gas limit, the sender will be charged a fee equivalent to the amount of gas used by the transaction, i.e., *Gas used* $\times$ *Gas price*.

Gas prices are part of the kind of auction process the miners perform when selecting transactions with higher gas prices, since the corresponding transactions are generally more profitable for them.

#### 3.4.4.2 EIP-1559

As we explain during our study in Chapter 5, gas prices in Ethereum skyrocketed in 2021, restraining the access to the chain. The rise in gas prices was a side-effect of Ethereum's growth in popularity and a consequence of the auction model adopted by this platform, known as first-price auction. We discuss more about auctions in Chapter 4, and the Ethereum management of transactions is actually one interesting application of this general field of economics.

Correspondingly, in August 2021, Ethereum activated the "London" hard fork, and, among the updates in this fork was the adoption of EIP-1559, which overhauled the transaction fee mechanism. Instead of a single gas price as before, users now have to list three separate values[4]:

1. a "Base Fee", which is determined by the network itself, depending on the current usage and which is subsequently burned, i.e., discarded;

2. a "Max Priority Fee", which is optional, determined by the user, and is paid directly to miners;

3. the "Max Fee Per Gas", which is the absolute maximum a user is willing to pay per unit of gas to get their transaction included in a block.

### 3.4.5 Ethereum Block

A block is a collection of transactions that have been executed by a miner. As of August 2022, the Ethereum blocks' content is made of [35] [36]:

- a timestamp, which is the time at which the block has been mined;

- the `blockNumber`, which is the total number of blocks in the blockchain;

[4] See `https://www.blocknative.com/blog/eip-1559-fees`

- the `baseFeePerGas`, which, according to EIP-1559, is the minimum fee per gas required in order for the transaction to be included in the block;
- the proof-of-work difficulty required to mine the block (see below);
- `mixHash`, a unique identifier for the block;
- `parentHash`, the Keccak-256 hash of the previous block's header;
- the transactions' The Keccak 256 hash of the root node of the trie constructed by each transaction of the block;
- `stateRoot`, the Keccak-256 hash of the root node of the entire state trie of the system (accounts' balances, contracts' storages, contracts' codes and account nonces) after the block's transactions have taken place;
- the nonce, which is a hash value that, together with `mixHash`, is the proof of the proof-of-work execution.

Blocks have a set limit of gas. Ultimately, miners can choose to construct a block however they want, but choosing transactions that fill up the block gas and with the higher gas price or priority fee will maximize the miner's reward from mining.

### 3.4.6 Ethereum PoW

As of August 2022, Ethereum's consensus algorithm is the proof-of-work (PoW) Ethash [37] algorithm. Through Ethash, in order to publish a block, miners need to find a certain nonce value, within a set target by the protocol. The mining process is, as a whole, a race between miners, with each participant trying to find the nonce value through a (costly) trial-and-error approach.

The general Ethash scheme can be described as follows.

1. A *seed* value can be calculated for each block, by scanning through all the block headers until the current block;
2. From this seed, a miner can compute a 16 MB random [5] *cache*, knowing that a 1-GB *dataset* can be generated from each cache value.
3. Mining consists then in trying to find a *nonce* number that, when hashed together with a randomly-selected slice of the dataset, produces a result below a desired a *target* value.

[5] Software-generated randomness is actually pseudo-randomness [38]

The *target* value is defined by the block's *difficulty*, being inversely proportional; a higher difficulty generates a lower target, and thus a smaller set of valid hashes. This *difficulty* parameter is used to keep block production constant and prevent the creation of uncle blocks. EIP-100 [39] defines how the difficulty is automatically incremented or decremented, taking in account the difference between the present block's timestamp and the previous block in the chain, as well as the presence of uncle blocks.

Miners who successfully create, or "mint", their block can reward themselves with 2 ETH in transactions similar to Bitcoin's coinbases, as well as the fees from users' transactions [6]. Of course, finding an acceptable nonce is very demanding in terms of computation time; in order to compete, miners need thus very high-performance, expensive equipment, and the energy consumption to take part on the PoW-consensus process is very high, which is nowadays largely criticized by the general public. In fact, maintaining Ethereum's network through proof-of-work requires an annual energy of 73.2 TWh, which is similar to the consumption of a country like Austria. Thus, the average miner will most likely lose a significant amount of money in order to mine a block, and so, it is in their best interest to mine a valid block, so the reward will cover their cost.

## 3.5 Scalability issues

The Ethereum system, in its current state, is harmed, both by its choices of consensus mechanism and its popularity. The causes of Ethereum's scalability problems related to proof-of-work are further explored in Chapter 5.

### 3.5.1 Layer 2 scaling solutions

"Layer 2" is a collective term for the solutions developed by the crypto-community to tackle Ethereum's scaling problems, while still taking advantage of the security and trust provided by the Ethereum platform. The general approach taken by layer 2 solutions is to execute batches of transactions outside of the Ethereum mainnet, and then to group the results into a single transaction to the mainnet Ethereum, thus reducing gas fees and waiting time. In this context, Ethereum mainnet is the "layer 1", while the outside of Ethereum platform in which transactions are executed is the layer 2.

There are many different types of layer 2 solutions that are built on top of Ethereum, for example side-chains, which are blockchains that run in parallel to the Ethereum network, or off-chain solutions, such as Rollups and ZK-rollups. In this document, we concern ourselves with the sidechain approach, specifically the Polygon proof-of-stake (PoS) chain.

[6] After the adoption of EIP-1559, part of these fees are burned instead of gifted to the miner

#### 3.5.1.1 Polygon PoS

Polygon is a technology platform that offers a myriad of layer 2-scaling solutions, mainly targeting the Ethereum platform. We focus on the Polygon PoS chain, a proof-of-stake blockchain that is compatible with the Ethereum virtual machine (EVM) and works as a sidechain for Ethereum.

Polygon PoS is an adapted implementation of the Plasma framework for the Ethereum blockchain. First introduced by Poon and Vitalik in the 2017 article "Plasma: Scalable Autonomous Smart Contracts" [40], Plasma chains are built on top of another blockchain (known as a "root chain" in this context). Plasma chains, or "child chains", connect to the root chain through smart contracts known as "bridges" that transfer data from one chain to the other. Bridge contracts are an interesting topic per se, but for this thesis, we are interested in Polygon PoS as a proof-of-stake blockchain.

Polygon PoS is a hybrid plasma and proof-of-stake platform. A Polygon node is equipped with a two-layer implementation: a validator layer, known as "Heimdall", and a block-producer layer, named "Bor". As their namesake suggest, the Bor layer will produce blocks while Heimdall validates them.

Heimdall uses the Cosmos SDK [41] (an open-source framework for building public proof-of-stake blockchains) and a forked version of the Tendermint Byzantine Fault Tolerant (BFT [42]) consensus engine [43], called Peppermint [44].

In a proof-of-stake consensus, such as the one adopted by Polygon PoS, users need to deposit a certain amount of coins in order to be able to participate in the block production. These coins are at "stake" for the purpose of ensuring the correct behavior of participants. Misbehavior can cause the staker to lost part or the entirety of their coins.

For Polygon PoS, in order to participate in the consensus mechanism, users need to stake some ETH in a specific smart contract in the Ethereum blockchain. Heimdall nodes monitor the staking balances in this Ethereum contract to continuously update the staking values of the "validators" (nodes participating in the validator layer).

The Bor implementation is a fork of Go-Ethereum [45], the official Ethereum client, with a few modifications to accommodate Polygon's consensus mechanism. Block producers are elected by the PoS stakers through a voting mechanism. A block producer will then be selected for a pre-determined interval of time to produce a block; if it fails to do so withing the time limit, its staked coins are "slashed", that is, the producer pays a pre-determined penalty cost, and a new producer is voted up.

Block producers will sign on the proposed blocks, continuously growing the blockchain. After a determined number of blocks on the Polygon chain, Heimdall will elect a "checkpoint proposer". A checkpoint is created by the proposer after validating the most recent blocks produced by the Bor layer, after the previous checkpoint. This checkpoint will contain a "RootHash", a Merkle hash of the Bor blocks. Once the proposal checkpoint is validated by at least 2/3 of the validators, the checkpoint is submitted to a checkpoint contract deployed on Ethereum's mainnet, and the rewards for participating in the consensus protocol are distributed on the Ethereum platform.

Since Bor is a fork of Go-Ethereum, it retains much of its characteristics, such as block data pattern, and even more importantly, the EVM. Thus Ethereum's smart contracts can be executed seamlessly inside the Polygon PoS chain.

### 3.5.2 Ethereum Merge

The Ethereum Merge update is Ethereum's response to its scalability problems. The Ethereum updates form a road-map of interconnected protocol updates, designed by the Ethereum Foundation, with the promise of making the Ethereum network more scalable, more secure and more sustainable. Among these updates, we highlight the "Merge" update, an update that will upgrade Ethereum's consensus protocol from proof-of-work (PoW) to proof-of-stake (PoS).

With the Merge, Ethereum will adopt a two-layer protocol similar to Polygon's PoS protocol. The current Ethereum implementation will become the "execution layer" that will handle transactions and execution, while the "Beacon chain" will become the "consensus layer", introducing proof-of-stake as the new consensus mechanism of the blockchain. These two chains will "merge" into a new improved Ethereum network. The Beacon chain is a proof-of-stake chain that has existed in parallel to Ethereum's mainnet since December 1st 2020. It was subjected to intensive tests, until the Ethereum Foundation judged it was ready to integrate Ethereum's mainnet. Figure 3.5 represents the merge between Ethereum's mainnet and the beacon chain.

The Merge occurred in September 15, 2022 [6], at the time we were closing this thesis document. Thus, during our research, it was still too early to assess how the update affected Ethereum ecosystem. Though, before the Merge, Ethereum updated their testnets with the Merge for testing purposes. We report on the opportunity we had to experiment with those in Chapter 5.

#### 3.5.2.1 Merge PoS protocol

In order to become a "validator" on the Beacon chain, a user needs to deposit 32 ETH as their "stake", as well as run 3 different clients: an execution client, a consensus client and a validator.

Once active in the consensus mechanism, validators will receive new block proposals from their validator peers. A validator needs to check the validity of the block data before voting on the approval or rejection of the block.

The block time in PoS is no longer dependent on PoW; the time is fixed and divided into slots (12 seconds) and epochs (32 slots). During a slot, a validator is randomly selected to be the block proposer. This validator will create a block and broadcast it to its peers, as a miner would do in Ethereum's PoW. To validate and vote on the block creation, a committee of validators is randomly chosen per slot as well.

Finality in Ethereum's PoS is achieved using "checkpoint" blocks. The first block in each epoch is a checkpoint. Validators vote for pairs of checkpoints that are deemed valid, the previous epoch checkpoint and the new recently created. When the pair of checkpoints acquires votes from 2/3 of

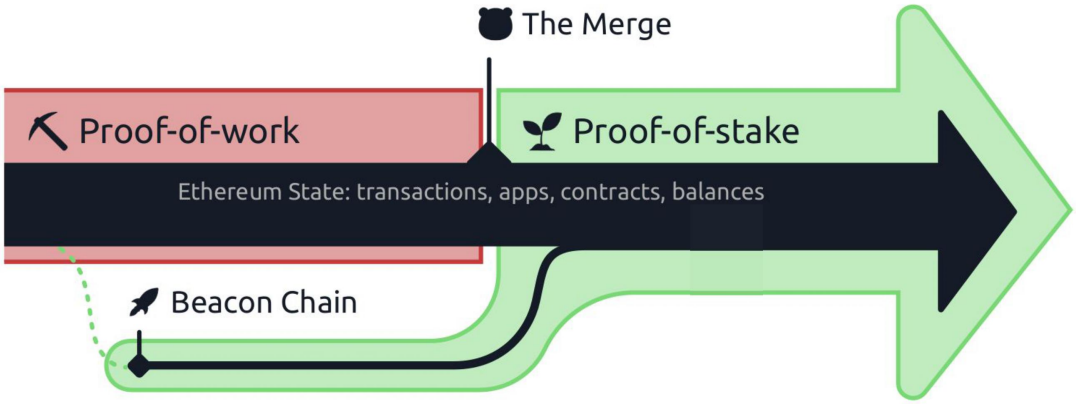


Figure 3.5: Ethereum Merge representation [46]

the validators, they are upgraded. The previous checkpoint sees its status change from "justified" to "finalized", and the current one is upgraded to "justified". Then a new epoch can begin.

Validators are rewarded for participating in the PoS mechanism, with different rewards for proposing blocks, voting and even whistle-blowing (attesting another user's dishonest behavior). The block producer will be rewarded with the transaction fees, the same as miners are in PoW, thus the Merge don't affect transactions' gas costs. Dishonest behavior from validators is penalized with "slashing" (destruction) of their staked coins.

Staking doesn't require an initial investment in powerful hardware, and it doesn't consume as much energy as PoW, since all the computation for finding a nonce value isn't present in the mechanism. The Ethereum foundation claims that "*The Merge will reduce Ethereum's energy consumption by 99.95%*" [46].

Due to not requiring expensive hardware and high energy consumption, theoretically, PoS is more democratized than poof-of-work. Requiring only an initial deposit of 32 ETH (though there are also different tiers, with smaller deposits), PoS is deemed more accessible than PoW, which the Ethereum Foundation hopes will help increase the decentralization over the network.

### 3.5.3 Tezos

Tezos is a third-generation blockchain that intends to address the cost, energy and scalability issues that affected earlier chains adopting the proof-of-work approach. Funded in 2017 by the largest Initial Coin Offering at the time [47], Tezos is characterized by its proof-of-stake consensus mechanism, its self-amending properties, and its smart contracts.

#### 3.5.3.1 Tezos Proof-of-Stake

Tezos' consensus mechanism is quite particular for a proof-of-stake mechanism. In Tezos' terminology, miners are known as "bakers", and mining is "baking" a block. In order to be able to participate in the proof-of-stake mechanism, bakers need to stake a minimum of 6,000 Tez (XTZ) [48], a sum known as a "roll". Tezos' proof-of-stake is particular, because it enables coin holders to "delegate" their baking power to another baker; the ownership of the coins isn't transferred to the baker, and both parts share the baking revenue. This delegation property has given the consensus mechanism the title of "delegated proof-of-stake". Delegation is interesting, because it enables coin owners to indirectly participate in the consensus mechanism without having to invest in expensive equipment, by abdicating their baking power to a baker they judge trustworthy.

Tezos' proof-of-stake mechanism is named "Tenderbake"; it is a Byzantine Fault Tolerant-style consensus algorithm, in which, differently from Bitcoin's and Ethereum's PoW, bakers don't produce a block directly. In Tenderbake, a baker will first propose a block, while others will accept or reject the proposal [49].

Tenderbake is executed for each new block. First, a group of bakers are selected at random, based on their stake, to form a committee, whose members are called "validators". The baking process will continue in rounds; each round is an attempt from the validators to reach a consensus on which block to publish. A round consists of the following steps:

1. the round's designated validator injects a candidate block;

2. this block diffuses over the network nodes to the other validators;

3. the validators vote either to accept or reject the block.

Rewards from baking are divided in different parts. Endorsing rewards are shared between all validators, while the block's proposer is rewarded with a baking reward plus the fees associated to the transactions included in the block. If a baker misbehaves, their staked coins are "slashed": for example, if double baking or double endorsing, which happens when a baker proposes or endorses two different blocks at the same time, occurs, the baker's stake is slashed by 640 Tez. Bakers that show evidence of double baking are rewarded with half of the slashed amount.

According to the Tezos' Foundation, Tezos has an yearly energy consumption of 60 MWh, or 0.00006TWh [50], i.e., 0.000082% of Ethereum PoW's 73.2 TWh.

#### 3.5.3.2 Tezos governance and self-amendments

In the Tezos system, stakeholders can participate in governing the protocol. Community members can propose updates for the protocol [51]; these protocols are voted for or against through an on-chain process. Once a proposal is accepted by the stakeholders, there's a two-week period for the community of bakers and developers to update their software. This is part of the self-amendment property of Tezos, which allows the chain to upgrade itself without the need for hard forks.

#### 3.5.3.3 Tezos smart contracts

Tezos' smart contracts are heavily based on Ethereum's, but with a greater concern about formal verification. Formal verification is the process of applying mathematical formal definitions to testify that a program conforms to it's specification [52]. The proposal is to avoid costly bugs, by verifying contracts before deploying them in the blockchain. Smart contracts in the Tezos blockchain are written in Michelson, a low-level stack based language. For high-level languages, Tezos proposes a handful, with SmartPy and LIGO being the most adopted ones by the community of developers [53]. In this document, we just focus on SmartPy.

**SmartPy** SmartPy, a Python library, functions as a complete system for the development and test of smart contracts. SmartPy is characterized by meta-programming; the Python code, when executed, is used to write the final Michelson contract. Yet, Tezos smart contract operations are quite similar to Ethereum's, with users having to forge transactions to activate a contract operation. A terminology difference is that "public" or "external" functions in Tezos are known as "entrypoints"[54].

Tezos uses a different gas system than Ethereum. A user is charged for each transaction in two different ways: a fee, which is credited to the block baker, and a certain amount of "burned" coins, i.e., sent to an unreachable account. For having a transaction performed, a user needs to provide a fee (in XTZ) and a gas limit; the transaction will then compete with other transactions to be added to a block, taking into account two limitations, namely the hard block gas limit (10,400,000 gas) and the hard operation gas limit (1,040,000 gas). Bakers then choose transactions, assuming that gas fits the block and fees respect a minimum.

Whenever the size of the blockchain storage increases due to a transaction, the issuer must pay a "burn", in XTZ. This happens when a contract storage increases (storage burn) or when a new contract is put on the chain (allocation burn).

### 3.6 Tools

Understanding the behavior of a distributed, asynchronous, complex system such as a blockchain is difficult. Luckily, some tools help in that regard; the ones used in this thesis are presented here.

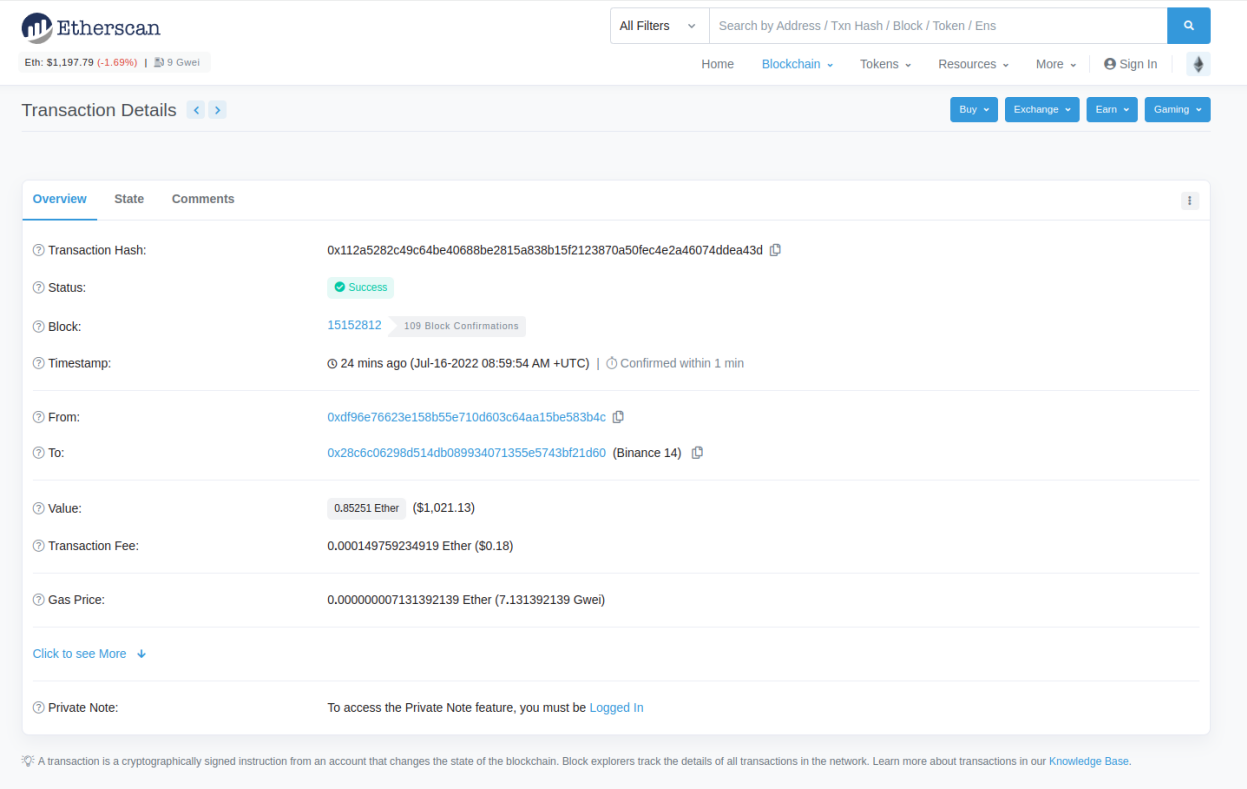


Figure 3.6: Example of a transaction from Etherscan(`etherscan.io`) the main Ethereum block explorer

### 3.6.1 Block explorers

Block explorers, as their name implies, are tools for exploring the blockchain data. Due to their continuous growth, blockchains are considered challenging to index, where "indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database" [55].

Block explorers retrieve data from blockchains, and index it appropriately, making the blockchain data easily available for users to query transactions, wallets or blocks, and also providing not only the chain's data, but also relevant insights, such as charts and statistics, that were particularly useful for this thesis. In Figure 3.6, one can see the interface of Ethereum's main block explorer, Etherscan (`etherscan.io`), displaying information from a transaction; relevant information, such as "Transaction Hash", "Block number", sender (From) and receiver (To), "Transaction Fee" and "Gas Price" are all available.

### 3.6.2 Truffle

Truffle [56] is a development environment for smart contracts. Developed by ConsenSys (`consensys.net`), Truffle offers a wide variety of functionalities for contract debugging, compilation and deployment. In addition to network management, it enables users to switch between mainnet and different testnets in a seamless manner. Truffle was originally conceived to attend to blockchains using the Ethereum Virtual Machine, although support for other blockchains has been added as time went [57].

### 3.6.3 Infura

Infura [58] is an Ethereum API provider that allows communication with blockchains without the need of setting up an Ethereum client, which can be a time consuming operation and is heavy on storage.

Infura provides access over HTTPS and WebSockets to the Ethereum network [59]. Offering different tiers for different prices, each with different levels of support and numbers of daily requests, the core service is however free, with a total of 100,000 requests per day, which was plenty for our research work. Infura's main target is the Ethereum blockchain, though support for a few other chains is also available.

### 3.6.4 MetaMask

MetaMask [33] is one of the most popular cryptocurrency wallets for ETH and ETH-based tokens (ERC20 and ERC721) [60]. MetaMask stores users' private keys, through which they can generate accounts that can store cryptocurrencies. MetaMask's main utility is managing users' private keys in order to sign transactions to communicate with blockchains.

On top of key management, MetaMask also offers functionalities such as coin exchange and the ability to connect to dApps (decentralized applications). MetaMast is characterized by being a so-called "hot wallet", which means that it needs to be connected to the internet to be usable, while "cold wallets" can keep data offline. In the case of MetaMask, being a hot wallet translates to it being a browser extension (though a mobile app version is also available). Image 3.7 shows an example of its Google Chrome MetaMask pop-up interface. MetaMask stores a crypted version of a users' keys in the browser's storage, decrypting them for usage once the users logs into the wallet with their passwords [61].

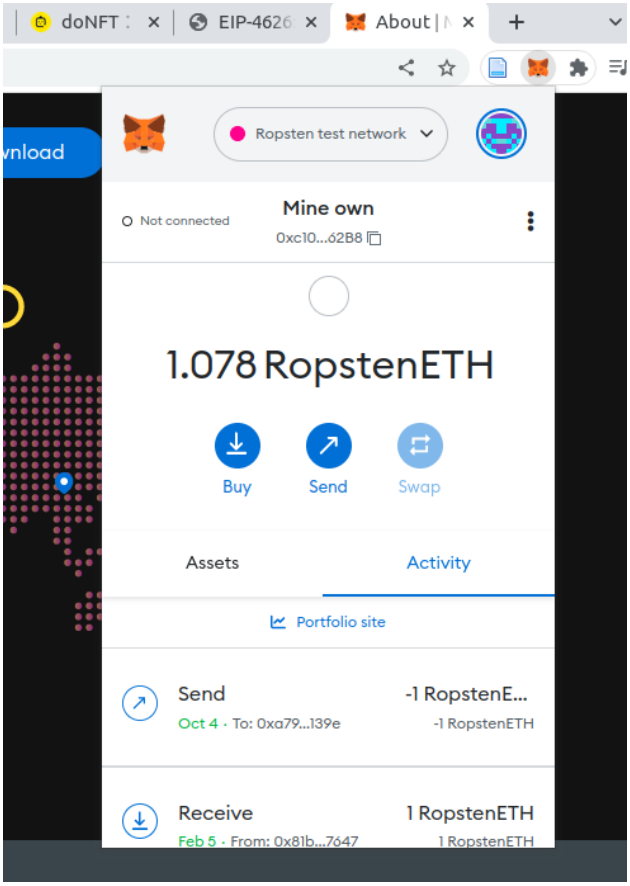


Figure 3.7: Metamask interface, showing the ETH balance for the account named “Mine own” on the Ropsten testnet

## CHAPTER 4

# VCG FOR SEARCH AUCTIONS USE CASE

*Ce chapitre présente notre principal cas d'utilisation pour notre étude de contrats intelligents, l'enchère de Vickrey-Clarke-Groves pour la recherche sponsorisée, ou VCG pour la recherche. L'objectif de ce chapitre est de souligner l'importance et l'applicabilité de VCG pour la recherche en tant que cas d'utilisation en présentant ses principales propriétés, qui revêtent une importance pour la théorie des mécanismes.*

*La section 4.1 commence en fournissant un aperçu général des enchères et des concepts de la théorie des enchères nécessaires pour analyser et caractériser l'enchère VCG. De plus, cette section présente le mécanisme général de Vickrey-Clarke-Groves, ainsi que ses propriétés. La section se conclut par la présentation d'une modélisation du mécanisme VCG général pour l'assistant de preuves Coq.*

*La section 4.2 présente enfin l'enchère de Vickrey-Clarke-Groves pour la recherche sponsorisée. Tout d'abord sont présentées les enchères pour la recherche sponsorisée, ainsi que leur objectif et les principales définitions associées. Une fois le contexte de la recherche sponsorisée établi, nous exposons l'algorithme de l'enchère VCG pour la recherche sponsorisée, notre principal cas d'utilisation. La section se termine par un aperçu des cas d'utilisation de VCG pour la recherche dans l'industrie.*

This chapter presents our main use-case for our smart contract study, the Vickrey–Clarke–Groves auction for sponsored search, or VCG for search. The aim of this chapter is to underscore the significance and applicability of VCG for search as a use case by presenting its key properties, that are of importance for the field of mechanism design.

Section 4.1 begins by providing a general overview of auctions and auction theory concepts, which are required to analyze and characterize the VCG acution. Furthermore, this section showcases the Vickrey–Clarke–Groves general mechanism, together with its properties. The section concludes with a modelization of the general VCG mechanism with the proof assistant Coq.

Section 4.2 finally presents the Vickrey–Clarke–Groves auction for sponsored search, beginning with a presentation of sponsored search auctions, their purpose and key definitions. With the setting of sponsored search in place, we present the algorithm for the VCG auction for sponsored search, our main use-case. The section ends with an overview of cases of VCG for search in the industry.

## 4.1 Auctions

Auctions have been considered since antiquity as a flexible way to sell a wide variety of goods. There are reports of auctions being used as early as 500 BC [62]. In recent years, with the advent of the internet, auctions have taken a very central role, for instance in e-commerce [63]. In particular, auctions for the sale of sponsored links, as they occur when any search query is performed on Google, for instance (see Figure 4.1), are, alone, responsible for the movement of hundreds of billions of dollars per year [64].

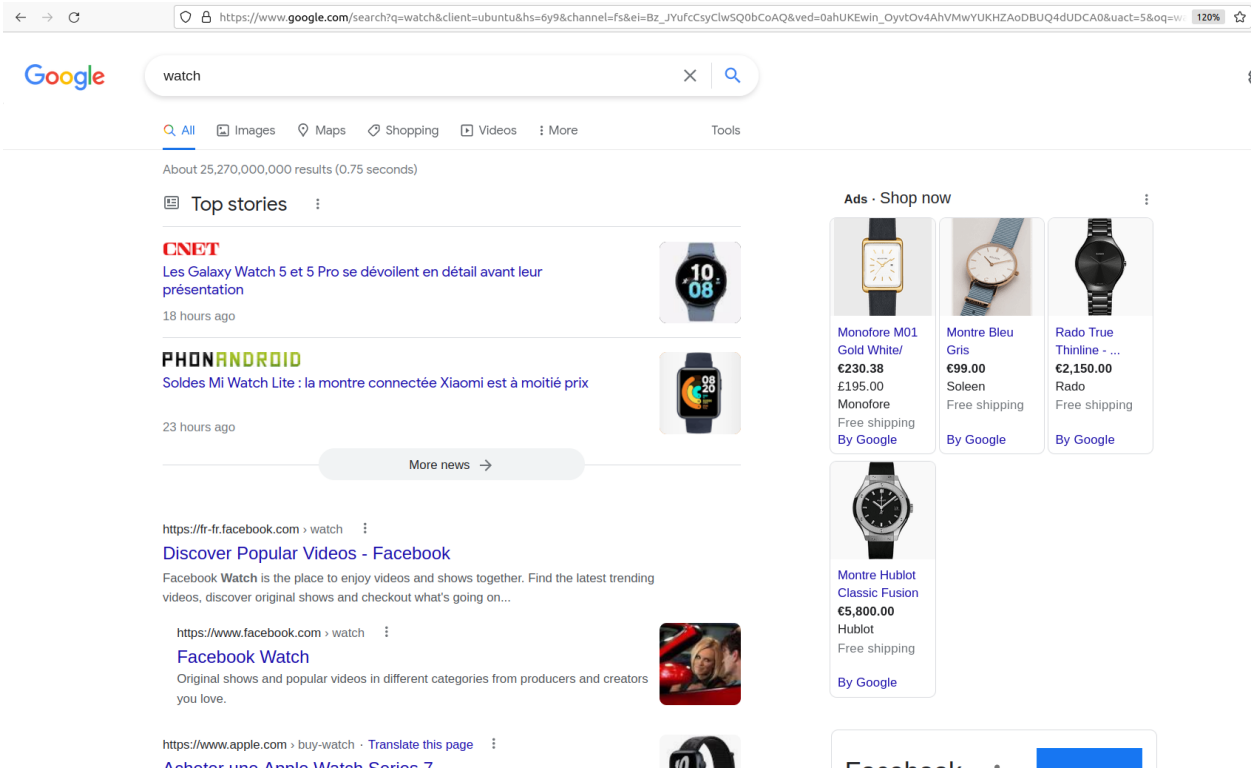


Figure 4.1: Example of a Google query result page, with ads displayed on the right. The page areas for ads have been auctioned to advertisers; their bids are linked to how potentially interested the expected viewer is for the products they help sell.

Readers unfamiliar with auction theory may think that auctions are restricted to what is commonly known as "English" auctions; these are ascending price auctions, such as one might come across in auction houses for the sale of art and antiques, or in internet marketplaces such as eBay or OpenSea. But auctions are in fact much more flexible than that.

At the most abstract level, an auction is a process to sell a good to potential buyers when the seller is unsure of the object's value or intends to maximize his or her revenue. In an auction,

the seller or a chosen intermediary, jointly known as the "auctioneer", offers one of more objects to potential sellers. The auctioneer does not know how much the buyers value the object being sold; otherwise they would simply ask them to pay for the corresponding price. The fundamental uncertainty regarding the values of objects that face both sellers and buyers is an inherent feature of auctions [65].

In an auction, the buyers communicate their evaluation of the sold objects to the auctioneer in the form of "bids"; bids are offers of a price. The value of a bid will depend on the buyer's strategy, and it usually reflects the buyer's evaluation or, at least, the one he or she is willing to provide to the auctioneer. In the sequel of this work, we assume that, once all bidders have cast their bids, the selected type of auction will decide on the winner(s), allocate the object(s) to them, and charge them a price related to their bid(s).

### 4.1.1 Principles

Auctions are very flexible processes and can take many shapes; they can be an eBay auction to sell a rare Game Boy model, a contractor's auction for a public construction project or a candle auction to sell fresh-caught fish. Auctions can even be automated and appear to happen instantly, such as the ones used to sell advertisements every time someone queries a search engine.

In this section, we seek to identify some important features that auctions have in common; this will help in our understanding of the VCG for search auction and of its properties, one of the main focal points of this thesis.

The premise of an auction is the offer of a number of goods, or objects, by an auctioneer. Interested buyers offer prices to try to win the objects. As seen before, these offers are known as bids; hence, whoever casts a bid is know as a bidder. Once the bidding phase is over, the auctioneer decides on an outcome, consisting on (1) assigning the auctioned object(s) to one (or more) of the bidders and (2) defining the prices these winning bidders need to pay for the objects. In this framework, an auction can be characterized by the two sets of rules for bidding and for computing the outcome.

The rules for bidding describe how bidders communicate their bids to the auctioneer. There may be rules related to the duration of the bidding phase, for example via a specified deadline, as is the case with eBay auctions. Yet, the duration can also depend on something less predictable; in candle auctions, for instance, bids are accepted until the light of a designated (physical) candle goes out. Another important aspect addressed in the bidding rules is how the bids are transmitted; for instance, they can be publicly announced, as occurs in the so-called "open" auctions. An eBay auction is characterized by this open-bidding style; the highest current bid is always listed. The antithesis of open auctions are "sealed-bids" auctions. In this case, bidders communicate their bids to the auctioneer in secret, for example in a sealed envelope. The idea is to make sure, for privacy reasons (something we address in Chapter 6), that no bidder knows how much the other auction participants have bid.

The second set of rules deals with the computation of the auction outcome. This outcome pro-

cess can again be divided in two parts: assignment and pricing. Assignment allocates the auctioned objects to bidders, based on the values of their bids. For a single-item auction, it is customary for the highest bidder to be awarded the object, while for multi-item auctions, the assignment process varies. We address in more details multi-item assignments when discussing VCG for search auctions (see Section 4.1.5).

Concerning the pricing decisions, there are multiple approaches, and the pricing method is a tool to influence how a bidder will choose their bid in the bidding phase. The straightforward pricing solution of charging the winner an amount equal to the value of their bid is known as "first-price" auction. first-price auctions, though easy to understand, are difficult for the bidders to reason about when putting their bids, as we see in Section 4.1.3.1. A popular alternative to first-price auctions is what is known as "second-price" auction. In this case, the highest bidder pays the price of the second-highest bid. If this can appear bewildering to the reader unfamiliar with auction theory, second-price auctions have some interesting properties that help bidders when deciding what bids to put; we go over these properties in Section 4.1.3.2

### 4.1.2 Game theory notions

In order to better understand some of the properties of auctions, a few notions on auction theory and game theory are required; here we intend to present these relevant notions. We first introduce a model of bidders' behavior in order to reason over their strategies and choices.

Bidders are supposed to be always interested in the objects on sale, and each bidder $i$ has an evaluation of how much these said objects are worth. This value, that we reference by $v_i$, represents the bidder $i$'s maximum willingness-to-pay for the object being offered (for simplicity, we assume here that only one object is auctioned). A lot can be inferred by how one decides to model this value; for now, we consider that each bidder's evaluation is private and not influenced by others.

By taking part in an auction, each participant is assigned an utility function $u_i$ (which we note $u$ when no confusion can occur). Utilities are a formal abstraction of the "enjoyment" one gets from participating in the auction process. For our model, we adopt the so-called "quasilinear utility model": if a bidder $i$ loses an auction, their utility is $u_i = 0$; if the bidder wins at a price $p$, then their utility is $u_i = v_i - p$. Formally, one has:

$$u_i = \begin{cases} 0, & \text{if } i \text{ loses;} \\ v_i - p, & \text{otherwise.} \end{cases}$$

We assume that bidders have the goal of maximizing their utility. For our model, it would mean obtaining the object with the lowest price possible. In our studies, we consider the bidders' strategy as trying to reach this goal.

In game theory, much can be observed from the analysis of strategies; here we introduce the notion of "dominant" strategies. A dominant strategy for a player is one that, when compared to

other strategies, gives the best utility, no matter the strategy chosen by other players. There are some special auctions in which "truth telling", meaning bidding one's true value $v_i$ directly, is the dominant strategy. An auction with this characteristic is known as "truthful" or "dominant-strategy incentive compatible".

### 4.1.3 Types of auctions

In this section, we explore different types of auctions, and analyse them with the concepts presented above.

#### 4.1.3.1 First-price auctions

The first-price auction is, as stated above, a type of auction in which the winner is charged the totality of their bid. An example of such an auction is the English auction, which is characterized by incremental open bids and first-pricing. This is the classic type of auction we find in auction houses. Bidders bid publicly, with ever-increasing bids, until no one is willing to raise the last-offered price. The pricing rule is first-price, so the highest bidder has to pay the value of their bid for the object. An English auction is the most familiar form of auction, and likely what comes to mind to the general public when referencing auctions. The term auction most likely has its terminological roots in this format, coming from *auctus*, "increasing" in Latin.

There is no dominant strategy for English auctions; if a participant bids their true value and wins, their utility will be ... 0. The bidder needs to come up with a bid that is below their true evaluation, but is still big enough to ensure their win, while not knowing how other bidders will bid, to get a strictly positive utility. Therefore, it can be difficult for bidders to decide how to bid.

#### 4.1.3.2 Second-price and Vickrey auctions

In second-price auctions, the winner is still the highest bidder, but pays the price of the second-highest bidder. At first glance, it may appear backwards, but second-price auctions have interesting properties. Second-price auctions are truthful; therefore, the dominant strategy for any bidder is to set their bid equal to their private evaluation. By the definition of dominant strategy, this strategy guarantees to maximize the player's utility, no matter what other bidders do.

A prominent example of second-price auctions is the Vickrey auction, first described by the Columbia University professor William Vickrey in 1961 [66]. The Vickrey auction is a sealed-bid auction; the bidders transmit their bids in secret to the auctioneer, which then awards the object to the highest bidder, charging the second-highest bid as price.

To better understand how this auction works, let us imagine a user $i$ with private value $v_i$. We can show that $i$'s utility is maximized by bidding $b_i = v_i$.

Let us thus consider $B$ equal to the highest bid among the other bids $b_j$: $B = \max_{j \neq i} b_i$. There are only two possible outcomes (for simplicity, we reason here while assuming that all bids are distinct):

- either $b_i < B$, in which case the bidder $i$ loses and has utility $0$;
- or $b_i \geq B$ , in which case $i$ wins, pays the second highest bid $B$ and has utility $v_i - B$.

Let's imagine the case where $i$ doesn't bid its true value, there are two cases.

- First, $i$ under-bids, i.e., $b_i < v_i$. In the case that $b_i < B$, $i$ has utility $0$, although in this case, there is still the possibility of having $B < v_i$, and, thus, the bidder harmed their chances by underbidding. If $b_i \geq B$, $i$ wins, with utility still being $v_i - B$, with no gain to their utility by underbidding.
- Second, $i$ over-bids, i.e., $b_i > v_i$. In the case that $b_i < B$, $i$ has utility $0$. If $b_i \geq B$, $i$ wins, with utility $v_i - B$. In this last sub-case, there are are two possibilities: (1) $v_i \geq B$, which would lead to a positive utility, and (2) $B > v_i$, which would lead to a negative utility, an unfortunate result.

Now, assume $i$ bids truthfully: $b_i = v_i$. If $v_i < B$, then the bidder $i$ loses and their highest utility is $0$. Otherwise, if $v_i \geq B$, $i$ wins and their utility is the positive value $v_i - B$ [67].

In a Vickrey auction, as long as the bidders do not deviate from their true value, they are guaranteed non-negative utility. Losers get $0$, and the winners gets $b_i - B$, where $B$ is the second-highest bid; if $b_i = v_i$ (truthful bidding), positive utility is guaranteed, since $u_i = v_i - B$ and $b_i \geq B$ (condition for $i$ to win). This property of non-negative utility, insured here by truthful bidding, is known in economics as the "individually rational" behavior of participants.

### 4.1.4 Mechanism design

Mechanism design is the science of rule making, a field of economics and game theory that reasons over games with incomplete information to generate a desirable outcome (as with auctions, an outcome is here an allocation rule and a payment rule).

The setup for a general mechanism is the following: a seller with one object to sell to $N$ potential buyers, each with a private evaluation $v_i$ for the object. A mechanism is then characterized by 3 components:

1. sets $M_i$ of possible messages $m_{ji}$ from each buyer $i$ to the seller;
2. an allocation rule $\pi : M \to \Delta$, with $\Delta$ being the set of possible outcomes, i.e., the probabilities of winning the object ($M$ is the Cartesian product of all $M_i$);
3. payment rule $\mu : M \to \mathbb{R}^N$.

The allocation rule, a function of $N$ messages, determines the probability $\pi(m)_i$ that $i$ gets the object. The payment rule, in turn, computes the expected payments $\mu(m)$ for all participants [65].

By now, the more attentive reader will have realized the similarity between the notion of a mechanism and auctions; indeed, auctions are instances of mechanisms. For example, the Vickrey auction is a mechanism instantiated as follows.

1. Messages are bids, and $m$ can be assumed to be the values of participants $(v_1, ..., v_N)$, due to truthful bidding.

2. The allocation rule is such that $\pi(m)_i$ is 1, if $m_i > \max_{j \neq i} m_j$, and 0 otherwise.

3. The payment rule is such that $\mu(m)_i = \max_{j \neq i} m_j$, if $m_i > \max_{j \neq i} m_j$, and otherwise $0$;

A mechanism can be designed to enforce some interesting properties on the game. For instance, "Incentive Compatibility"(IC), denotes that participants can obtain the best outcome for themselves by acting in accordance to their true values [68]. IC is a characteristic of the so-called "direct" mechanisms, mechanisms in which the set of messages is equal to the set of private values of participants; if, for instance, $m_i = v_i$, then this is the same as truthful bidding for auctions. Another relevant property of direct mechanisms is "Individual Rationality" (IR); an IR mechanism guarantees the participants non-negative utility, i.e., $u_i \geq 0$. Finally, the last relevant property of mechanisms is "welfare maximization". Social welfare can be defined as a the sum of all winners' values; a mechanism is then "welfare-maximizing", or efficient, if it maximizes the group's globally stated interest in the goods. If a mechanism is welfare-maximizing, it means that the (single) object being sold is given to the buyer who values it the most. The welfare-maximizing allocation $\pi_\Delta$ has more total value to the bidders than any other from the group $\Delta$; no other allocation could make anyone better off without making someone else worse off.

### 4.1.5 Vickrey-Clarke-Groves mechanism

The Vickrey–Clarke–Groves mechanism, is named after William Vickrey, Edward H. Clarke, and Theodore Groves. For Vickrey, this is due to his contribution with the Vickrey auction, while Edward H. Clarke[69] and Theodore Groves[70] are both credited for their study on public choice problems, in which agents decide whether to undertake a public project – e.g., the construction of a bridge or highway [71].

The Vickrey–Clarke–Groves (VCG) mechanism set up is as follows:

1. $N$ strategic participants;

2. a finite set $\Omega$ of all the possible distributions of the sold objects between the participants;

3. a private valuation $v_i(\omega)$ for each outcome $\omega \in \Omega$ and each agent $i$.

VCG is a generalization of the Vickrey auction for different outcomes and multiple objects. But the notion of charging the winner the highest bid is extrapolated into the concept of "externality". Externality is the loss in social welfare a bidder imposes by his or her participation in the auction on

the rest of the participants. In the Vickrey auction's case, when one charges the second highest bid to the single winner, what is actually charged is the externality. For example, let assume a Vickrey auction with the sorted bids $v_1 > v_2... > v_N$. Bidder $1$ wins the auction, and creates a total welfare of $v_1$, while the rest of the bidders, $2, 3, ...N$, have social welfare 0, as a group. Now let's imagine the same auction without the first bidder; the new order of bids would be $v_2 > v_3... > v_N$, and the new winner would be bidder $2$ and the welfare would be $v_2$. Thus, in this example, we can attest that the externality imposed by bidder $1$ is $v_2 - 0 = v_2$, the difference between the social welfares of the others without and with bidder $1$, which is the same as the Vickrey payment for bidder $1$. The Vickrey–Clarke–Groves mechanism has the same payment rule as the one introduced here, but is generalized for more than single-object auctions.

The VCG mechanism is defined by the following three steps:

1. accept a bid $b_i(\omega)$ from each buyer $i$ for each outcome $\omega \in \Omega$;

2. compute an optimal outcome $\omega^*$ that maximizes the reported[1] social welfare $\sum_{i=1}^{N} b_i(\omega)$ over all $\omega \in \Omega$;

3. charge each bidder $i$ their externality, meaning the social welfare loss caused to the other bidders by $i$'s presence: $p_i = (max_{\omega \in \Omega} \sum_{j \neq i} b_j(\omega)) - \sum_{j \neq i} b_j(\omega^*)$.

The VCG mechanism is of great significance for the study of mechanism design; it is the golden standard for a mechanism, due to its properties of being truthful, individually rational, and welfare-maximizing.

### 4.1.6 VCG modelization and properties in Coq

Even though the previous definition of the VCG mechanism is somewhat precise, we believe a more formal approach to mechanism design in general is warranted, in particular given the increasing importance of mechanisms in the society at large. This is even more the case in the context of smart contracts, since their code cannot be updated once deployed; getting the algorithm right the first time is the rule of the game here. The formal verification of the correctness of algorithms is the most reliable way to validate their desired behavior.

We propose here a proof-of-concept application of this philosophy by introducing a Coq specification of the VCG mechanism described above. The Coq proof assistant, a formal proof-management system, provides a formal language to write mathematical definitions, algorithms and theorems about these algorithms. It also provides a user environment for the semi-interactive development of machine-checked proofs.

We thus describe below our modelization of the VCG mechanism presented above.

[1] Note that the bids can be seen here as proxies of the actual values.

**Definition** For functional programmers, a mechanism in Coq is simply a higher-order function or module, here VCG. The VCG mechanism, in Listing 4.1, is abstracted over the type $O$ of possible auction outcomes, a particular instance $o0$ (to ensure non-emptiness) and an agent $i$. All the types used here are finite types, e.g. $'I\_n$ for $A$, i.e., the sets of bounded natural numbers, here in [0, $n$[.

Here, any agent, among $n$, is defined by its $bidding$, a finite function that values any possible outcome in the Coq domain $nat$ of natural numbers (for simplicity, we do not use reals here). The VCG mechanism, given its last parameter, a tuple bs of biddings, must compute an optimal outcome oStar that maximizes the total bidSum $o$ of bids. In a truthful mechanism, where the bids of agents and their values coincide, this outcome maximizes the global good, or social welfare. For agent $i$, the price she accordingly has to pay to win whatever is in oStar for her is a penality induced by the impact on the global good of her presence in the bidding process (welfare_with_i) compared to when she is not (welfare_without_i, which would have yielded a possibly different optimal outcome).

```
1  Variable (O : finType) (o0 : O) (i : A).
2
3  Definition bidding := {ffun O → nat}.
4  Definition biddings := n.-tuple bidding.
5
6  Variable (bs : biddings).
7  Local Notation "'bidding_ j" := (tnth bs j) (at level 10).
8
9  Implicit Types (o : O) (bs : biddings).
10
11 Definition bidSum o := \sum_(j < n) 'bidding_j o.
12 Definition bidSum_i o := \sum_(j < n | j != i) 'bidding_j o.
13
14 Definition oStar := [arg max_(o > o0) (bidSum o)].
15
16 Definition welfare_with_i := bidSum_i oStar.
17 Definition welfare_without_i := \max_o bidSum_i o.
18
19 Definition price := welfare_without_i - welfare_with_i.
```

Listing 4.1: VCG mechanism in Coq

**Properties** We formally prove that the VCG mechanism enjoys useful properties such as No Positive Transfer (all prices are positive, and thus the auctioneer does not have to pay bidders), Individual Rationality (for any agent, the price is less than the value of the outcome for him) and the most important one, Truthfulness (see Listing 4.2).

The VCG mechanism assumes the existence, for any agent $i$, of a valuation value i that he

assigns to any outcome in $O$. The utility of the bidding result for $i$, among $n$ agents bidding $\mathrm{bs}$, is then the difference between whatever the perceived value is and the price paid (note the three explicit arguments to the mechanism functions $\mathrm{oStar}$ and $\mathrm{price}$). The truthfulness property that $\mathrm{truthful}$ expresses is key. It states, that all things being equal, as stated by $\mathrm{differ\_only\_i}$, the only way $i$ can increase their utility is by bidding, for any outcome $o$, what is for him its true value in o.

The complete proof of the VCG mechanism's truthfulness can be found in the repository [72]. Note also that this line of work has been since largely developed further to address other mechanisms, via a new $\mathrm{mech.v}$ Coq library (see [73], [74] and [72]).

```
1  Variable (value : bidding 0).
2
3  Definition utility bs := value (oStar 0 o0 bs) - (price 0 o0 i bs).
4
5  Definition differ_only_i bs' := forall j, j != i → tnth bs' j = 'bidding_j.
6
7  Theorem truthful bs' :
8    'bidding_i =1 value →
9      differ_only_i bs' →
10       utility bs' <= utility bs.
```

Listing 4.2: Truthfulness definition

## 4.2 VCG for sponsored search

The Vickrey–Clarke–Groves mechanism, though of great importance for the field of mechanism design, has been severely underutilized in marketplaces [75] [76] [77] until Facebook adopted an instance of the mechanism for the sale of advertisement links in the late 2000s, known as VCG for sponsored search.

### 4.2.1 Sponsored search

Today, the most common type of auction are sponsored search (SS) auctions. This type of auction is designed with the purpose of selling sponsored links to advertisers. These links are advertisement printed simultaneously with the results of a search engine. In such an auction, the keywords that define the search request performed by an internet user can be, in addition to side information such as time or location, used by advertisers to tailor the links specifically to the end user's interests. The "value" of placing such a dedicated link on the side of the returned search page is of course related to the overall market and perception by the advertisers of the odds that this particular user will end up following the links.

Online advertisement correspond to the major source of revenue for companies such as Google or Facebook. To illustrate the importance of SS auctions with some data, in 2021, Google's ad

revenue amounted to 209.49 billion dollars, with correspond to 80% of the company revenue that year [64]. In turn, Facebook amassed nearly 115 billion dollars from ad revenue in 2021, 97% of the total 117.9 billion generated by the company in 2021 [78] [79].

For some context of the magnitude of sponsored search auctions happening right now, it has been reported that Google processes 63,000 search queries every second, which adds up to 5.6 billion searches per day [80]. For each of these queries, there's an associated sponsored link auction to sell advertisement links.

#### 4.2.1.1 Basic model for sponsored search auctions

In a sponsored search auction, the auctioneer wants to sell a certain number of advertisement slots. The participating buyers are publishers, interested in the keywords related to the query. Each result page has a certain number of "slots" (i.e., predefined locations and sizes on an HTML page) offered; these slots have different levels of prominence, quantified into what is know as a Click-Through Rate (CTR). A CTR is the ratio of how many times an ad was, in the past, clicked by the final user over the number of times an ad shows on screen for its target audience (such an appearances is also known as an "impression").

Sponsored search auctions happen in milliseconds, before the query result page is loaded on the client device. Participants submit keywords and bids on the "cost-per-click". When the advertiser's keyword match a user's query, the advertiser automatically enters in the auction for these query's advertisement.

For example, a query containing the word "Ferrari" will trigger an auction for advertisers interested in the keyword "car". A certain number, $k$, of slots, each characterized with a ctr $\alpha_j$ are put on sale. The auction system then compares the bids from the $n$ advertisers interested in the "car" keyword. In accordance with the auction mechanism selected, slots are attributed to the $k$ best-bidding advertisers, and a price per click $p_i$ is computed for them. The query result page for "Ferrari" will then be shown to the internet user with the $k$ slots occupied by ads from the winning bidders. If the user clicks on one of the sponsored link, he will be sent to the advertiser's web page. The owner of the ad will then pay the search engine for sending the user to its web page, hence the name "pay-per-click" pricing.

#### 4.2.1.2 Generalized second-price sponsored search auction

The most common type of auction adopted for sponsored search auctions is Generalized Second Price (GSP) [63] [77]. First adopted by Google in 2002, it was soon adopted by other major search engines. A GSP auction can be described by the following steps:

1. accept a bid $b_i$ from each bidder $i$, and relabel the bidders so that $b_1 \geq b_2 \geq \ldots \geq b_n$;

2. assign each bidder $i$ of the $k$ first bidders to the $i$-th slot (the others lose);

3. charge each of the first $k$ bidders a price $p_i = b_{i+1}$ per click.

GSP auctions, though similar to Vickrey auctions (if $k$ = 1, GSP is the same as a Vickrey auction), is not truthful [74]. For $k > 1$ cases, there can be an incentive for participants to underbid, acquiring fewer clicks, but at a much reduced price that increases their utility [81].

### 4.2.2 VCG for sponsored search algorithm

A better alternative to GSP that is truthful and welfare-maximizing is a specialization of the VCG mechanism, seen in Section 4.1.5, for sponsored search.

The VCG for search algorithm, in which $n$ bidders vie for $k$ slots, each characterized with a ctr $\alpha_j$, can be outlined as follows, where the ctrs $\alpha_j$ are assumed down-sorted [82]:

1. accept a bid $b_i$ from each bidder $i$, and relabel the bidders so that $b_1 \geq b_2 \geq \ldots \geq b_n$;
2. assign each bidder $i$ of the $k$ first bidders to the $i$-th slot (the others lose);
3. charge each such bidder a price $p_i = \frac{1}{\alpha_i} \sum_{j=i+1}^{k+1} b_j(\alpha_{j-1} - \alpha_j)$, with $\alpha_{k+1} = 0$.

Intuitively, the price (per click) $p_i$ paid by the bidder $i$ is designed to compensate the loss in social welfare suffered by all the other bidders by the mere presence of the bidder $i$. In the framework of search engines, such a VCG algorithm must be run each time a web page is about to be displayed on a computer, meaning billions of times per day.

We provide in Listing 4.3 a specification (proofs omitted) in Coq of VCG for search, assuming that the bids are already sorted and that $k < n$. An auction is defined by two tuples, in $ctrs$ and $bids$, indexed by slots and agents. The VCG for Search algorithm expects thus as input a tuple $cs$ of down-sorted rates and a tuple $bs$ of $bids$, assumed as well to be down-sorted. The agent $i$ wins slot $i$ (with thus $i < k$), paying for it $price\ i$ to offset the negative impact on the global social welfare incurred by her presence. This value, as proposed by Vickrey, Clarke and Groves, is the sum of all the externalities, i.e., financial losses, of the agents ranked after $i$ according to $bs$, who thus do not get slot $i$.

For example, if $cs$ = (5, 3, 1) and $bs$ = (100, 50, 10, 4), then agent 0 will get slot 0 and pay $50 * (5 - 3) + 10 * (3 - 1) + 4 * 1 = 124$; agent 1, slot 1 for $10 * (3 - 1) + 4 * 1 = 24$; and agent 2, slot 2 for $4 * 1$ (agent 3 gets nothing; $cs$[3] is assumed 0).

```
1 Definition ctrs := k.-tuple ctr.
2 Definition bids := n.-tuple bid.
3
4 Variable (cs : ctrs).
5 Notation "'ctr_ s" := (tnth cs s) (at level 10).
6
7 Hypothesis sorted_ctrs : sorted_tuple cs.
```

```
8
9  Variable (bs : bids).
10 Notation "'bid_ j" := (tnth bs j) (at level 10).
11
12 Lemma slot_as_agent_p (s : slot) : s < n.
13 Definition slot_as_agent (s : slot) := Ordinal (slot_as_agent_p s).
14
15 Definition slot_pred (s : slot) : slot := ord_pred k s.
16
17 Definition externality (s : slot) :=
18   let j := slot_as_agent s in 'bid_j '(ctr_(slot_pred s)) '(ctr_s)).
19
20 Definition price i := if i < k then \sum_(s < k | i.+1 <= s) externality s else 0.
```

Listing 4.3: VCG for sponsored search

### 4.2.3 VCG for sponsored search in the industry

Previously to Facebook's adoption of VCG for search in 2006, both Vickrey auctions and VCG mechanisms were underrepresented in marketplaces, due to the complexity of the mechanism, the overall low revenue for the auctioneer and the need for users to have to reveal sensitive information such as their true value. These drawbacks of the mechanism are well known since the 1990s [75] [76] [77].

For a social network platform such as Facebook, the argument for adopting a welfare-maximizing auction for advertisements is that, in theory, users will only be shown advertisements that are relevant to them, and thus won't be overwhelmed with advertisements that would impair their experience in the platform. According to John Hegeman, Facebook's chief economist, VCG aids Facebook on its long-term focus on creating value for the participants, by prioritizing content quality over short-term revenue [83].

Facebook employs an hybrid version of VCG for sponsored search in which there is an extra factor associated to each participant, named the "quality score", an assessment from public's feedback on the quality of the advertisement. In order for advertisers to participate in Facebook's auctions, they first need to create an advertisement campaign. To do so, the advertiser needs to select the target audience for the campaign, composed according to an age group, a location as well as selected keywords of interest for the ad. It is also needed to set-up the budget and longevity of the campaign. To compute the bids, Facebook offer a few alternatives; while it is still possible for an advertiser to set its bids manually, it is discouraged, and advised only for "advertisers who have a strong understanding of predicted ads interaction rates". Otherwise Facebook calculates the bid for participants, taking in account the budget and longevity of the campaign, as well as the intent of the advertiser.

From Meta Business Help Center [2] and other sources of sources of information Facebook provides about its auctions for advertisers, it is never explained the type of auction that is taking place. It is stated that " we won't charge you more than your bid to show your ad" [84], but the auction mechanism works as a black-box for the advertisers. The problem with this lack of transparency is that there's lot of trust that advertisers need to place on Facebook. An advertiser has to trust that Facebook won't disclose or sell their information, such as budget to competitors. Discouragingly, Facebook doesn't have a great reputation of respecting the privacy of its users' data. As proof of this sorry state of affairs among many other lawsuits concerning privacy of user's data, one can recall the Cambridge Analytical data scandal of 2018 [2], in which personal data belonging to millions of Facebook users was collected without their consent by British consulting firm Cambridge Analytica, predominantly to be used for political advertising.

Regardless of privacy issues, advertisers have also to take Facebook's word on the validity of the auction results, and, if the advertiser uses the platform's automatic bidding tools, they also have to trust the platform's return estimations. Recently, evidence have arise of Facebook's dishonesty on revenue estimation for advertisers. On March 29, 2021, an US District, Judge James Donato, from San Francisco, ruled that a lawsuit accusing Meta Platforms Inc's Facebook of deceiving advertisers about its "potential reach" can proceed into a class action. This will allow potentially millions of individuals and businesses to sue as a group [3].

In consideration of these lack of trust and transparency in a digital advertisement platform, we believe that sponsored search would greatly benefit from the trust and transparency of blockchain technologies. In the next chapters, we explore a VCG for sponsored search implementation in smart-contract form that takes advantage of the transparency and trust of public blockchains to build more trustworthy solutions for VCG-like mechanisms.

[2] `https://www.facebook.com/business/help`

**CHAPTER 5**

# VCG FOR SPONSORED SEARCH IN SMART CONTRACT FORM: EXPERIMENTS FOR PERFORMANCE EVALUATION

*Dans ce chapitre, nous présentons les implémentations de VCG pour la recherche en tant que contrats intelligents, et nous réalisons deux études de référence en utilisant ces contrats comme base pour comparer différents systèmes de blockchain. Notre intérêt pour ce travail est double : (1) comparer différents systèmes de blockchain en tant que plateformes pour les contrats intelligents et (2) analyser l'impact des plateformes de développement distribué sur les mécanismes d'enchères tels que VCG pour la recherche.*

*La section 5.1 met en contexte le manque de rigueur scientifique dans les comparaisons de blockchains et souligne l'importance de la recherche présentée ici.*

*La section 5.2 présente notre implémentation directe de l'algorithme VCG pour la recherche en tant que contrat intelligent, à la fois en Solidity pour Ethereum et en SmartPy pour Tezos. Ces contrats servent de base à nos comparaisons de référence.*

*Cette étude comparative est divisée en deux parties. La première, présentée et discutée dans la section 5.3, se concentre sur la comparaison de la blockchain Ethereum, fondée sur la preuve de travail, populaire, avec l'infrastructure plus avancée fondée sur la preuve d'enjeu de Tezos.*

*La deuxième partie, dans la section 5.4, se concentre sur l'impact des mises à jour récentes du protocole Ethereum, notamment la solution de preuve d'enjeu (PoS) de la couche 2 offerte par Polygon et la mise à jour Ethereum Merge qui a introduit PoS dans la chaîne.*

In this chapter, we present implementations of VCG for search as smart contracts, and perform two benchmark studies using these contracts as a basis to compare different blockchain systems. Our interest for this work is two-fold: (1) to compare different blockchain systems as platforms for smart contracts and (2) analyse the impact that the distributed development platforms have on auction mechanisms such as VCG for search.

Section 5.1 contextualizes the lack of scientific rigor in blockchain comparisons and underscore the importance of the research here presented.

Section 5.2 presents our direct implementation of the VCG for search algorithm in smart contracts, in both Ethereum's Solidity and Tezos' SmartPy. These contracts serve as a basis for our

benchmark comparisons.

This benchmark study is divided in two parts. The first, presented and discussed in Section 5.3, focuses on the comparison of the popular, proof-of-work-based Ethereum blockchain with the more advanced, and proof-of-stake-based, Tezos infrastructure.

The second part, in Section 5.4, focuses on the impact of recent updates to the Ethereum protocol, namely the Layer 2 proof-of-stake (PoS) solution offered by Polygon and the Ethereum Merge update that introduced PoS into the chain.

## 5.1 Blockchain comparison

Although Bitcoin already dates from 2009, blockchain systems are still a fairly recent technology, just recently attaining mass attention[1]. The recent economical success of cryptocurrencies [85] [86] [87] and other decentralized tokens, e.g., Non-Fungible Tokens (NFTs) [88], could explain the many different alternative blockchain systems that appeared in the last decade, Ethereum (2015), Tezos (2018), Avalanche (2020), Polkadot (2020) and Polygon (2020), among others.

The recent sprouting of different blockchain systems, each with their own coins, consensus mechanisms, smart contracts, wallets, clients, and many more technical details, makes it difficult for researchers, investors, developers and general users to choose a system. Every new release of a new blockchain system, or an update to a new one, is always followed by press releases and blog posts boasting how better this new version is compared to the market staples, with the description of new consensus protocols, listings of faster of transaction times and/or promises of scalability [89] [46]. Although there are comparisons and surveys of blockchain systems [90] [91] [92] [93] [94] [95], there is still a lack of an agreed-upon, reliable method to compare different systems [96], on top of what we believe to be a lack of scientific rigour in the way the existing comparisons are managed.

We judge that with the in-depth comparative analysis below of our chosen blockchains as platforms for VCG for search contracts, we collaborate to the challenge of defining a proper way to compare different blockchain system, at least from a smart-contract point of view.

## 5.2 Naive VCG for search smart contract

For this benchmark study that intends to test and compare different blockchains as development platforms for smart contracts, we use a naive version of VCG for search. We refer to this VCG implementation as ”naive”, because it is a direct translation of the VCG for search algorithm to a smart contract form, without taking into account the repercussions that a distributed/public implementation can have on the auction algorithm. This version is not only utilized for our benchmark study of different blockchains, but it serves as basis for our study of the implications that the

[1]Google trends (`trends.google.com`) reveals that the peak of interest in the term ”blockchain” was in December 2017, which coincides with the (at the time) Bitcoin record value of $19,650

blockchain approach can have on the chosen auction mechanism (see, for a discussion of this kind of impact, Chapter 7).

Implementing on a given blockchain infrastructure VCG for search as a smart contract depends on the programming language adopted by the blockchain platform. In this thesis, we implement VCG for search using two distinct languages: Solidity, for Ethereum and other Ethereum-Virtual-Machine-(EVM)-compatible chains, and SmartPy, for the Tezos blockchain. We strive to have similar code for both implementations, in order to make the comparison as fair as possible. For the presentation of the contract, we use Solidity in code examples, since its usage is more intuitive than SmartPy.

A typical contract execution can be described by a sequence diagram (see Figure 5.2). There are two types of actors involved, auctioneer and bidders, the same as in every auction. The whole process is divided into three parts. First is the auction setup; for it, the auctioneer calls the function[2] `openAuction[uint[] calldata CTRs]`, which starts a new auction, using the provided argument as the appropriate CTRs values to be auctioned. Once the auction is opened, bidders can call `bid(uint)` to store their bids in the contract. Once the bids are received and the auction can be closed, it is up to the auctioneer to perform the needed `closeAuction` transaction. The contract will then generate the VCG-specific list of winners and their respective prices.

As mentioned in Section 3.4.1, smart contracts are characterized by their storage (estate) and code (functions). We discuss below these two key parameters for our VCG implementation.

### 5.2.1 VCG contract storage

The following data structures are used to implement VCG for search[3]:

- `owner(address)`, the address of the user who owns the auction smart contract and has the role to act as the auctioneer;
- `isOpen(bool)`, a flag indicating if an auction is currently opened;
- `ctrs(uint[])`, the array of CTRs of the slots being auctioned;
- `bids(uint[])`, the array of bids sent by the bidders;
- `agents(address[])`, the array of addresses of the bidders;
- `prices(uint[])`, the array of prices updated at the end of the auction with their proper values.

[2]The `uint` type denotes unsigned integers; the `[]` notation is used to introduce arrays of variable length; the keyword `calldata` is used to indicate that a value is located in the argument storage.

[3]The `bool` type is used for booleans; `address` is the Solidity builtin type for identifying participants.

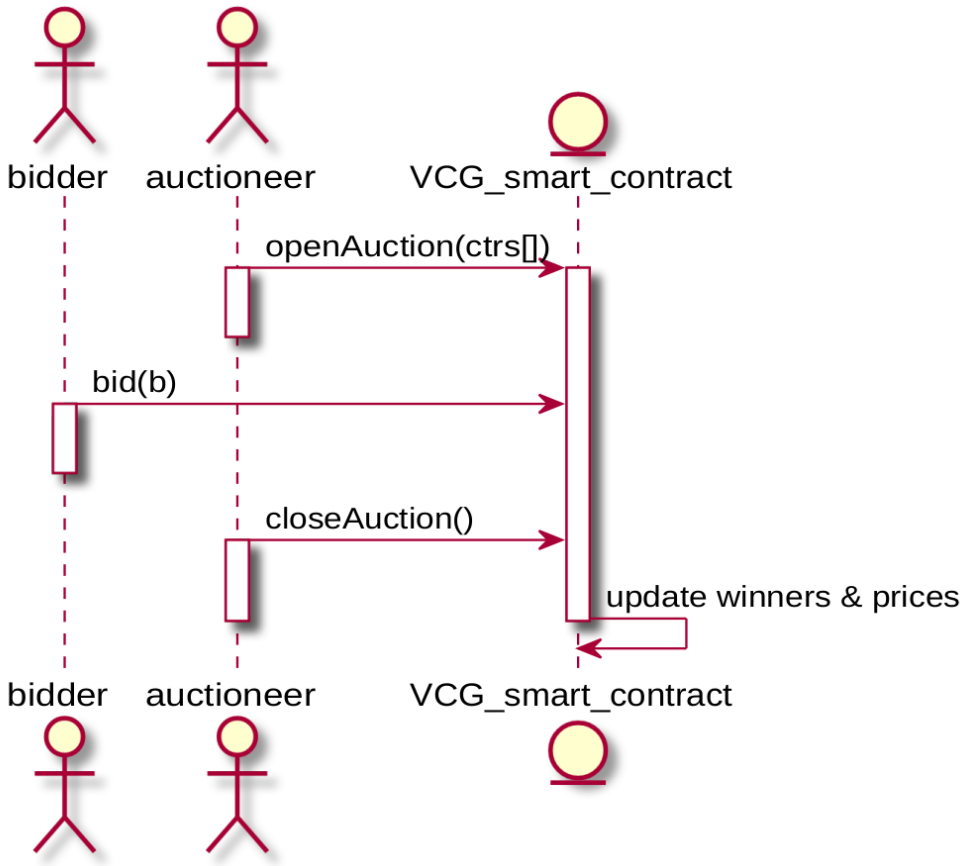


Figure 5.1: Sequence diagram of the VCG for search smart contract

### 5.2.2 Public functions

Here we specify the main public functions (these would be called "entry points", in SmartPy) of the VCG for search contract (only the `bid` function is not reserved to the contract owner role):

- `transferOwnership` transfers the contract ownership;
- `openAuction` opens an auction, providing it an initial `ctrs` array argument;
- `bid` enables one bid from a participant to be received by the contract (its value argument is stored in `bids`, while the originator address is registered in `agents`);
- `cancelAuction` cancels the auction (the bids and agents' addresses are erased);

- `closeAuction` closes the auction, sorts the bid list, via insertion sort, and updates the `prices` array with the proper VCG for search values.

In order to perform the sorting of bids required when closing a VCG auction, our contract implements the "insertion sort" algorithm. This choice was first adopted with the intent of incrementally sorting the bids as they arrive through the different bidding transactions. It became however apparent that later bidders would need to have to bear with higher transactions fees, since the list of bids would be increasing along the bidding process, leading to possible longer insertion steps, which could be considered as unfair. Moreover, through the gas used by a bidding transaction, one could have inferred some information about the number of previous bids and thus gain some insight about one's chance of winning the auction, creating again unfairness and possible bias.

Consequently, we decided to exclude the sorting of the bids from the bidding phase, including it instead inside the `closeAuction` function, as to make the auctioneer bear the costs of sorting. Although insertion sort is not adapted to large sets [97] [98], we decided to keep this already existing implementation; since both Solidity and Tezos' contracts are the same in this regard, the exact choice of sorting algorithm doesn't affect our comparison analysis.

### 5.2.3 Test protocol

Our benchmark study relies on a unit test that performs the following transactions on each tested blockchain:

- deployment of one VCG smart contract per auction;
- opening of an auction with a specified set of CTRs;
- bidding phase, during which a variable number of bids are issued to simulate participants;
- auction closure, producing tables of winners and prices.

We opted to deploy a new contract each time a test is performed to better track the costs incurred by the addition of more storage to a contract (e.g., the burned XTZ for Tezos, see Section 3.5.3.3).

## 5.3 Proof-of-work and proof-of-stake benchmarks

Our first benchmark study came from the desire to compare two smart contract platforms with two different consensus approaches, proof-of-work from Ethereum and proof-of-stake from Tezos. This study was realized between the last quarter of 2020 and the first quarter of 2021, and the results were gathered in an article titled "Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos" [7] and presented in the Fourth International Symposium on Foundations and Applications of Blockchain 2021 (FAB '21).

### 5.3.1 Ethereum versus Tezos

Ethereum's launch in 2016 marked the start of the second generation of blockchains, due to its introduction of smart contracts that enabled the appearance of decentralized applications. Following the success of Ethereum, a myriad of different blockchain platforms supporting smart contracts emerged on the crypto scene, e.g., Tezos, to the point that today, choosing a blockchain system to host a so-called "distributed application", or dApp, is no straightforward choice. Indeed, the parameters to consider are plentiful: popularity of the blockchain, prices, technological particularities, compatibility with other chains, environmental impact, to cite a few.

As presented in Section 5.1, we believe that more scientific work needs to be done in the way blockchain comparisons are presently conducted. We intend to contribute to this effort by the present analysis of two key architectures, Ethereum and Tezos, using a real-life use case, the Vickrey-Clarke-Groves auction for sponsored search (VCG) algorithm.

### 5.3.2 Development and tests

Both blockchains provide online integrated development environments (IDE) for programming and testing smart contracts: Remix IDE[4] for Solidity and SmartPy IDE[5] for SmartPy/Tezos. Both IDEs' standard method for testing is to simulate the contract in a sandboxed blockchain [99]; if this is appropriate to check the functional side of smart contracts, it is not representative of the full-fledged behavior of an actual blockchain. To provide performance results that we believe to be more pertinent, we opted to test our VCG contract in "testnets", i.e., Ropsten for Ethereum and Delphinet for Tezos, which are bona fide versions of the corresponding blockchains dedicated to tests, and but without requiring to pay for smart contract execution. This way we can expect to experience the behavior of a full-fledged blockchain without paying the transaction fees, though we still felt the need to validate the results by comparing our results to both mainnets' actual data (in Section 5.3.5.3, 5.3.5.4).

To deploy and communicate with our contracts, we used Truffle, a development environment for smart contracts[6], initially conceived for Ethereum development, but for which a Tezos integration, though still under development, presented enough functionalities for our tests. Truffle's contract abstraction provides means to interact with contracts via JavaScript. We present the infrastructure for both tests in detail in Sections 5.3.3.2 and 5.3.4.2.

Our full test for this first benchmark consists of a series of unit test auctions, each with increasing numbers of participants and slots, namely 10 bidders with 4 and 8 CTRs, 20 bidders with 4, 8 and 16 CTRs and 50 bidders with 4 CTRs. Recall that each CTR correspond to a slot being auctioned; thus, in a run with 10 bidders and 4 CTRs, we have 4 winners with corresponding prices per click. We halted our tests after 50 participants and 4 slots because the gas consumption was

[4] `remix.ethereum.org`
[5] `smartpy.io/ide`
[6] `trufflesuite.com`

```
1     event Open(
2         uint[]  ctrs
3     );
4     event EndAuction(
5         address[] agents,
6         uint[] prices
7     );
8     event Bid(
9        address error,
10       uint price
11        );
```

Listing 5.1: Solidity events

already high enough to reach the gas limit of a Ropsten block; we discuss the important impact of this performance limitation in Section 5.6. We note $n\_m$ an auction with $n$ participants and $m$ slots.

### 5.3.3 VCG in Ethereum

We describe here the way VCG contracts are implemented within Ethereum, the first blockchain to support smart contracts and still today the most popular choice for the development of decentralized applications. Starting with a presentation of the particularities of our Solidity contract (Ethereum's programming language for smart contracts [20]), we also present the infrastructure used for our tests and finish with an analysis of the limitations to our tests imposed by the blockchain.

#### 5.3.3.1 Solidity contract

The VCG contract in Solidity for the Ethereum blockchain is given in Appendix A and is rather straightforward to understand. Yet, it exhibits some particularities compared to its SmartPy counterpart, which we discuss below. For instance, in Solidity, one can use events (see Section 3.4.1.2) to "subscribe" to the contract and monitor it's functioning from outside of the blockchain, for example from a front-end application. In our VCG contract, we introduced events for auction opening, closing and bids (see Listing 5.1).

Another advantage of Ethereum is its support for inheritance between contracts. Our VCG contract expands another small contract called "ownable", to be able to grant the role of "owner" to any account. In our VCG contract, "owner" has the power to be the auctioneer, or grant this role to someone else. Solidity's inheritance support is particularly useful when paired with secure libraries for smart contract development such as OpenZepellin[7], a repository of audited contracts

[7] `openzeppelin.com`

and libraries, well known and utilized by the Ethereum community, which helps increase confidence of the reliability of one's contracts. Though there is an "ownable" contract available in OpenZepellin's libraries, we decided, for ensuring the self-contained nature of our test program, to create our own for this benchmark.

Solidity has two types of memories: "storage", for the contract's stable state variables, registered in the blockchain, and "memory", which only stores variables during the execution of a transaction. Memory is cheaper to use (in terms of gas) than storage, because there is no need to update the blockchain' data when it is used [100]. The actual kind of memory used for keeping the values of variables can affect their behavior, a prime example being arrays, which are crucial for our VCG contract. Indeed, only "storage" arrays can be of dynamic size; "memory" arrays have to be of fixed size.

The last point we need to mention is the presence of visibility qualifiers for the contract variables and functions. These can be one of the following: "external", for functions only, meaning that the function is to be called from other accounts, being them users or contracts; "public", for functions and variables that can be called/read externally and internally; "internal", for functions/-variables that can only be used internally or by derived contracts; or "private", for functions/variables that can be called/read internally, but not in derived contracts [101]. A proper usage of these qualifiers helps to keep the contract more organized. For instance, our most complex function, `closeAuction`, uses two other internal functions: `swap`, to swap the positions of two array items, and `calculatePrice`, to compute the winners and prices at the end of a VCG auction.

#### 5.3.3.2 Ethereum Infrastructure

The infrastructure for Ethereum testing we used is summarized in Figure 5.2. Its main components are:

- a Python script that generates JavaScript test files according to specific testing parameters;
- a Metamask crypto wallet that hosts fake testnet coins to pay for the test transactions;
- Truffle, the development environment that is used to forge the transactions, sign them with HDWalletProvider and send them to Infura;
- HDWalletProvider, a library that can be used to sign transactions with accounts' mnemonics;
- Infura, a development suite that provides APIs to access Ethereum networks;
- and, finally, the Ropsten network, the test blockchain running our VCG smart contract;

Testnets are free, but in order to be able to use them, one needs an account holding enough testnet Ether (the cryptocurrency specific to Ethereum). We used the popular cryptocurrency wallet MetaMask [33] to create an account for our tests. Metamask uses a mnemonic phrase from which

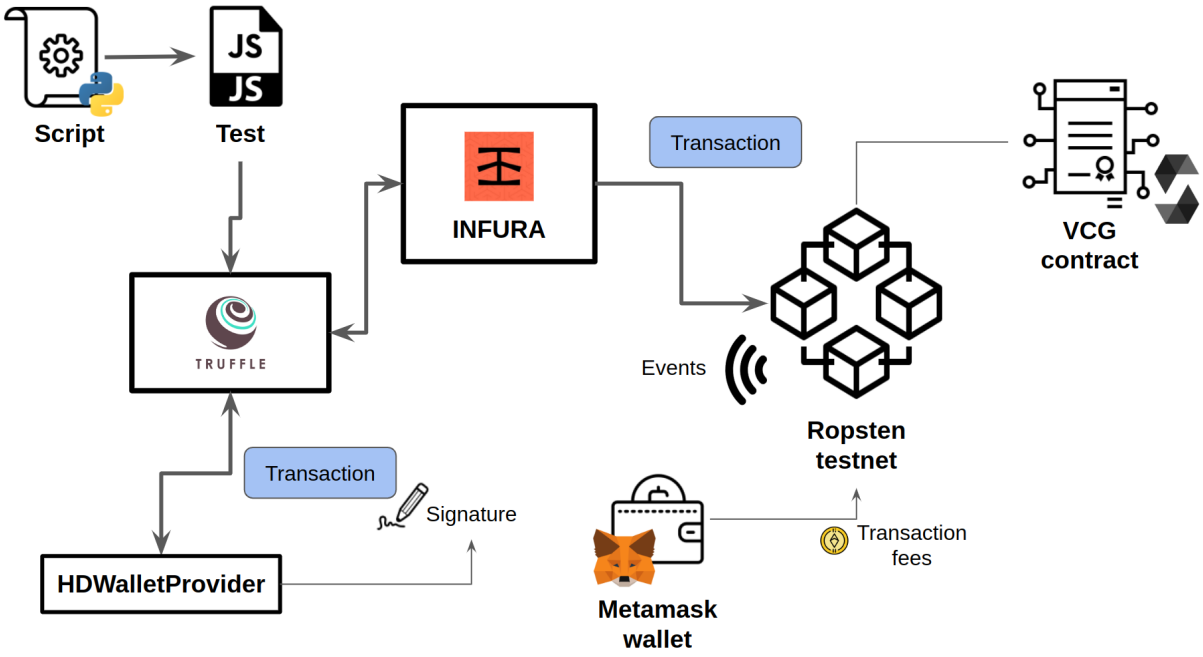


Figure 5.2: Infrastructure for Ethereum's test

it will generate a public and private key pair; the corresponding account address is derived from the last 20 bytes of the hash by the hash Keccak-256 function [102] of the public key [103]. With an account created, one can use one of the available Ropsten so-called "faucets" applications to obtain (free) Ropsten Ether coins. With this wallet setup ready, we utilized of a Python script to generate the JavaScript tests files for both Ethereum and Tezos (see below).

The JavaScript test files were executed via a Truffle terminal. Truffle [56] is a development tool for smart contracts, that offers a test framework for such applications. However, in order to execute these tests in a blockchain (either mainnet or testnet), Truffle needs assistance. For performing API calls to the Ethereum Ropsten network, we used Infura [58], a well-known Ethereum API provider; this saved us the work of setting up an Ethereum client on our test machines, just for performing contract tests.

Finally, since Truffle cannot sign transactions on its own, in order to do so, we used HDWalletProvider. HD (Hierarchical Deterministic) WalletProvider [104] obtains the test transactions forged by the Truffle environment and signs them with our wallet's mnemonic.

#### 5.3.3.3 Limitations

Even though our test environment is rather sophisticated, it still suffers from limitation. Interestingly for our comparison purposes, all these were derived from the Ethereum environment. At the time of our tests, Ropsten was the testnet that better reproduced the behavior of the Ethereum mainnet; therefore, all the restrictions suffered by our tests would be also experienced in the "real world", i.e., were we to run our contract onto the mainnet.

**Gas limit** During our tests, we were not able to close an auction with 4 slots and 50 bidders, since such transaction would take close to 8 million gas.

In Ethereum's mainnet, there is a gas limit for each block; currently, it is set to 15 million units of gas (though, depending on user's demand, it can be doubled to 30 million units of gas). For the Ropsten testnet, the limit is much more flexible, regularly staying under 10 million. There is no rule for how much gas a transaction can take to itself, as long as it stays within the block limit. Theoretically, if listed with a high enough gas price to take the lead over all other transactions in the mempool, it could be possible to fill a block with a single transaction. Yet, it is ultimately the miner's decision on how to allocate transactions to compose a block. From our experience with the tests executed in this benchmark, we were not able to execute transactions with a gas usage of more than 8 million gas.

**Pending transactions** Another problem that we encountered occasionally when executing our test protocol were "pending" transactions. This occurs when a transaction is under-priced compared to the network demand; if so, it stays in the so-called Mempool of to-be-run transactions until a miner treats it.

In Ethereum, each transaction contains a number called "nonce" associated with the user's address. This nonce is the number of transactions sent by that given address, which has to be always increasing; for example, the system will not allow a user to submit a transaction with nonce $2$ before one with nonce $1$ is treated. This is a measure introduced to prevent double-spending [105].

When a transaction is pending, one cannot use the same address to send new transactions, until the pending one is treated. This is a big hindrance for our scripted test approach. To get rid of a pending transaction, we had to forge a substitute one with the same nonce used by the pending transaction. Since this situation was not managed by our test script, each time a transaction was pending for a long period of time, the test run had to be put to a halt.

Although we list pending transactions as a limitation, it is in fact a consequence of our test approach, i.e., to try to automate blockchain transactions without taking explicitly into account this issue.

**High transaction prices and faucets** As already mentioned, this benchmark study was conducted between 2020 and 2021, when interest for Ethereum were at an all-time high, resulting in a fierce competition for block space, even on testnets, resulting in high gas prices on Ropsten. Therefore, our wallet would be depleted after just a couple of test runs, and we would thus need to often replenish it via Ropsten faucets, which were not always reliable, due to the high number of users at the time. This was a significant hindrance at that time.

### 5.3.4 VCG in Tezos

As done for Ethereum, in this section we present the Tezos development of the VCG for search contract.

Tezos is a third generation blockchain, with a proof-of-stake consensus and a number of smart contract programming languages that were, in particular, designed for easy formal verification [106]. Among these programming languages for smart contracts, the main ones are SmartPy and Ligo [107], which are both compiled into the lower-level Michelson dialect, the domain-specific, stack-based language used for the implementation of Tezos's smart contracts [108]. For our Tezos VCG implementation, we opted for SmartPy as our programming language, since it is widely used and offers a new twist on programming contracts via a kind of high-level metaprogramming approach.

The infrastructure for our test is based on Ethereum's (presented in Section 5.3.3.2), and we present the main differences in the following subsections. Finally, as with Ethereum, we present the limitations that the Tezos' development environment imposed in our tests.

```
@sp.entry_point
    def bid(self, params):
        l = sp.local('l', sp.len(self.data.bids))
        self.data.bids[l.value] = (params)
        self.data.agents[l.value] = sp.sender
```

Listing 5.2: SmartPy entry point for bidding

#### 5.3.4.1 SmartPy contract

SmartPy is a high-level smart contract library for Python[8]. SmartyPy has special constructs for the development of contracts, though the syntax is regular Python. Note in particular that even SmartPy-dedicated control structures have been introduced, for instance for `while` loops, which makes for a somewhat unusual programming model.

When starting a contract, it is first necessary to import the SmartPy library into the contract Python file, via `import smartpy as sp`. The VCG contract starts by defining a VCG class, inheriting from the "contract" class from SmartPy `class SponsoredVCG(sp.Contract)`. SmartPy contracts do not provide functions as do their Ethereum counterparts; instead they offer entry points that users have to specify when calling a Tezos smart contract. For example, the bidding entry point of the VCG contract is defined in Listing 5.2.

Another distinguishing point of SmartPy is the need to specify the types for expressions, which normally is not required in Python, but is required in SmartPy's target language, Michelson [110]. For example, we can see the casting to natural number `nat()` of the indexes of our bids and agents mappings in the (sub) entry point `insertSort()` in the Listing 5.3, the entry point for insertion sort, used in the VCG contract to compute the winners. As one can see on Line 15, a function to cast values to naturals has been introduced in order to get less cluttered code. The full SmartPy contract can be found in Annex B.

#### 5.3.4.2 Tezos infrastructure

The infrastructure for the Tezos tests is diagrammed in Figure 5.3. By comparing it with Ethereum's counterpart in Figure 5.2, one can observe that the Tezos one is less complex, which hints to easier development. Unfortunately, on the other hand, Tezos is considerably less popular than Ethereum[9], which reflects in limited support and community of developers, which in some sense is reflected in our infrastructure. As mentioned before, Tezos offers a powerful IDE[10] for the development,

[8]Recently, SmartPy grew to include both TypeScript and OCaml syntaxes for its contracts [109], with SmartPy's IDE (`smartpy.io/ide`) support still in beta and alpha testing, respectively

[9]At the time of writing, Tezos has 2.9M accounts [111], while Ethereum sports a staggering 201.23M accounts [112]

[10]`smartpy.io/ide`

```
@sp.sub_entry_point
    def insertSort(self, param):
        self.data.isOpen = False
        jx = sp.local('jx', param)
        jint = sp.local('jint', sp.to_int(param))

        def swap(i):
            tempBid = sp.local("tempBid", self.data.bids[nat(i)])
            tempAgent = sp.local("tempAgent", self.data.agents[nat(i)])
            self.data.bids[nat(i)] = self.data.bids[nat(i - 1)]
            self.data.agents[nat(i)] = self.data.agents[nat(i - 1)]
            self.data.bids[nat(i - 1)] = tempBid.value
            self.data.agents[nat(i - 1)] = tempAgent.value

        def nat(x):
            return sp.as_nat(x)

        sp.while ((nat(jint.value) > 0) & (self.data.bids[nat(jint.value)] >=
    self.data.bids[nat(jint.value - 1)]) ) :
            swap(jint.value)
            jint.value -= 1
```

Listing 5.3: Casting indexes to natural in SmartPy

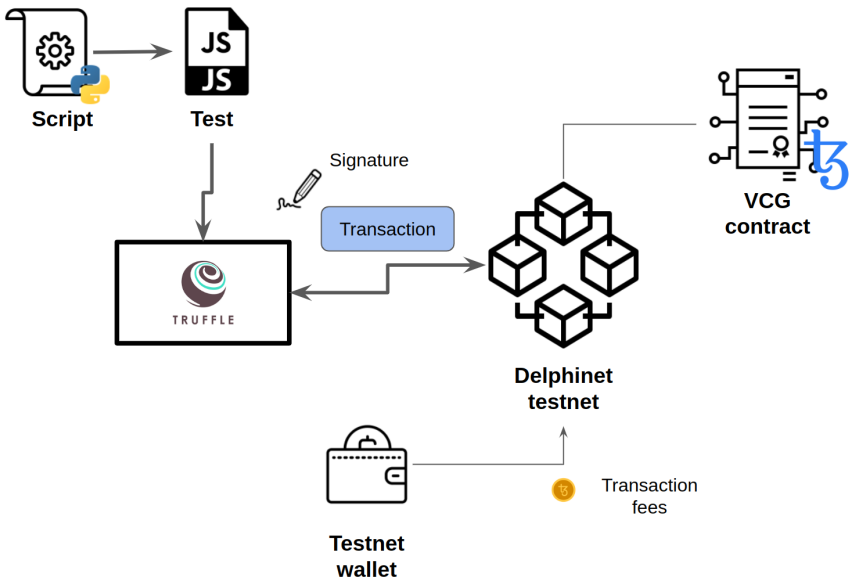


Figure 5.3: Infrastructure for Tezos testing

deployment and test of its smart contracts, but since our goal is to have the closest comparison possible between both systems, one needs to go beyond this simulation environment.

In Tezos' case, we did not use a MataMask wallet to create a test account; instead, we relied on pre-constructed testnet wallets available[11], supplied by the Tezos itself. Fortunately, there is also an experimental version of Truffle for Tezos, which enabled us to execute the same tests in both blockchains. In Tezos' case, a HDWalletProvider to sign transactions is not needed, since Truffle does it, when a private key is stored in a proper configuration file. Finally, the last difference between both infrastructures is that we do not needed Infura to connect to the Delphinet testnet.

#### 5.3.4.3 Limitations

Compared to the Ethereum's tests that were plagued with network issues, Tezos' ones were smoother, probably due to the proof-of-stake consensus, but also to the lower number of network users. Some of limitations encountered in this case were related to the infrastructure. We started our tests using Tezos' Tezster-CLI [113], a tool box to build, deploy and interact with Tezos' testnets, though it proved to be unfit for out serialized test approach; fortunately we could use the Truffle integration for Tezos, which allowed us to reuse part of the infrastructure used for the Ethereum tests. Yet, because Tezos' Truffle integration is still experimental, it is very rigid in its parameters, and we were not able to configure our own gas limit or gas prices; they remained preset by Truffle.

Finally, the biggest limitation came from Tezos' self-amending properties [114], which enables the chain protocols to update without the need of a hard fork of the blockchain. Upgrades to the chain occur every few months, and every time one happens the chain becomes unstable and some tools become buggy, due to incompatibilities with the new protocol. During the development of our tests, we witnessed the change from the Carthage [115] proposal to the Delphi [116] one, which forced us to rework our test scripts. While the Edo [117] proposal update was approaching, in November 2020, we felt pressured to finish all our tests in time, in order to avoid possible complications with the new update. This is a design issue specific to Tezos which may, as one can imagine, have a long-lasting impact on the popularity and ease-of-use of this infrastructure, at least until a solution to this important problem is provided.

### 5.3.5 Results

Our comparison analysis between Ethereum and Tezos when running VCG for search auctions focuses on three key performance parameters: programmability, performance and cost. The metrics for our comparison are derived from the data available through Truffle and the blockchains' block explorers (see Section 3.6.1), Etherscan[12] for Ethereum and TzStats[13] for Tezos. The parameters selected were gas and burned coins, block time and transaction fees cost (in dollars) as metrics

[11] `faucet.tzalpha.net`
[12] `ropsten.etherscan.io`
[13] `tzstats.com`

for our comparison's parameters. Gas and burned coins are used for computation (CPU, storage) performance assessment; they also help illustrate how both platforms units of gas are not directly comparable. "Wall-clock" execution time could be considered as the time performance parameter, but when working with smart contacts, since it is directly linked to each blockchain's block time, we consider the latter as a more representative time comparison parameter. Finally, monetary considerations are strongly linked to blockchain technologies, so we look at cost issues related to our contract execution. Cost is, in some sense, a better parameter for comparison than the previous ones, since it is a good indicator of the practical usability of smart contracts and blockchains.

#### 5.3.5.1 Programmability

The first, important, though somewhat subjective, point of comparison between the two blockchain environments is the ease of programming and interacting with smart contracts.

Ethereum's main language for writing smart contracts is Solidity, similar to Java and C++, which we judge to be straightforward for anyone accustomed with object-oriented programming. Ethereum also has a very active community of developers, with a fair amount of support for Solidity developers. Ethereum Improvement Proposals (EIPs) [24], as by their namesake, are improvement proposals that are discussed by the Ethereum community, and in the case of smart contracts generate Ethereum Request for Comments (ERCs) [118] that are documented standards for the development of contracts[14]. Standards are very important for smart contracts development in Ethereum. Since trust and security are crucial for the development of dApps, the adoption of one of the many agreed-upon standards for contracts is a way to give some credentials to a contract. There are repositories of verified and audited contracts, implementing different ERCs and other contract standards, which, again, can be used to boost user confidence in a dApp [119].

According to Analytic Insight's survey on Ethereum development tools, "Ethereum development tools are one of the best options to develop dApps or smart contracts" [120]. And, indeed, big staples of the crypto-industry are being first designed to serve the Ethereum blockchain, namely Metamask [33], Truffle [56], Hardhat [121] and Infura [58]. All these tools, as we could attest during our tests, are highly effective, making the process of deploying and communicating with a blockchain streamlined. It is also worth noting that the Ethereum organisation offers interesting and educational tools such as Solidity's IDE, Remix[15], which offers the possibility to execute transactions broken down by EVM op-codes and to follow the changes in storage and gas consumption.

Tezos, on the other hand, is considerably less popular than Ethereum, with support either technical or from the community of developers being more scarce. For our development and tests, we had to rely on tools developed by the Tezos Foundation, and be in contact directly with the Tezos' team. In terms of programmability, Tezos offers four languages for smart contracts, including

[14] ERCs are very important for smart contract development in Ethereum, the most prominent being ERC20 [26] and ERC721 [28], which define fungible tokens (other coins in the Ethereum network) and non-fungible tokens (NFTs) respectively.

[15] `remix.ethereum.org`

Michelson, the stack-based language that is, in the end, executed inside its blockchain. As already mentioned, we opted for SmartPy, a Python library; scripts are then regular Python scripts that use SmartPy constructs. SmartPy, in our experience, presented a steeper learning curve compared to Solidity; first, there is the need to specify types, which are not native to Python and, at first, felt very unfamiliar; and second, SmartPy relies on meta-programming, i.e., the functions described in the program are used to construct a smart contract that will only be executed once the contract is deployed. Meta-programming can be a bit daunting for inexperienced users.

Finally, in terms of technical support, Tezos proved more difficult to put to use than Ethereum. We started our tests with Tezster-CLI, a specific tool for Tezos's contracts, which happened to be not easily adaptable to our sequence of unit tests. Fortunately, there was an experimental version of Truffle for Tezos, which not only was more adapted to our type of test, but also brought our infrastructure closer to Ethereum's.

#### 5.3.5.2 Gas and Burned

The experimental data obtained vary significantly according to the phase of the VCG auction process and the blockchain on which they run.

**Deployment.** On both blockchains, the gas for each deployment of a VCG contract is always the same. Ethereum consumes 976,061 gas, while Tezos needs 24,017 gas while burning 1.183 XTZ for the allocation of 4,475 bytes.

**Opening.** When opening an auction, the ctrs are stored in the blockchain. As can be expected, the gas and burned increase linearly with the number of slots being auctioned.

**Bidding.** Bids behave differently on each blockchain. Ethereum is more homogeneous, with the first `bid` transaction always needing more gas, since the first `push` sets up the storage for the array of bids and agents. The first bid consumes 105,917 gas, while the subsequent bids, having only to insert a `uint` and an address, always consume 75,917 gas.

For Tezos, gas consumption increases with a mean of $208.2 \pm 1.1$ (s.d.) with each subsequent bid, while the amount of coins burned is constantly 0.00925 XTZ or 0.0095 XTZ, depending on the size of the bid.

**Closing.** The close auction function/entry-point is the most relevant for our comparison, since the bulk of the VCG algorithm is performed here. The array of bids is sorted (we implemented a simple insertion-sort algorithm), and this sorted array is used in the third step of the VCG algorithm in order to compute the prices for the winners. Figure 5.4 is a graph of the closing gas for each of our tests. Note that it was not possible to close the 50-bid auction in Ethereum, the gas surpassing what the Ropsten network is accepting as gas limit for a single auction.

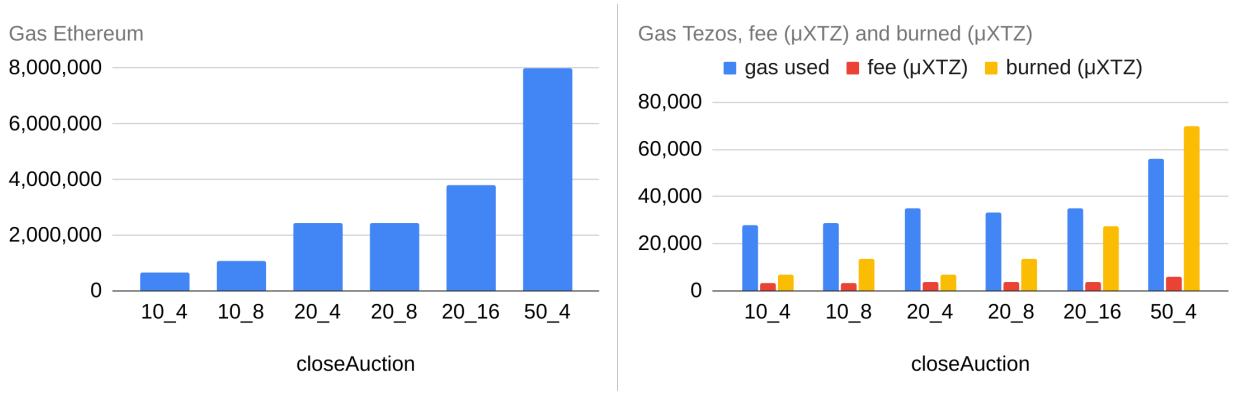


Figure 5.4: For each VCG contract $n\_m$ closing transaction, gas consumption on Ethereum (left) and gas, fee and burned for Tezos (right, where the Y axis scale is in gas and $\mu$XTZ).

#### 5.3.5.3 Block time

For Ethereum's main network, using Etherscan, we measured the block time at 14.82 $\pm$ 1.63 (in seconds), while the Ropsten network clocks at 14.5 $\pm$ 1.2 seconds. Measuring Ropsten directly from Truffle, we got a mean of 14.16 $\pm$ 7.72. For Tezos, its main network is advertised as providing a constant block time of 60 seconds, while Delphinet uses half of it, i.e., 30 seconds. Using Truffle, our tests on Tezos showed a block time of 43.07 $\pm$ 14.63 seconds. Note that we are not waiting for the suggested confirmation blocks.

#### 5.3.5.4 Price

For our experiment, coins on testnets are free, so the ETH and XTZ amounts that were spent for this benchmarking had no actual value. Yet, an approximate prediction of the prices one would have to pay to run our VCG test can be obtained by taking the main network prices for both of these coins. At the time of our experiment (Mar 10 2021, 10:24 UTC), one ETH is valued at $1,827, and one XTZ is $4.25.

For Ethereum, we used ETH Gas Station[16] to get a quote for gas prices. For test purposes, we used the price category "Standard" (91 Gwei/gas, at the time of test), which led to the following prices for the deployment and bidding phases: $162.27, and $17.6 (first bid) and $12.6 (subsequent ones). The varying closing phase prices can be deduced from Figure 5.4; for instance, the price for a $10\_4$ auction was $49.8.

For Tezos, we used as transaction fees the ones automatically suggested by Truffle, while the burned costs, related to storage increments, are 0.00025 XTZ for 1 byte at the time of test, for both the main and testnets. The prices for the deployment phase are $0.03 for fee and $5.02 for burned. For the bidding phase, the fee paid by each bidder increases by $0.000088 $\pm$ 0.0000029 each

[16] `ethgasstation.info`

time, while the burned remains somewhat constant, between $0.039 and $0.04. For the closing phase, one can refer to Figure 5.4 to get an estimate, where, for a 10_4 auction, the fee paid by the auctioneer would be $0.133 and the burned, $0.028.

### 5.3.6 Discussion

Our goal with this benchmarking study was to compare the performance of two very similar smart contracts on Ethereum and Tezos. Translating a Solidity contract to the Tezos blockchain environment proved to be quite difficult, even though this could be somewhat expected since Ethereum is the most popular dApps platform, with thus a lot of support from its community, while Tezos, at the time of this writing, is much less used. From our experience, most complications with Tezos are inherent to its design philosophy. In particular, the self-amending property of this blockchain translates into testnets being abandoned every time there is a new protocol upgrade, which led to temporary complications for our study, either because of bugs or because some tools were not adapted to the new testnet as fast as expected.

Ethereum's scalability characteristics is a big drawback for our VCG implementation. The gas limit for blocks implies a very small limit for the number of bidders, especially when compared to standard VCG auctions in industry. Adding the system's popularity to the scalability problems is rising gas prices, which results in a high average transaction fee of $39.49 (recorded in February 23, 2021). These values could be considered acceptable for a transfer-value system, but, for a dApps platform, they could lead to users abandoning the system. However, the EIP 1559 [122] proposal to reform the Ethereum fee market and the introduction of a proof-of-stake approach within Ethereum Merge are two welcoming changes that could positively impact the Ethereum results in our benchmarking.

## 5.4 Benchmarking Ethereum's upgrades

Our first benchmark made clear the limitations of proof-of-work (PoW) in face of proof-of-stake (PoS); Tezos outperformed Ethereum in terms of scalability and price. Ethereum's popularity grew above what the chain's PoW consensus could support, reaching a critical point between 2020 and the first quarter of 2021 (period in which we performed our comparison test between Ethereum and Tezos, in Section 5.3.1). DappRadar's[17] "2020 Dapp Industry" Report [123] describes that, in a record breaking year for transaction volume for the blockchain industry, with 270 billion transactions, 95% of these came from the Ethereum ecosystem. The corresponding competition for space in a block made the gas prices rise considerably, as is visible on the "Ethereum average transaction fee graph" from the investment research platform YCharts [18], in Figure 5.5, and, as shown in our first benchmark comparison in Section 5.3.6, the high transaction prices limits the usability of the

[17] `dappradar.com`
[18] `ycharts.com`


Figure 5.5: Ethereum Average Transaction Fee [123]

chain. Finally, in face of the high energy consumption and negative ecological impact of proof-of-work consensus, this technology is being criticized at large, with many condemning blockchain technologies in general as environmental threats [124] [125] [126].

In face of the aforementioned issues, the blockchain industry in general is trying to depart from technologies that rely on a proof-of-work consensus. DappRadar's 2021 "Dapp Industry Report" [127] reveals a migration of users from Ethereum to other PoS-based blockchains. No longer encompassing 95% of dApps transactions, Ethereum now holds 60% of this market, due to more advanced and appealing systems such as BSC, Solana, Terra, Avalanche, Hive, or Wax entering the field while offering more robust alternatives to the one presented by Ethereum.

Yet, while new independent chains are appearing, there are "third party" groups that still wager in Ethereum and want to profit from its success. One example is the recent appearance of so-called layer 2 solutions. The Ethereum Foundation defines layer 2 as "a collective term to describe a specific set of Ethereum scaling solutions. A layer 2 is a separate blockchain that extends Ethereum and inherits the security guarantees of Ethereum" [128]. These solutions take advantage of the underlying security infrastructure of Ethereum, but offer lower transaction prices. The most widely adopted layer 2 solution is Polygon PoS, a proof-of-stake blockchain that operates in parallel to Ethereum and is EVM-compatible, meaning that it can run Ethereum's smart contracts.

Ethereum is also actively trying to update its protocol. First, on August 2021, Ethereum activated the London fork, which implemented the EIP-1559 [122], to remodel its gas system. And the bigger update to the chain yet, known as "Ethereum Merge", occurred in September 2022; it intends to add PoS functionalities to Ethereum [46].

We present below the results of a second experiment based on VCG for search on these new infrastructures, and compare them to the original Ethereum.

### 5.4.1 Target

For our second benchmark, we decided to stay solely within the Ethereum system, its better programmability and support, in terms of tools and community, in our opinion, offering more possibili-

ties for the progress of our VCG contract development and analysis. For comparison with Ehereum mainnet, We chose two of Ethereum's improvement solutions: the layer 2, PoS-based "Polygon PoS" and Ethereum Merge (in the form of an Ethereum Merge testnet, since the real update was scheduled for September 2022 [129] and our tests took place in April 2022). With this comparison, we want to answer two questions, the first being if these upgrades solve the scalability problems revealed by our first benchmark, and the second, how these two solutions compare to each other. This is interesting because this analysis will assess the relevancy of the layer 2 Polygon solution in face of the upcoming Ethereum Merge. In this comparison, since all chains are Ethereum-Virtual-Machine (EVM) compatible, the Solidity contract and much of the infrastructure presented above is reusable, which renders our comparison as fair as possible.

### 5.4.2 Development and tests

As this benchmark focuses on EVM-compatible blockchains, it was possible to reuse our VCG contract for all tests, as well as the infrastructure. This facilitated enormously the execution of the tests and data gathering.

#### 5.4.2.1 Development

In this benchmark study, the contract used is relatively close to the one used in the Ethereum and Tezos benchmarks, with a few touch-ups and updates, mainly concerning the use of the Solidity function "require", one of Solidity's state-reverting exceptions used to handle errors. We used it to assert the proper phase of the VCG for search auction, which previously was made with a series of conditional statements, to keep the code close to Tezos.

#### 5.4.2.2 Infrastructure

Our test infrastructure was kept very similar to the Ethereum version given in Section 5.3.3.2. This time, all blockchains were compatible with "Truffle", "Infura" and "MetaMask", another point for the increased fairness of this benchmark comparison.

A point of distinction between our benchmark tests is, that for this experiment, we did not use our Python script; instead our test was automated in a single JavaScript file, taking advantage of the test framework integrated in Truffle.

### 5.4.3 Test protocol

The test protocol is almost the same as the previous one described in Section 5.2.3. Once again, we follow the pattern of deploying a contract, opening an auction with CTRs, bidding and finally closing the auction. This time, we only executed one run, with 3 CTRs and 10 bidders, since we thought this would be enough to provide a first-order comparison of both approaches, given the regularities observed in the first benchmark experiment.

We first executed our test on the PoW Ropsten testnet[19], considered to be our control case, given its similarity with the current Ethereum standard. Then we continued with the execution of the same test script in both Polygon's main testnet and Ethereum's Kilin. Polygon's main testnet is called Mumbai, which, the Polygon Foundation claims, "replicates the Polygon mainnet" [130]. Kilin is another Ethereum testnet [131]. At the time of our benchmark test, it gave us the best representation of how the Ethereum blockchain would behave after its update.

## 5.5 Results and discussion

We present below the performance results induced by the execution of the VCG for search benchmark on the two Ethereum upgrades mentioned above, and discuss them.

### 5.5.1 Ropsten PoW control case

Before we present the results from Polygon and Ethereum Merge, we first present the results for our control case, Ethereum's Ropsten tests.

#### 5.5.1.1 Gas consumption

Compared to our first benchmark results from Section 5.3.5, the gas consumption changed due to the changes made in the contract, as reported on Section 5.4.2.1, as well as due to possible changes in the gas calculation by Ethereum.

| Transaction | Old VCG (gas) | new VCG (gas) |
|---|---|---|
| Deployment | 976,061 | 972,473 |
| openAuction | 122,847 | 126,539 |
| First bid | 105,917 | 112,516 |
| bid | 75,917 | 78,316 |

Table 5.1: Gas usage comparison between the old and new contract versions of VCG for search.

Table 5.1 compares gas consumption for equivalent transactions. Note that `closeAuction` gas usage changes depending on the value and order of the bids, and since they were randomly generated on our first benchmark, they are, as expected, no longer comparable.

[19]As part of Ethereum Merge effort, Ropsten was updated to proof-of-stake in May 2022; the organization was updating the consensus mechanism of their testnets, as a rehearsal for the Ethereum mainnet merge. Our test was executed on April 2022, one month before the update, so the network was still adopting PoW. As of today, July 2022, the testnet has been deprecated. This is one additional example of the challenges that one has to address when doing research on blockchain technologies.

#### 5.5.1.2 Ropsten's execution time

Once again, we took the "wall clock" approach for our time measurement, but here, we adopted a new technique to compute times, namely we took into account the block number of the block where each transaction was added to the blockchain; this way, we can tell how many blocks each transaction had to wait to be treated. And the ratio of block time vs. number of blocks should give us a result closer to the block time as advertised by the chain.

| Transaction | Time (ms) | Block number | **N.** of blocks |
|---|---|---|---|
| openAuction | 21,930 | 12216730 | 1 |
| bid | 33,016 | 12216731 | 1 |
| bid | 39,994 | 12216732 | 1 |
| bid | 36,064 | 12216733 | 1 |
| bid | 6,915 | 12216734 | 1 |
| bid | 13,895 | 12216735 | 1 |
| bid | 8,905 | 12216736 | 1 |
| bid | 5,932 | 12216737 | 1 |
| bid | 13,969 | 12216738 | 1 |
| bid | 35,888 | 12216741 | 3 |
| bid | 33,906 | 12216742 | 1 |
| closeAuction | 6,888 | 12216743 | 1 |

Table 5.2: Ropsten time of transaction with block number and number of blocks for an auction with 3 CTRs and 10 bidders.

The wall-clock time average measured via our Truffle tests is $21.44 \pm 13.44$ s. In Table 5.2, we provide the results of a Ropsten test taking in account the block number in which each transaction was added to the chain; we see an average of $19.44 \pm 12.87$ s. The results, even when normalized by block number, are still a far cry compared to the values announced by the Ethereum foundation of 12 to 14 seconds of average block time [132].

### 5.5.2 Gas usage for EVM-compatible contracts

The concept of gas was introduced by Ethereum, as a "unit that measures the amount of computational effort required to execute specific operations" [133]. The specific operations aforementioned are the EVM opcodes [34], with each being associated with a gas cost [134]. In the EVM, gas consumption is computed as the sum of the associated gas costs of all the EVM opcodes executed during a transaction. Since for our benchmark we are executing the same test script in the same contract in all blockchains, we expected for the gas used to be the same across the board.

As seen from Table 5.3, this prediction for gas used is met. It is worth mentioning that, while

| Transaction | Ropsten (gas) | Matic (gas) | Kiln (gas) |
|---|---|---|---|
| deployment | 972,473 | 972,473 | 972,473 |
| openAuction | 126,539 | 126,539 | 126,539 |
| First bid | 112,516 | 112,516 | 112,516 |
| bid | 78,316 | 78,316 | 78,316 |
| closeAuction | 193,537 | 193,537 | 193,537 |

Table 5.3: EVM-compatible gas usage by transaction.

block explorers such as Etherscan[20] list transaction fees in their native token currency, some chose to list it in dollars as well, though if it is in a testnet, it is customary to set the price as $\$0$. Contrastingly, Mumbai's Polygonscan[21] lists the transaction fees in dollars, in accordance with the Polygon token price. This is most likely a strategy to highlight Polygon's economical advantages over Ethereum. We analyse these prices in Section 5.5.3.

### 5.5.3 Polygon PoS

Polygon PoS, in its origin, was an independent blockchain, named Matic [89]; it re-branded itself in February 2021 [135] to offer scaling solutions, mainly to Ethereum. Polygon PoS is Polygon proof-of-stake chain, which runs in parallel to Ethereum's mainnet, and, supposedly, offers a cheaper option for transaction executions, while relying on the safety and trust of Ethereum.

#### 5.5.3.1 Gas and transaction fees

Since the gas consumption is the same in the different blockchains for this benchmarking study (as seen in Section 5.5.2), we focus only here on a monetary comparison. To build the comparison summarized in Table 5.4, we first verified the validity of Mumbai's Polygonscan prices in USD, by comparing them to our own estimations of the transactions fees in dollars. For our own calculations, we used Polygon mainnet's prices for both the Polygon coin and gas cost. The price for one Polygon coin (on Apr-24-2022) was $ 1.35, based on Yahoo! Finance's listing [22]. For the average gas price, the same day, based on Polygon's "PoS Average gas Price Chart" [23], we used 19.07 Gwei. With this setting, on average, the prices with mainnet data turn out to be 8.01 times larger than the values provided by Polygonscan. Our understanding is that Polygonscan used testnet gas prices to compute the prices in USD, which are unrealistic compared to mainnet, and thus cannot be of much use for our comparison.

[20] `etherscan.io`)
[21] `mumbai.polygonscan.com`
[22] `finance.yahoo.com`
[23] `polygonscan.com/chart/gasprice`

To compare with Ethereum's, we computed the transaction fees in USD based on the average Ethereum coin price in USD and the average gas price provided on the day of the test (Apr-25-2022) by YCharts[24].

| Transaction fees | Ethereum (USD) | Polygon (USD) |
|---|---|---|
| Deployment | 164.29 | 0.025 |
| openAuction | 21.37 | 0.0032 |
| First bid | 19 | 0.0029 |
| bid | 13.23 | 0.0020 |
| closeAuction | 32.69.8 | 0.005 |

Table 5.4: Speculative transaction prices in dollars for Ethereum and Polygon.

From our collected data, one sees that Polygon is on average 6541.45 times cheaper than Ethereum mainnet, comforting Polygon's economical value as Ethereum's layer 2 chain.

#### 5.5.3.2 Execution time and block time

From our tests, the "wall-clock" time for the execution of our transactions were on average $10.75 \pm 2.86$ s, normalized, while taking in account the block numbers, we had $9.93 \pm 0.52$ s. This quite consistent, but not as fast as Polygon mainnet, which clocks on average at 2.3 s per block [136]. Compared to Ropsten results, presented in Section 5.5.1.2, Polygon' Mumbai is faster and more uniform, reflecting the impact of the different block production procedure of Polygon (described in Section 3.5.1.1), which gains in consistency, compared to Ethereum's PoW.

### 5.5.4 Ethereum Merge

Ethereum Merge is an update of the Ethereum network that had as main goal the introduction of PoS to the network. At the time of writing, the so-called "merge" was scheduled for September 19 [129][25], though during our experiments, Ethereum was updating its testnets with the merge. Which gave us the opportunity to experiment with the new chain ahead of time. Kiln, a merge testnet provided by Ethereum, that was intended to function as Ethereum post merge, gave us the opportunity to examine its behaviour with our VCG test.

#### 5.5.4.1 Gas and transaction fees

Once again, the gas used by the transactions are the same (see Section 5.5.2), though this is a good opportunity to explore Ethereum's EIP-1559 [137] that became active in Ethereum's networks

[24] `ycharts.com/indicators/ethereum_price`,`charts.com/indicators/ethereum_average_gas_price`

[25] The merge update took place on September 15, 2022 [6]

on August 2022. EIP-1559 intends to overhaul Ethereum's transaction fee, which has become forbiddingly expensive, as we could attest in our first benchmark (section 5.3.6). As explained in Section 3.4.4.2, gas price in EIP-1559 is now divided in a Base Fee and a Max Priority Fee, with a price cap specified by Max Fee Per Gas.

We used gas prices estimations from Eth Gas Station[26] for our comparison. For the time and day of our analysis (July 24th 2022, 11:40am CET), Eth Gas Station lists Base Fee as 5 Gwei ($0.33/Transfer) and Max Priority Fee as 2 Gwei ($0.13/Transfer). Eth Gas Station also lists the "legacy" gas price, which at the time of this comparison is 30 Gwei. For our comparison, we used the ETH price of 1,603.49 USD. As we are able to attest by the Table 5.5, EIP-1559 prices are in average $23.33\%$ of the legacy Ethereum price. The benefits of the London fork are clear, and have been reported in studies such as "Empirical Analysis of EIP-1559: Transaction Fees, Waiting Times, and Consensus Security" [138].

| Transaction fee | Legacy gas (USD) | EIP-1559 (USD) |
|---|---|---|
| Deployment | 46.78 | 10.91 |
| openAuction | 6.08 | 1.42 |
| First bid | 5.41 | 1.26 |
| bid | 3.76 | 0.88 |
| closeAuction | 9.31 | 2.17 |

Table 5.5: Transaction fees in dollars: EIP-1559 versus legacy gas.

#### 5.5.4.2 Execution time and block time

From our test's measurements, the "wall-clock" execution time was on average $21.27 \pm 8.02$ s, and taking into account the block numbers, we measured $15.11 \pm 5.52$ s, which is very close to the advertised 15 seconds per block. As with Polygon, Klin's block production is faster and more uniform than Ethereum's PoW-based one.

### 5.5.5 Polygon versus Ethereum Merge discussion

Both Polygon and Ethereum Merge were conceived to improve (and capitalize, in Polygon's case) on the struggles that a PoW consensus inflicted on Ethereum, suffering as it approached its scalable limits in between the end of 2020 and the first quarter of 2021, as shown by our first benchmark study and Figure 5.5, which reflects the unacceptably high prices of Ethereum at the time. Through our data gathered by our VCG tests, we were able to confirm the advantages of both Polygon and Ethereum Merge over the current PoW-based Ethereum. Not only both PoS-based chains are faster

[26] `ethgasstation.info`

and more consistent in their block production, but, in the case of Polygon, their transactions are cheaper.

The more interesting discussion here, looking at the future of blockchain technology, is a comparison between Polygon and Ethereum Merge. With the recent update of Ethereum's mainnet (which took place as we were closing this document), will Polygon's role as an Ethereum layer 2 still be relevant? From our experiments, we collected the data in Table 5.6, reflecting our results in this comparison. Based on our data, Polygon has a clear advantage, both economically and in terms of speed.

In terms of block time, though Polygon's Mumbai is shown to be relatively close to Kiln, we know that Polygon's mainnet aims at a 2.3 seconds per block, which is faster than Ethereum Merge ever intended to be, 12 seconds being their target block time (see Section 3.5.2.1).

With regards to monetary value, at the time of our analysis, Polygon has been shown to be be 1862,60 times cheaper than Ethereum's legacy approach to transaction fee, and 434,60 times cheaper than the transaction fees computed according to EIP-1559. Note though that, recently, in March 2022, a drop in Ethereum's price in addition to a loss of interest in NFTs made Ethereum's gas made it fall lower than Polygon's [139].

It should be pointed out that for the time being, even after the merge, Ethereum is relying on layer 2 solutions to help escalate the Ethereum network, before their next sharding update, that will focus on the chain's scalability [140].

In conclusion, Polygon PoS will probably still be relevant after the Merge, for it is speed and affordable price, though it is not sure that this will be the case forever, since price is always related to popularity, and a shift in traffic to Polygon can always tip the scale. It's also important to notice the workload involved in integrating Polygon to Ethereum, with bridges.

| | Polygon | Ethereum Merge |
|---|---|---|
| Price per gas unity (USD) | 0.00000002 | 0.00001 |
| average block-time (s) | 9.93±0.52 | 15.11±5.52 |

Table 5.6: Comparison between Polygon and Ethereum Merge.

## 5.6 Discussion about the impact of performance issues on the VCG mechanism

As stated in our original goals for these benchmark studies, our second objective with our tests was to assess the implications of a public ledger implementation on the VCG for search auction mechanism. We divide our analysis of the implementation impacts in 2 parts, the first being focused on the practical aspects of VCG, related to the way VCG for search is used in the industry today.

Remembering the usage of VCG for search in the industry (as presented Section 4.2.1.1), VCG for search auctions happen in parallel to search queries, which take a couple of milliseconds.

Sponsored search auctions can moreover involve thousands of bidders in a single auction[27]. These characteristics, of high speed with a high number of participants, are not compatible with the data obtained in our benchmark tests. The auction speed in our implementation is mostly dependent on the execution time of `closeAuction`, which is intrinsically related to a chain's block time. In our benchmark comparison, the fastest block time we encountered occured on Polygon's mainnet, with an average of 2.3 seconds per block (Section 5.5.3.2). This value, though low compared to other protocols, is clearly not low enough for the current applications of VCG for search.

Concerning the number of participants, we were unable, as stated in Section 5.3.3.3, to close an auction with 50 participants in Ethereum, and even though Tezos was able to accommodate more participants, their number will always be associated to a block's gas limit, which is a stringent bound. Smart contracts in public blockchains were conceived for coin exchange and simple computations, and though it is possible to execute our VCG algorithm for a certain number of participants, accommodating the number of bidders as required by the industry would be impossible.

The second point is the impact of our implementation on the VCG for search auction mechanism. The practical aspects of VCG for search, though relevant for VCG in the industry, are not specified in the description of VCG for search auction mechanism (see Section 4.2.2). From our analysis, the main negative impact of a blockchain implementation of VCG for search is the ensuing loss of privacy. Public blockchains, as the ones used in our benchmark study, are characterized by their transparency, an important trait for ensuring their security and trust (a big plus compared to traditional auction mechanisms that are most of the time "black boxes" for bidders).

For VCG, transparency means the loss of sealed bids (refer to Section 4.1.1), an integral part of the mechanism. In our VCG contract, bids can be traced back to their originating wallets, and the bid values are available to be read by anyone interested. This lack of privacy could affect the bidders' incentives to bid truthfully, a key issue we analyse in Chapter 6.

The idea of implementing VCG as a smart contract, though initially appealing due to the transparency of its data processing, had some less positive implications. While the practical limitations on time, number of participants and possibly price could be overcome if we chose a more suited scenario for a more general VCG-based mechanism outside of sponsored search (for instance, for high-stake auctions such as radio-wave spectrum allocation for mobile operators or high-value manufacturing contracts), the impacts on bidder privacy cannot be ignored. In the next chapter, we explore some possible solutions for keeping the bids private.

[27]This approximative number of participants in a sponsored search auction was given to us by a member of the research team of Criteo [141].

**CHAPTER 6**

# PRIVACY IN SMART CONTRACT AUCTIONS

*La transparence est un élément clé pour favoriser la sécurité des blockchains publiques ; dans ces systèmes, toutes les données, y compris les codes de programme, sont facilement accessibles à toute personne intéressée, ce qui facilite la détection des fraudes ou des informations altérées. Cependant, cette transparence, bien qu'elle soit une propriété fondamentale des blockchains publiques, constitue un handicap sévère lorsqu'on souhaite maintenir une partie de ces informations secrète.*

*Dans notre analyse comparative de l'exécution de VCG pour la recherche sur une blockchain publique, dans le chapitre 5, nous avons déjà mentionné que notre implémentation de VCG pour la recherche était affectée par cette problématique de transparence et le manque de confidentialité qui en découle. Pour que l'enchère VCG fonctionne comme initialement prévu (voir la section 4.2), un certain degré de confidentialité est nécessaire. Le mécanisme est une enchère scellée où dire la vérité est une stratégie dominante, et demander aux enchérisseurs d'exprimer publiquement leur vraies préférences sur un registre distribué public constituerait un obstacle majeur à l'adoption de VCG ou de tout autre mécanisme d'enchère scellée dans une implémentation blockchain.*

*Dans ce chapitre, nous abordons la notion de confidentialité dans la section 6.1 et comment elle se rapporte au cas spécifique de l'algorithme VCG pour la recherche. Les sections 6.2 et 6.3 présentent notre analyse de la manière dont l'industrie aborde les problèmes de confidentialité sur les blockchains publiques, et enfin, la section 6.4 présente des versions mises à jour de notre implémentation de VCG pour la recherche qui parviennent à préserver, au moins en partie, la confidentialité des enchérisseurs. Le chapitre se termine par une conclusion et une discussion de nos résultats dans la section 6.5.*

Transparency is a key enabler of the security of public blockchains; in these systems, all data, including program codes, are readily available for anyone interested, making it easy to spot frauds or altered information. However, this transparency, though a fundamental property of public blockchains, is a severe handicap when one wishes to maintain some part of this information secretive.

In our benchmark analysis of public-blockchain execution of VCG for search in Chapter 5, we already mentioned that our VCG for search implementation was affected by this transparency issue and the consequential lack of privacy. For the VCG search auction to behave as initially intended (see Section 4.2), a certain amount of privacy is necessary; the mechanism is a truthful

sealed-bid auction, and calling for bidders to publicly express their private evaluation on a public ledger would be a big setback for the adoption of VCG or any other sealed-bid auction mechanism in a blockchain implementation.

In this chapter, we go through the notion of privacy in Section 6.1 and how it relates to the specific case of the VCG for search algorithm. Section 6.2 and 6.3 present our analysis of how the industry approaches privacy issues on public blockchains and, finally, Section 6.4 presents updated versions of our VCG for search implementations that manage to preserve, at least, part of the bidder's privacy. The chapter ends with a conclusion and discussion of our findings, in Section 6.5

## 6.1 Privacy

Privacy is a complex and subjective notion. In the words of the American jurist and economist Richard Posner, "*much ink has been spilled in trying to clarify the elusive and ill-defined concept of privacy. I will sidestep the definitional problem by simply noting that one aspect of privacy is the withholding or concealment of information*." [142]. We follow the same philosophy in this document, adopting the notion that privacy is the ability to withhold any information that one desires, and having the power to decide what to share, with whom and when.

Of course, one can envision a whole continuum of privacy policies, depending on the amount, nature or scope of the privacy-endowed data. In our examination of VCG for search as a smart contract, we are interested is what crucial information need to be withheld for the auction to retain its characteristics, and how the lack of privacy originating from a public blockchain would affect its behavior.

### 6.1.1 Privacy in truthful sealed bid auctions

Regarding sealed-bid auctions, the main privacy concern is the secrecy of bids, whose information need to be concealed from other bidders and shared exclusively with the auctioneer. One could also argue that the information of who is participating in such an auction is also a potentially sensitive information.

Truthful sealed-bid auctions such as VCG, since they promote truthful bidding by design, face an extra challenge concerning privacy. A truth-revealing strategy implicitly forces bidders to reveal valuable information [71]. Rothkopf, Teisberg and Kahn (1990) [75] argue that, in Vickrey's conception of the auction model [66], Vickrey considered the auction as an isolated event, which doesn't correspond to the reality of an actual auction, which is always part of a larger stream of commerce activities, in which a participant's true value being reveled can negatively impact future negotiations. Rothkopf et al. cite auctions for public constructions, which are followed by extensive negotiations, as a good example of these long-term issues: "*Most winning bidders will negotiate for financing, construction, government permits, and labor. In these negotiations, a winner bidder would be in disadvantage if other parties knew its true cost.*" [75]. Rothkopf

argues that participants would be reluctant to participate in a truth-revealing auctions, unless in the limited number of cases in which participants wouldn't be afraid of privacy loss such as auctions of collector seals by hobbyists [1].

Interestingly, VCG for search is a particular case. It is a truthful sealed-bid auction, though one could argue that the results are partially public, since once the ads are printed on a search result page, the winners and potentially the ranking of their bids are at least partially known. Bidders that lost the auction can keep their privacy, but winners are on display. To this extent, bidders can know if they were over-bidden, and in some cases if they themselves overbid the other participants, which could lead the winner to re-evaluate their choice of true value [2].

As we can assess, privacy in auctions isn't relevant just in the context of smart contracts, in the next section we explore some proposals for privacy preservation on centralized auction system.

### 6.1.2 Proposals for secure sealed-bid auctions

Design of secure auctions has been an actively researched topic since the 1990s [144]. Here we present the main studies on how the securing of sealed-bid auctions is being approached in centralized systems. The focus of the studies presented here isn't the Vickrey-Clarke-Groves for sponsored search variant, but they nonetheless can help us to create a panorama of the different issues of privacy plaguing auctions and the techniques being used in the literature to approach them.

**Incremental bidding** In "An ascending implementation of the Vickrey-Clarke-Groves mechanism for the Licensed Shared Access" [145], Chouayak et al. introduce an incremental General VCG implementation (note this is for General VGC, not the sponsored search version) for the allocation of bandwidth, divided in identical data blocks, among Mobile Network Operators. This incremental approach, based on the clinching auction approach proposed by Lawrence M. Ausubel [146], works in rounds, as follows.

1. The auctioneer starts by offering a supply of $k$ bandwidth blocks for a certain price $P$.
2. The bidders respond with their requested number of blocks with a price $P$.
3. If the sum of the requested blocks by the bidders is larger than $k$, the auctioneer increases the price $P$ by a predetermined amount $y$. A new round then starts, with the auctioneer offering $k$ blocks with the increased price $P + y$.
4. The auction ends when the aggregated demand from bidders is lower than or equal to the $k$ offered blocks;

The incremental aspect of this approach preserves some privacy of the winners, since they don't have to reveal the entirety of their true values.

[1] Vickrey auctions have been used in philately since 1893 [143].

[2] The so-called "winner's curse".

**Cryptographic proofs** In "Secure sealed-bid online auctions using discreet cryptographic proofs" [144], the authors propose a secure sealed-bid auction system, in which neither the auctioneer nor the bidders learn the value of the losing bids. The system applies cryptographic tools, such as circuit-based cryptographic proofs (boolean circuits that represent a theorem, which is proved true if and only if the theorem holds [147]) and additively homomorphic encryption for bit commitment. The proposal is claimed to be one among the set of protocols in which the bidders decide among themselves who has the highest bid, without the participation of the auctioneer.

Homomorphic encryption is a special form of encryption which allows specific types of computations to be carried out on ciphertexts, so that when decrypted, it matches the result of the same operation being performed on the originating plaintext [148].

For example: let $Enc(x)$ denote the homomorphic encryption of $x$, and $a$ and $b$ two plaintext integers. If the homomorphic encryption is additive, which means, it preserves the homomorphic for addition mathematical operation, $Enc(a + b)$ will be equivalent to $Enc(a) + Enc(b)$.

The proposed auction, similar to the clinching auction, works as follows.

1. The bidders encrypt their bids, bit by bit and commit their encrypted bids, as part of a commit-reveal scheme (see below);

2. the auction proceeds in rounds, with the auctioneer presenting a price in each round. The price starts with a maximum price, is decreased in each turn, until a minimum price is reached.

3. The bidders respond positively or negatively to each proposed price by the auctioneer.

4. When a bidder's bid matches the current value offered by the auctioneer, the bidder is required to reveal the committed bid, for public inspection.

5. After the definition of the highest bid, the rest of the bidders publish proof certificates that their bids are lower than the winning one.

Proof certificates are data elements that contain sufficient information to recreate cryptographic proofs of a related property and verify those without any disclosure of private information. Here, bidders publish enough data (but without revealing their bids) for anyone to be able to reconstruct integer-comparison circuits and perform the computation of the AND gates, connected to input and output ports, that specify the property of interest. These circuits can be executed with the crypted committed bids as inputs, due to the additively homomorphic encryption characteristic of the encryption used.

**Multiparty computation** In the article "Efficient Privacy-Preserving Protocols for Multi-unit Auctions" [149], the authors address the possible lack of trust on the auctioneer by part of the bidders, stating that bidders might doubt the correctness of the auction outcome or might be reluctant to share their true values, since these valuations are often based on sensitive information. This

lack of privacy is viewed as a critical problem for sealed-bid auctions. To mitigate the said lack of trust, the authors propose a mechanism in which bidders jointly compute the auction outcome, independently of the auctioneer.

Bidders partake in a distributed generation of El Gamal keys. El Gamal is a public-key cryptosystem that has homomorphic-encryption characteristics. Due to these homomorphic characteristics, via multiparty computation, the bidders can jointly compute the auction results without having to ever reveal their bids. The process indeed achieves privacy, but at the cost of high computational and communication complexity.

An interesting characteristic of this approach, as well as the previous one, is that both require users to submit zero-knowledge proofs (ZKP) in order for users to prove the validity of their private computations, to prevent attacks from ill-intentioned bidders.

First introduced by Goldwasser et al. in the seminar paper "The Knowledge Complexity of Interactive Proof-Systems" [150], zero knowledge protocols are a way for a prover to demonstrate the validity of an statement (the proof) to a verifier, without revealing anything more than that the statement is true. The denominated "zero knowledge" comes from the fact that no extra knowledge can be acquired from the proof, except for it's validity.

**Discussion** These proposals demonstrate how privacy is a very important issue for auctions, even for centralized systems, with the proposals achieving different levels of bidder's privacy. The issues addressed by these approaches is however drastically different from our's. Not only is VCG for search a different auction type than the multi-unit sealed bids, with more complex pricing calculations, but we were restricted by the limitations of being in a distributed blockchain-based system. There are no means of secure or private communication and computation inside of the blockchain, since all the data is transparent. And the small gas limits of blocks restricts what can be accomplished with smart contracts, as we saw in our benchmark studies (see Chapter 5). However, the simple fact of being inside a transparent distributed system remedies some of the challenges encountered by centralized systems; for example, the bidders can trust the auctioneer, since all the auctioneer auctions are visible in the smart contract. The same could be said for the bidder's actions, which do not require ZKPs to attest the honesty of their auctions. But, of course, this significantly increased level of transparency strongly impacts what can be done when wanting to add some level of privacy.

## 6.2 Privacy in public blockchains

Public blockchains, as the ones covered in this document, are by definition transparent platforms; all the information stored into blocks is publicly available to read. This means that every transaction is public, traceable to the originator's address, and both its inputs and changes to the blockchain storage are recorded on the chain. This transparency is what ultimately guarantees the security of the network and is fundamental for verifying the soundness of its data.

Since users communicate with the blockchain via transactions, as defined in Section 3.4.3, we characterize the privacy of transactions via two properties [151]:

- **anonymity**, i.e., hiding the identities of both sender and receiver of a transaction;
- **confidentiality**, i.e., hiding the amount transferred, or in the case of smart contracts, the input and estate changes.

The blockchains covered in this document already provide some form of weak anonymity. It is not possible to link addresses to real-life people, unless we become aware of some off-chain transaction that reveals who owns the private key behind an account, for instance via the payment for some real-world service with cryptocurrencies or the purchase of cryptocurrencies on a trading platform with fiat money. We define this type of anonymity as "pseudo-anonymity", because once an user's identity is revealed, it cannot be concealed again.

On the other hand, public blockchains cannot grand any confidentiality to their transactions. As previously stated, in a public blockchain, all the information related to a transaction is available and transparent, since this is crucial for the safety and correctness of the chain's data. This lack of confidentiality is a major challenge for the widespread adoption of public blockchains. For example, blockchains as payment platforms have the inconvenience of enabling anyone in possession of a wallet's address to keep track of all its spending.

The lack of confidentiality limits the types of smart contracts that can function in a public system; two instances that would greatly benefit from the security of blockchains but are hindered by the transparency are voting systems and the one covered in this document, namely auctions. Yet, some projects intend to bring back some notion of privacy to blockchain systems; most focus on anonymity of transactions, and we explore the most promising solutions in Section 6.3.

## 6.3 Privacy solutions for public blockchain systems

As the utilization of smart contracts expands over simple token transfers, there has been big efforts in the business and research communities to try to mitigate the inherent lack of privacy in blockchains system [152] [153]; this issue is seen as a significance hindrance for the widespread adoption of blockchain-based systems [154] [155].

### 6.3.1 Existing proposals

In this section, we present some interesting approaches for bestowing some secrecy in blockchains and smart contracts. We discuss some other proposals specifically related to auctions afterwards.

**Tornado Cash** Tornado Cash [156] is a (now infamous, due to the controversy the use of this mixing software technology raised [157]) Ethereum mixer. Mixers in cryptocurrency platforms are software solutions for the "mixing", or shuffling, of crypto-coins in order to obscure the origin

of the corresponding transactions. If this can improve the anonymity of the blockchain in general, it can also, as was the case for the ban, be used for fraudulent financial activities, i.e., money laundering.

Tornado Cash uses zk-SNARK ("zero-knowledge succinct arguments of knowledge") proofs to break the link between the depositor and the withdrawer of a coin. A zk-SNARK is a proof of a certain computation; it is difficult to compute and easy to verify, "zk" stands for zero-knowledge, which means that the corresponding property can be proved without "knowledge", or full revelation of information [158]. Note that the zk-SNARK technology is growing in popularity, also being used by other systems such as Zcash [3].

Tornado Cash generates and communicates zk-SNARK proofs off-chain. So, upon deposit of some coins, a zk-SNARK is generated and communicated to the depositor via a secure channel. Later, with any different account, the zk-SNARK proof can be revealed and, if verified, the user can withdraw the coins. The usage of zero-knowledge proofs completely obscures the link between the deposit and withdraw transactions.

**BroncoVote** This proposal for a secure and private e-voting system is based on Ethereum's smart contracts. BroncoVote [159] makes use of an off-chain server for performing Paillier homomorphic encryption (PHE), for the processing of users' votes. Proposed by Paillier in 1999, Paillier homomorphic encryption is a symmetric public-key cryptography algorithm that is characterized by its additive homomorphism property.

Homomorphic encryption is alluring for smart-contract developers, because it allows users to perform computational transactions directly on hidden crypted values, and the in-chain computation can be performed without the need for decryption, efficiently hiding the input and output values. In BroncoVote, PHE is used to compute the sum of votes. Paillier homomorphic encryption is achieved through an off-chain server that voters need to communicate with before casting their votes. Unfortunately, this off-chain server approach obviously centralizes and adds vulnerability to the system.

### 6.3.2 Public blockchain auctions

Presently, most auctions occurring on public blockchains are open, first-price auctions, mostly related to the sale of NFTs. For example, CryptoKitties[4] adopts first-price, time-based, descending Dutch auctions to sell its crypto-cats. And OpenSea[5], currently the "largest web3 marketplace for NFTs and crypto collectibles" offers its uses the choice of both English and Dutch auctions to sell their assets.

Both types of auctions are first-price [6], and in this case, for a bidder to be publicly announcing

[3] `https://z.cash/`

[4] `cryptokitties.co`

[5] `opensea.io`

[6] Dutch auction is strategically equivalent to a first-price sealed-bid auction [65]

their bid is desirable. For example, in case of goods such as NFTs, keeping a public record of all the prices payed during auctions is advantageous, as a testimonial of the value of the sold good. Yet, presently, the major concern in most blockchain-based auctions are front-running attacks.

The concept of front-running attacks originated in financial markets, to refer to a stock-trading strategy used by an agent having inside knowledge of a future transaction that will affect the corresponding price substantially [160]. Front-running attacks are a well known issue since the 70s [161]. It is illegal in the EU and USA and considered unethical.

In blockchain systems, the front-running practices take a different form, due to the transparent nature of the systems. Attackers need only to observe the mempool, where pending transactions reside before being added into a block. An attacker can then analyse the (transparent, by definition) transactions in the mempool, and upon identifying a transaction that are deemed profitable by the observer, it will send the same transaction, i.e., front-run, but with a higher gas price, in order to try to be treated by a miner before the original transaction in the mempool is.

In auctions, blockchain's front-running attackers can overbid honest participants or bid first, thus assuring their ownership of the good being auctioned. To avoid front-running attacks, some dApps are opting to execute their auctions off-chain, in a decentralized manner. The Sandbox[7], a virtual Metaverse based on Ethereum, sells its valuable NFTs "LANDs" through off-chain auctions. The auction then generates a signature, which the winners can use to retrieve their "LAND" NFT.

### 6.4 Adding privacy to VCG for search

In light of the aforementioned transparency requirement of blockchain systems and the privacy needs of VCG for search, we define below the main points that we judge needed to be addressed in our implementation of VCG for search in smart contract form:

- bidder's anonymity, i.e., keeping the address of the bidder in secret;
- bidder's confidentiality, i.e., keeping the value of the bid in secret;
- records secrecy, i.e., avoiding keeping a record of the auction data (bidders, bids, winners and final prices);
- transfer secrecy, i.e., keeping secret the transfers of both goods and payment

In this section, we present three proof-of-concepts proposals for privacy enhancement on VCG for search in smart contract form, with their corresponding solidity implementation.

Our intent is to address the privacy points presented in Section 6.4 without relying on centralized solutions, while preserving some of the needed transparency in order not to compromise the

[7] `sandbox.game`

```
1 //struct for payment
2     struct Price {
3         uint256 price;
4         bool payed;
5     }
6     //mapping from winners to prices
7     mapping(uint256 => Price) internal winnersAndPrices;
```

Listing 6.1: Payment struct and map

bidders' trust on the auction. The real challenge faced here is to find the proper balance between transparency and obscurity. We also make use of our implementations to verify the feasibility of the POCs and their monetary impact, derived from the extra gas used by the contracts to preserve user's privacy.

Our proposals combine existing modern techniques to improve privacy in the blockchain; our contribution here has been to analyse their relevance to our VCG mechanism, and discuss what type of changes their use would require on our smart contract. Of course, these techniques could be used in other applications that require the same or similar level of privacy, and so our privacy-increased VCG solutions below can be seen as significant use cases for all such contracts.

### 6.4.1 Payment function

Before heading into our proposals, it's relevant to mention that our first implementation of VCG for search in Solidity, as presented in Chapter 5, is not concerned with prize payments, that is, the act of the VCG winners paying their corresponding prices at the end of the auction. To put this contract in sync with the issues at stake in this chapter, we define payments as the transfer a certain quantity of Ether (ETH) in accordance with the prices stipulated by the execution of the VCG algorithm. Considering that this chapter is interested in the privacy of the system, and regular transactions are essential for Ethereum's operations, we added a payment function to our VCG contract. The payment is based on a "Price" struct, containing the *uint256* price in coins and a *bool* to register if this payment has already been executed. A mapping entitled "winnersAndPrices" keeps track of the winners' addresses and their corresponding Price structs. Listing 6.1 represents the "Price" struct and the "winnersAndPrices" map. Through a "payment" function, winners will send ETH to the contract which will update the "payed" bool.

### 6.4.2 Commit-reveal VCG

A technique in smart contract development that is commonly used to prevent front-running attacks is the so-called "commit-reveal" scheme [162] [163], which is an adaptation of the homonymous cryptographic algorithm. This technique builds upon two parts: the first, *commit*, as it name suggests, forces participants to commit to a bid value by submitting a hashed version of said value; in

```
1 function calculateHash(uint256 value, string calldata password) external view
    returns (bytes32) {
2       return (keccak256(abi.encodePacked(value, password, msg.sender)));
3    }
```

Listing 6.2: View function for computing Keccak-256 hashes

the second, it is required for participants to *reveal* their original values. At the end, the contract can verify the validity of the revealed value by comparing it to the committed hash.

For our POC implementation of the VCG contract with a aforementioned scheme, we require bidders to commit an hashed version of their bids, using the Keccak-256 hash function, and then to reveal their bids before the results of the auction are computed. The intention of this two-phase approach is to hide bids from other potential bidders, in order prevent them to be influenced by or take advantage of the existing bids. The sequence diagram provided in Figure 6.1 illustrates the basic process of a VCG for search smart contract integrating the commit-reveal scheme for bids.

#### 6.4.2.1 Commit-reveal VCG smart contract implementation

The auction contract operates similarly to our original implementation, described in Section 5.2, the main difference deriving from the way in which bids are recorded in terms of storage and functions. This time we use two different tables to store bids:

- `bytes32[] public hashedBids`, a table of committed hashes;
- `uint256[] public bids`, the table of revealed bids.

In addition, we introduce an extra mapping, "indexes", that associates a bidder's address to its index in the "bids" and "agents" arrays.

The `hashedBids` table stores the hashed value resulting from the Keccak-256 hash value of a bidder's bid $b$, appended to a user-selected password $pwd$ and the user's address $address$, to ensure ownership. In order to compute the hash to be committed, bidders can use any implementation of Keccak-256, although we offer one in our contract, via the view function [8] `hash`, presented in Listing 6.2. We enforce here the use of one-time string password to enhance the security of our hashes; otherwise it would be potentially possible, in a relatively easy manner, to decipher a bid value via brute force [165].

The `bid` function now receives the `bytes32` result of the hashing function and stores it in the "hashedBids" table.

[8] *View* and *Pure* functions in Solidity are functions that can be executed locally, without data being shared in the blockchain. View functions don't store any data on-chain, while Pure ones don't store nor read parameters from the blockchain [164]

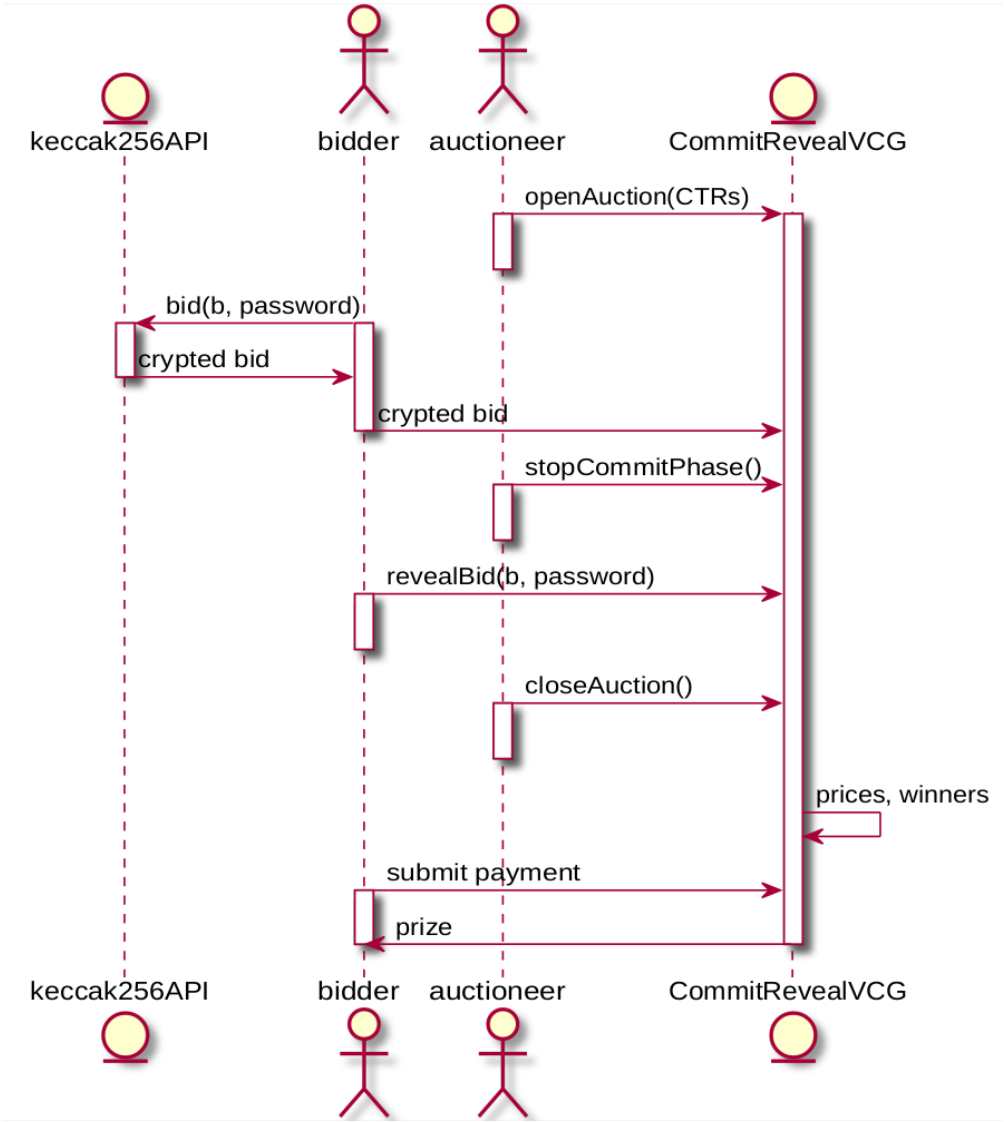


Figure 6.1: Sequence diagrams of commit reveal VCG for search smart contract

```
1  //different stages of the auction
2      enum Stages {Close, Commit, Reveal, Payment}
3      modifier atStage(Stages _stage) {
4          require(stage == _stage, "Wrong stage. Action not allowed.");
5          _;
6      }
7      function nextStage() internal {
8          stage = Stages(uint256(stage) + 1);
9      }
10     Stages public stage = Stages.Close; //start with a closed auction
```

Listing 6.3: Stage control

Note that, due to the added complexity of the commit-reveal scheme, phase control requires here more than the single *bool* "isOpen" present in our previous VCG implementation. Phase, also called "stage", control needed to be made more explicit into the contract, to regulate its flow through the different steps of the auction. The stages introduced here are: "Close", "Commit", "Reveal", and "Payment". Functions in this case have thus modifiers "atStage", which will be reverted if the function is called out of its determined stage. Listing 6.3 presents the code associated with contract stage control.

After the reception of the encrypted bids, the commit phase is over; in our implementation, the decision of phase termination is taken by the auctioneer by signalling to the contract that the contract won't accept more commitments, via a call to the "stopCommitPhase" function[9].

In the next stage, the "revealBid" function receives, from each bidder, their `uint256` bid value and *string* password as inputs; the function then checks that the hash of these inputs, once appended to the message sender's address, is equal to the previously committed encrypted value, as shown in the code-extract listing 6.4. If this requirement is met, the uncrypted bid is then stored in the "bids" table; otherwise the transaction reverts (it is the bidder's responsibility to handle errors).

Once the auctioneer decides that the reveal phase is over, it is their responsibility to compute and publish the results. In this VCG instance, the computation of prices and the creation of the "winnersAndPrices" map is achieved within the "closeAuction" function, similarly to the naive VCG implementation presented in Chapter 5.

The execution of "closeAuction" will change the contract's phase to "Payment", and now the auction winners can call the function payment, with a transaction carrying the winner's associated price in coins (ETH in this case). The contract will check if the transaction sender is in the

[9]In our implementation, due to the sequential nature of our tests, the reveal phase will naturally occur after all participants committed their bids, and thus the auctioneer knows when to call "stopCommitPhase". A more realistic implementation could be achieved with the auctioneer selecting a certain number of blocks, relative to each phase of the auction process, as a waiting time for each of the contract phases, i.e., commit and reveal. A function modifier would then compare the current block number to the previous phase's block to verify if the phase is within the number of waiting blocks and therefore active.

```
1 require(hashedBids[index] == keccak256(abi.encodePacked(value, password, msg.
    sender)),"wrong value or password");
```

Listing 6.4: Verifying the validity of a bid value and a password

"winnersAndPrices" map and if the transaction value corresponds to the price stored in the "Price" structure; if both checks are statisfied, the "payed" boolean of the Price structure is set to *true*.

A full implementation of the VCG for search smart contract, integrating the commit-reveal scheme for bids, is, with a corresponding JavaScript test file that illustrates its mode of operation, is available[10].

#### 6.4.2.2 Privacy enhancements

Commit-reveal-based protocols are commonly used to prevent others from obtaining information from transactions and front-running, based on such information, this scheme is thus efficient in our effort to hide the bids until all bidders are committed to a certain value. This way no bidders could be influenced by other's values, which was our intent with this approach.

Yet, analysing the efficiency of our approach under the scope of the parameters specified in Section 6.4 shows that our commit-reveal VCG POC is still on ineffective other grounds. Indeed, our implementation offered no enhancement in terms of bidder's anonymity and transfers secrecy; bidders still use their addresses to bid and are required to pay via a transfer to the contract.

On the other hand, it provided some partial relief in terms of bidder's confidentiality and record secrecy. Bidder's confidentiality is ensured as long as the reveal phase is not started; but, as the real values start to be stored in the contract storage, bidders can observe one another's values, and those who estimate that their chances to win the auction are too small can opt not to reveal their values, thus preserving their true values and saving on transaction fees. For the same reason, this implementation offers partial record secrecy for those bidders that opt not to reveal their values.

#### 6.4.2.3 Trade-offs

The nature of the commit-reveal process imposes a split of our bid function in two steps, commit and reveal, for a smart contract. Of course, this means a duplication on the number of transactions required for participation in the auction. This higher number of transactions increases the cost for participating, due to having to pay more often transaction fees. This also affects the latency of the whole auction.

Another possible trade-off is for users not to reveal their bids, since some privacy can be generated from this feature, as stated in the previous paragraph, this can affect the VCG prices. In an auction with $k$ slots and $n$ participants, if only the winning participants decide to reveal their bid, that is only $k$ bids are taken into account for the price calculations (as shown in Section 4.2.2), the

[10] github.com/LucasMSg/Smart-Contracts-For-Auctions-From-Experimental-Assessment-To-Privacy

last winner will have to pay a price of 0, even though the VCG for search algorithm stipulates here a non-negative value if someone else has bid below the last $k$th bid. It is up to the auctioneer to decide, in this case, if the whole transaction should be reverted or not, since the auction original specifications are, in some sense, not met here.

### 6.4.3 Commit-reveal VCG with Diffie–Hellman key exchange

Diffie–Hellman key exchange (DHKE) is a scheme for the secure cryptographic exchange of keys over a public channel. The scheme was first introduced by Whitfield Diffie and Martin Hellman in their seminal "New Directions in Cryptography" paper [166]; the so-called DH key exchange is a method for transmitting keying information over public channels. For such a scheme to work, participants need to cooperate, exchanging messages to generate a shared secret key, using modular exponentiation.

The characteristic of being designed for a public, insecure communication channel makes the DH key exchange an interesting match for public blockchain systems, specially since one of our goals is the preservation of the decentralized nature of our system. Here we propose a new POC implementation of the VCG for search smart contract; it builds upon the DH key exchange to create a shared key between bidders and the auctioneer, in an attempt to introduce some privacy to the system, in accordance to our defined privacy requirements (see 6.4).

#### 6.4.3.1 Diffie–Hellman key exchange

The algorithm for a two-party exchange of a commonly agreed-upon secret key can be summarized as follows.

1. Participants 1 and 2 agree on a prime number $P$ and a generator number; this can be done on a non-secure channel.

2. Participants choose private keys randomly; assume, participant 1 chooses $a$ and 2 chooses $b$.

3. Participants calculate $A = G^a \ mod \ P$ and $B = G^b \ mod \ P$, respectively; these are known as public keys.

4. Participants exchange their public keys.

5. Independently, participants calculate the secret key $sk = (G^B)^a \ mod \ P$, which can be shown to be the same as $sk = (G^A)^b \ mod \ P$.

A third-party observer is unable to calculate the secret key $sk$, because it is unable to infer the values of either $a$ or $b$ from the transactions. For our implementation, we used a multi-party version of DHKE; it generalizes the two-party approach to an arbitrary number of participants, with each added participant having their own copy of the private key. Once the secret key is generated, the

participants can use it for safe communication, using symmetric cryptography, in which there's only one secret key for both encryption and decryption.

#### 6.4.3.2 Multi-Party SmartDHX smart contract

Our implementation of Diffie–Hellman key exchange builds upon the Multi-Party DHKE smart contract SmartDHX proposed by Robert Muth and Florian Tschorsch [167]. With SmartDHX, the authors show that it is possible to implement Diffie–Hellman key exchange almost entirely in a smart contract form. There is, however, still need for some limited use of JavaScript for the generation of random numbers for the private keys used for the DHKE, Solidity being unequipped to do so, since its semantics are deterministic, a necessity for all nodes in the blockchain to be able to replicate the same execution.

The Multi-Party DHKE smart contract SmartDHX is composed of various subcontracts represented in the class diagram in Figure 6.2. Each participant in the key-exchange process relies upon an associated *MultipartySmartDiffieHellmanClient* contract, which inherits the DH functionalities from *SmartDiffieHellman*. In order to coordinate the exchange of messages, there is a controller contract, *MultipartySmartDiffieHellmanController*. For the modulus operation, SmartDHX makes use of the cryptographic primitives from the Solidity library[11] *SolCrypto*, in particular its relevant function *BigMod*, for modulo calculations.

The sequence diagram of Figure 6.3 presents a use case of the interactions between these contracts in order for users to execute DHKE via the SmartDHX contract and generate their secret keys. The diagram, designed from the point of view of "user1", shows their interactions with the associated MultipartySmartDiffieHellmanClient client, represented by *MSDHClient1*. User1 uses a random *seed* to generate their private keys, $a$[12] and $A = G^a \ mod \ P$ (with the generator $G$ and the prime $P$ available to the different users, since they are part of the client contract), then they can start participating in the shared key generation process. For simplicity, we assume the existence of just another participant, with their own private key, $B$; the shared key, in this case, is $AB = (B)^a \ mod \ P$, as seen in the algorithm described in Section 6.4.3.1. The presence of more participants would require performing multiple transactions of the *answer* and *generateAExtB* functions. The *answer* function stores the intermediaries keys; for example, in a 3-person key generation process, with private keys $a$, $b$ and $c$, one would have to handle the keys $AB = (B)^a \ mod \ P$, $AC = (C)^a \ mod \ P$ and $BC = (C)^b \ mod \ P$, generated by *generateAExtB(a, B)*, *generateAExtB(c, A)* and *generateAExtB(b, C)* respectively. These intermediary keys would need to be "answered", via the *answer* function, so they can be stored in the controller contract, while the global shared key $ABC = (BC)^a \ mod \ P$ will be calculated via an extra *generateAExtB(a, BC)*.

[11] `github.com/HarryR/solcrypto`)

[12] Private key a is generated via *uint256(keccak256(abi.encodePacked(seed)));* with the random seed number generated by a JavaScript program.

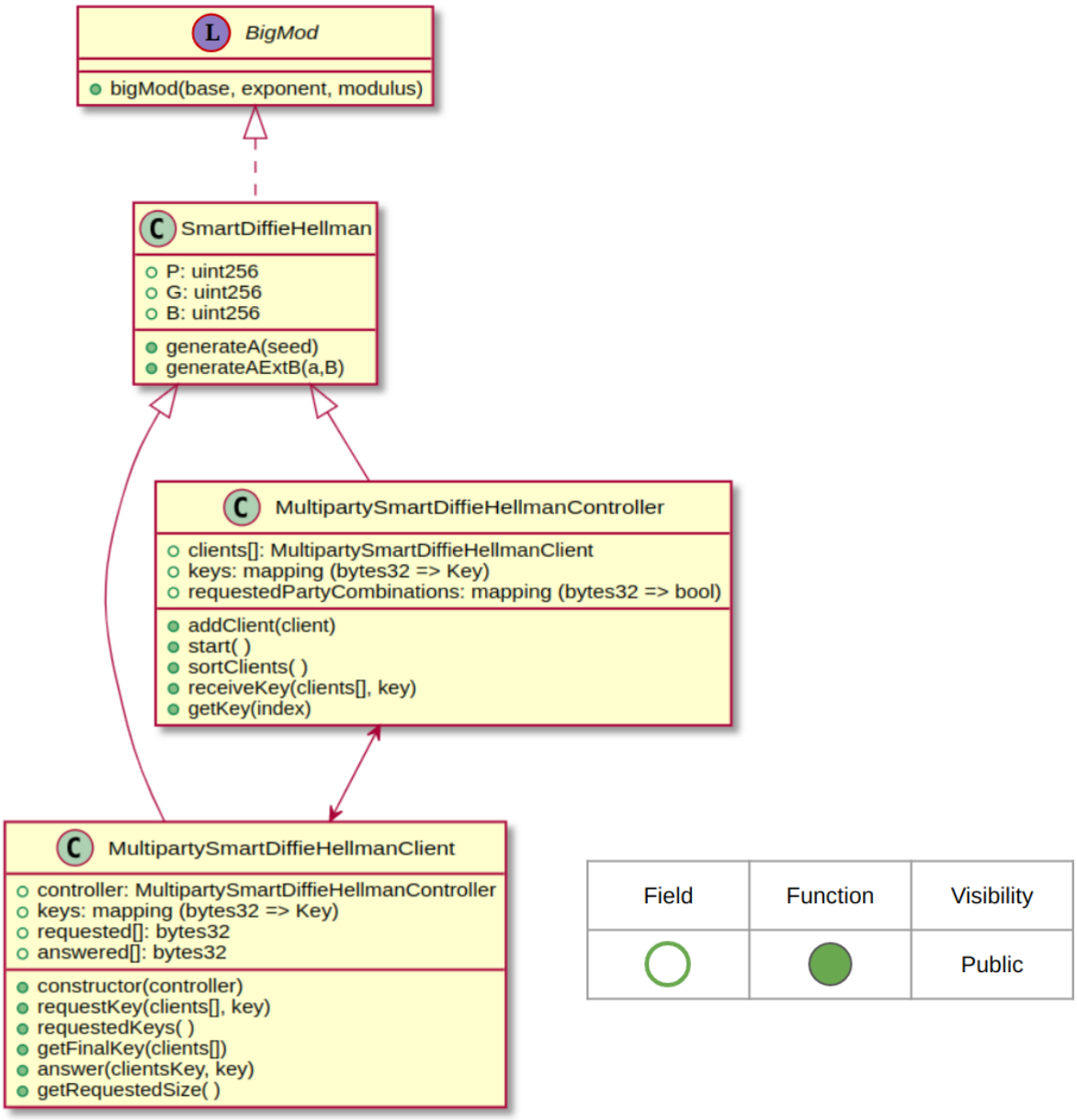


Figure 6.2: Class diagram for the SmartDHX smart contract

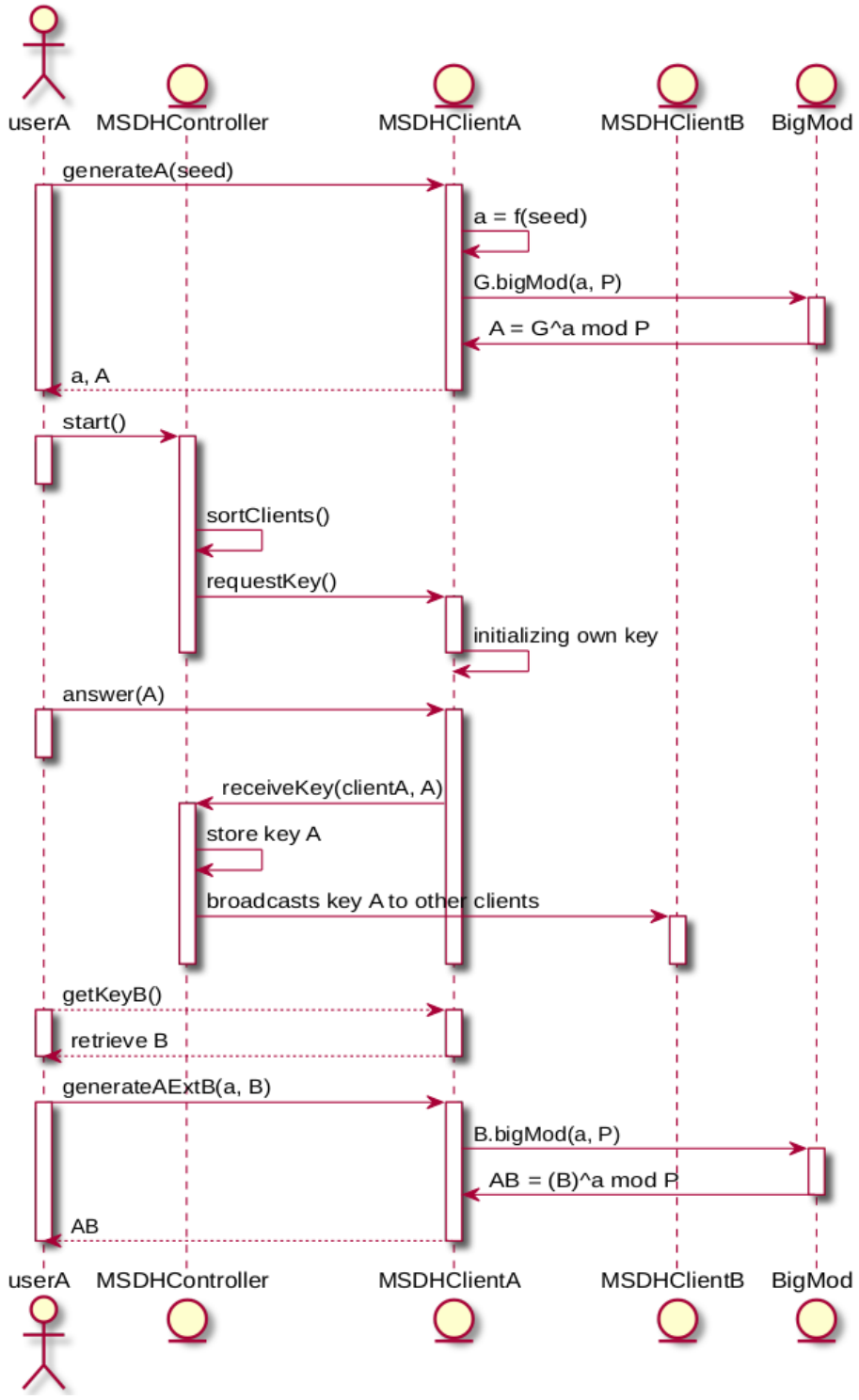


Figure 6.3: SmartDHX smart contract sequence diagram (for 2 participants)

#### 6.4.3.3 VCG for search with Diffie–Hellman key exchange smart contract

There are multiple possible approaches for integrating the Diffie–Hellman key exchange algorithm into our VCG smart contract. Our first proposal is to use the shared key introduced within the "reveal" step of the commit-reveal scheme above. However, instead of having bidders reveal their values of bids and passwords to the whole blockchain, now the bidders reveal those values encrypted via a new secret key, generated ahead of time by all the participants with SmartDHX.

The auctioneer and other key holders can then decrypt the bids and execute VCG privately, either locally or in a view function, i.e., in a manner that won't expose the bids publicly. The auctioneer then can publish the auction results, once the list of winners and corresponding prices are known, by storing these results in the smart-contract storage; the winners can then proceed with their payments.

An auction execution following this technique is represented in the sequence diagram of Figure 6.4. The auction starts with the execution of the multi-party SmartDHX, as presented above. The auctioneer and all the bidders have access to an associated *MultipartySmartDiffieHellmanClient* contract, while the auctioneer is in charge of starting the *MultipartySmartDiffieHellmanController*. Once a secret key is generated for the coming auction, a symmetric encryption system is used by all the participants to encrypt and decrypt messages with the shared key.

The VCG contract follows then much of what was established by our Commit-reveal VCG, presented in Section 6.4.2.1. The contract uses the same stages and function modifiers, to ensure the correct flow of the auction, with the same responsibility over the flow of control being delegated to the auctioneer, such as choosing to stop the commit phase and when to decide to compute the results. Also, as for the commit-reveal contract, we store the bids in two different tables, "hashedBids", to store the *bytes32* Keccak-256 hashes of the bids and passwords, and "encryptedBids", to store the revealed bids and passwords; in this instance, those are *bytes*, result of the encryption with the secret key.

Unfortunately for this first proposal, the Ethereum community strongly rejects the notion of cryptography routines being executed inside smart contracts [168], with the claim that users should make their secret keys public once they are in a blockchain transaction. With the proof-of-concept contract we just described, we nonetheless want to argue that it could be useful to allow cryptography uses through *view* functions, which can be executed locally, without the need to communicate with the blockchain network, though a centralized approach could also be used for the cryptographic part of the process.

For our POC, we used a simple one-time pad cipher [169] for the implementation of the symmetric encryption and decryption processes with the secret key. Note though that OneTimePad isn't being quite properly used here, for we reuse a key that should be used "one-time" only; yet, this serves to show the feasibility of a view-based encryption approach. The use of a one-time pad also made it necessary for the format of the bids and passwords to follow certain rules, for them to be able to be decrypted by other users without ambivalence; for our tests, we imposed that the bids need to be written in two digits, and the passwords in 8 characters, though other rules could

```
1  function decrypt(bytes memory message, bytes calldata key) public pure returns
       (string memory) {
2          bytes memory plainText = new bytes(message.length);
3          bytes memory messageinKey = abi.encode(key);
4          for (uint256 i = 0; i < (message.length); i++) {
5              plainText[i] = message[i] ^ messageinKey[i];
6          }
7          return abi.decode(plainText, (string));
8      }
```

Listing 6.5: One-time pad decipher via "decrypt" function

be implemented.

Once the auctioneer chooses to terminate the bidding process and compute the results, this has to be done in three parts. First, the values from the "encryptedBids" need to be recovered, which can be achieved in different ways, (1) by retrieving the bids from the "hashedBids" table and executing the one one-time pad cipher off-chain, or (2) by making use of the "decrypt" *view* function present in the contract. Listing 6.5 presents the code of the decrypt function, making use of the one-time pad cipher. The implementation also provides a "retrieveAllBids" function, another *view* function that decypers all bids with the secret key. It's the auctioneer's responsibility to ensure that the decrypted bids are in accordance to the committed hashes from "hashedBids".

Second, once the bids are retrieved, the auctioneer computes the results with "closeAuction", which in this instance is a *view* function that will return the array of winners and their corresponding prices. Finally, the auctioneer can publish the results and create the "winnerAndPrices" map with the results obtained from "closeAuction". This separation between the computation of the VCG results and the publication of the results is necessary to ensure that the bids are never revealed on-chain. The winners can perform now their payments via the "payment" function.

It's worth noticing that, in our tests, the auctioneer is always honest in their calculations; we didn't concern ourselves with a possible misbehavior from their part, though any owner of a private key can verify the published results by computing the auction results themselves, and refrain from executing the payment if misbehavior is encountered.

The full implementation of our commit-reveal VCG with Diffie–Hellman key exchange contract is available[13]. A JavaScript test file for the contract is also available at the same location.

#### 6.4.3.4 Privacy enhancements

We assess here the performance of our Diffie–Hellman POC, taking the parameters stipulated in Section 6.4 as guiding rules.

**bidder's anonymity** Our implementation offers no anonymity for the bidders; on the contrary, it

[13] github.com/LucasMSg/Smart-Contracts-For-Auctions-From-Experimental-Assessment-To-Privacy

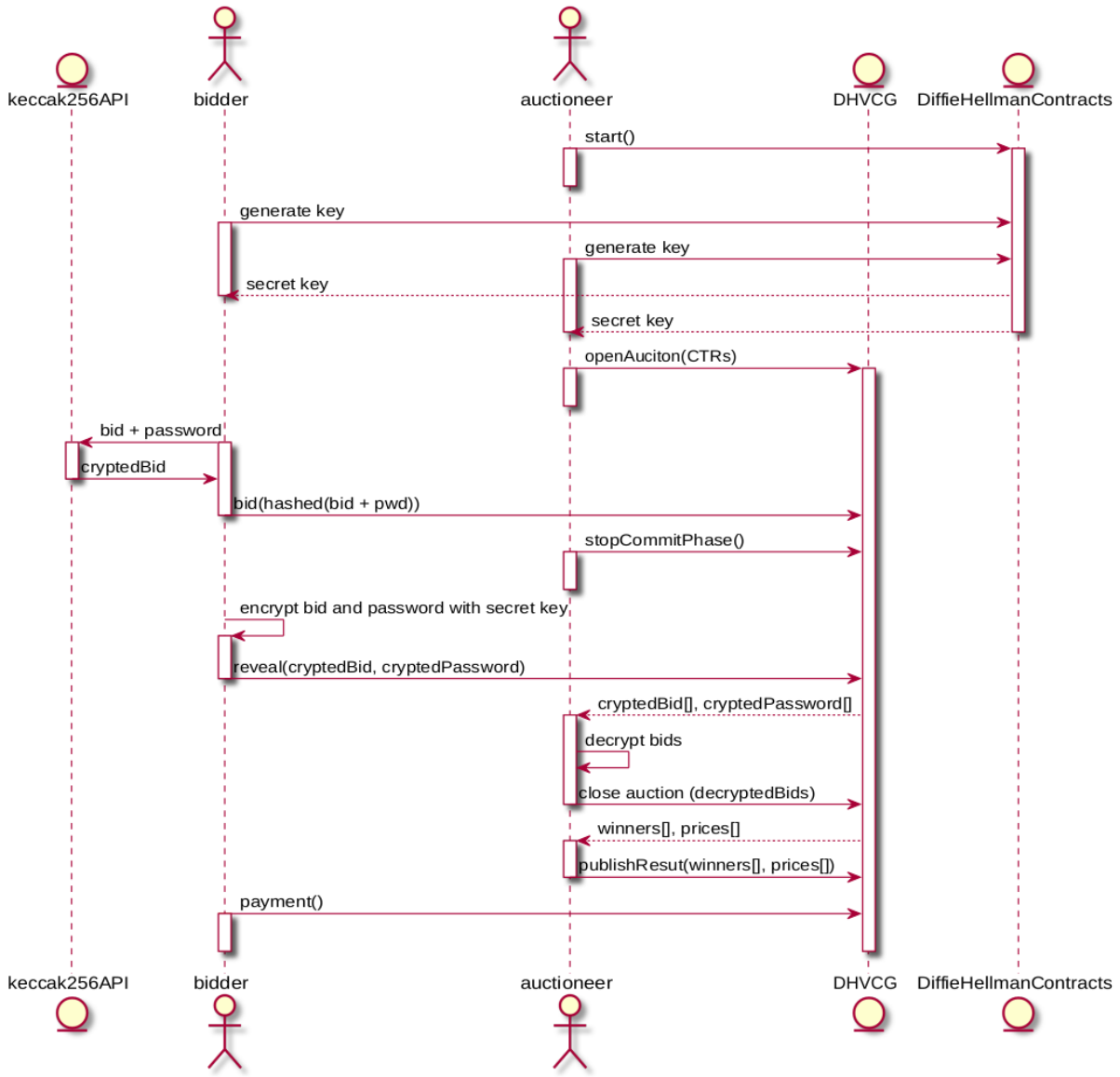


Figure 6.4: VCG with Diffie–Hellman

partially aggravates it, by requiring each participant to have an associated Diffie–Hellman client contract (MultipartySmartDiffieHellmanClient) in order to participate in the shared-key creation.

**bidder's confidentiality** In terms of bidder's confidentiality, our Diffie-Hellman VCG contract is efficient in preserving the secrecy of the bids from onlookers that do not participate in the shared-key creation, and therefore don't hold the shared key needed to decrypt the bids. Note that there is no confidentiality for the winners, since winning bidders have their addresses and prices published on-chain.

**record secrecy** Record secrecy is kept just for the bids, due to the presence of bidder's confidentiality, but the addresses of the bidders are public, similarly to the records of winners and prices.

**transfer secrecy** There is no increase in transfer secrecy with this approach: all transactions can be traced back to their originators.

#### 6.4.3.5 Trade-offs

The consequences of adopting the Multi-Party SmartDHX are, performance-wise, substantial, specially regarding the number of transactions. First, new smart contracts have to be added, one *MultipartySmartDiffieHellmanController* and one *MultipartySmartDiffieHellmanClient* for each participant, which require deployments and associated costs and time.

According to it's article [167], SmartDHX multi-Party key-generation process requires performing at least $\sum_{k=1}^{n} k$ transactions [167], $n$ being the number of participants (in this case, the number of bidders plus one auctioneer) [14].

This increase in both the number of contracts and the number of transactions induces penalty in terms of cost and latency of the auction. Note, though, it could be possible to reuse the shared key for subsequent auctions, reducing the number of transactions.

Logically, this higher number of transactions and the necessity for bidders to participate in the creation of the shared keys in addition to the auction itself increase the bidders' commitment to the auction process, in terms of participation and financially.

Finally, all the privacy enhancements derived from the secret key generated by the DHKE process would be lost if one of the bidders decide to make it public; in this case, the privacy levels return to those of commit-reveal VCG seen before. This is a serious security issue, and even more so as the number of auction participants increase.

[14] In Section 6.4.5, we see that the number of transactions is actually much higher.

### 6.4.4 Commit-reveal VCG with Diffie-Hellman and Mixer

The final proof-of-concept approach to privacy improvement in VCG for search smart contract that we introduce here is an upgrade of the VCG with Diffie–Hellman key exchange presented above. This time, we build upon the introduction of a so-called "mixer" in order to enhance confidentiality and records secrecy, obscuring winners and prices. In this approach, we're interested in hiding the source of payment for the "prizes", in the case of VCG for search case, the auctioned slots.

#### 6.4.4.1 Mixer

Mixers, also known as tumblers, are services that attempt to increase privacy in blockchain systems, by breaking the link between the origin of coins and their receiver. Mixers such as "Tornado Cash" (an infamous mixer used for money laundering in Ethereum, see Section 6.3) allow multiple users to deposit fixed amounts of coins on a shared pool, where the coins are mixed together, then users retrieve those coins with different accounts than those used for deposit.

Mixers such as Tornado Cash and MicroMix [170], use zero-knowledge proofs generated off-chain upon deposit, the prove then can be revealed for retrieving the tokens.

#### 6.4.4.2 Diffie-Hellman mixer

The idea for our implementation is loosely based on the mixer proposed by MicroMix [170]. An Mixer for transferring coins using zk-SNARKs ("Zero-Knowledge Succinct Non-Interactive Argument of Knowledge"), a type of zero-knowledge proof, that MicroMix generates off-chain. For our solution, we wanted to make use of the shared key generated by Diffie-Hellman key exchange, in a way that all calculation could be realized on-chain, which wouldn't compromise the decentralization of our system.

Our POC uses a mixer to list and receive payments from bidders, who, this way, can use different accounts than those used for bidding; this added level of indirection increases the transfers secrecy. Our solution relies on both the easiness for users to create new accounts [171] and the use of a Diffie–Hellman secret key, created by multi-party key generation between the bidders and the auctioneer.

#### 6.4.4.3 Smart contract implementation

Our implementation expands here upon our previous VCG with Diffie–Hellman version. A typical execution of VCG with DH and mixer is represented in the sequence diagram of Figure 6.5.

In this new version, as part of the "encryptBid" function, in which bidders reveal their committed bids encrypted by their secret keys, bidders will also include a wallet address (from a wallet for which they hold the private key), also encrypted with the secret shared key. The auctioneer can decipher this address, and use it, instead of the original address, when publishing the results with

"publishResults". The "payment" function only accepts payments from the associated addresses, breaking the link between bidders and payers.

The contract is similar to the Diffie–Hellman version presented in Section 6.4.3.3, with the introduction of the necessary changes for the adoption of our mixer. A new map, "encryptedAddresses", associates the index of the bidders to an encrypted value of the new address the bidder wants to use to execute the payment. These encrypted addresses will need to be decrypted by the auctioneer, and the addresses corresponding to the winners of the auction will be stored together with the winning prices by "publishResults" function.

As with the previous implementations, the full code of our VCG with Diffie–Hellman mixer contract is available[15], with an associated JavaScript test file.

#### 6.4.4.4 Privacy enhancements

The improvements on our privacy points derived from the addition of a mixer go as follows.

**bidder's anonymity** Anonymity is partially assured with the addition of a mixer, since bidders can use a one-time "throw-away" account to bid, and use their main wallet to pay and receive the auction's prizes.

**bidder's confidentiality** As with our previous proof-of-concept, the values of the bids are only known by the holders of the DH secret key. With the addition of a mixer, bidders can benefit from some confidentiality. Indeed, with no way to connect winners' addresses to bidders, it isn't clear for onlookers who won and who lost the auction, though this is still known information for the participants.

**records secrecy** In our implementation, the bids are not recorded in the blockchain, and the same goes for the winners; there is no way to connect the bidders to the winning addresses, unless the secret keys are made public, an issue we already raised.

**transfers secrecy** A mixer enhances the transfers secrecy, though the addresses of the bidders and winners can be traced, the connection between the addresses of bidders and winners is obscured by the secret key.

#### 6.4.4.5 Trade-offs

The trade-offs with this DH and mixer approach are similar to the side-effects of the Commit-reveal VCG with DHKE implementation, presented above. The security based on DHKE once again demands a relatively higher number of transactions, and client contracts for all participants. The usage of a mixer is also more demanding on the auctioneer, who here needs to decode the "encryptedAddress" and publish the new addresses with the "publishResults" function. This greater

[15] github.com/LucasMSg/Smart-Contracts-For-Auctions-From-Experimental-Assessment-To-Privacy

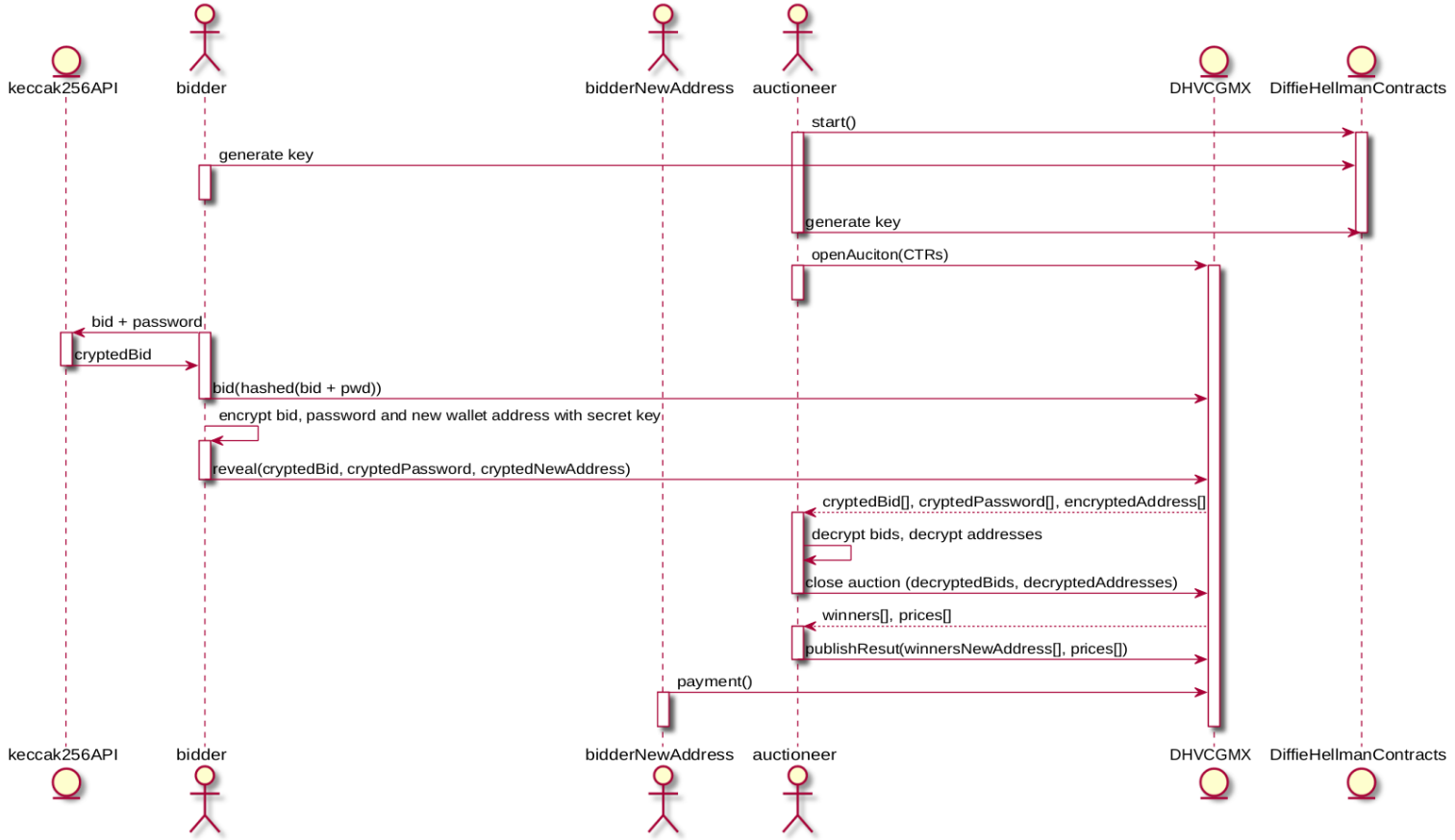


Figure 6.5: VCG with DH and mixer sequence diagram

responsibility enforced on the auctioneer enhances the centralization of the auction mechanism. Though a different implementation could be adopted in which any participant could publish the results, this would require a voting system between the participants in order to express agreement or disagreement on the resulting values.

### 6.4.5 The price of privacy

In this section, we compare and analyse our naive VCG contract with the implementations of the privacy-enhanced proof-of-concept algorithms presented in this chapter, in terms of number of transactions and gas usage, with the associated monetary cost. With this comparison, we intend to analyse the feasibility of our POCs.

As previously discussed, the "Commit-reveal VCG with Diffie–Hellman key exchange" and "Commit-reveal VCG with Diffie-Hellman and Mixer" versions perform the computation and the publication of the winners and results in two different functions, with the results being computed by a *view* function while the results are stored by "publishResults". In order to make of a fair comparison between the different proposals, we developed implementations of both the naive VCG and "Commit-reveal VCG" with a similar division; thus, their results are generated by a *view* function "closeAuction", and the results are published via "publishResults". Thus, the contracts dealt with in this comparison are: 'naive VCG" (VCG), 'naive VCG with separated results" (VCGSep), "Commit-reveal VCG" (CRVCG), "Commit-reveal VCG with separated results" (CRVCGSep), 'Commit-reveal VCG with Diffie–Hellman key exchange" (DHVCG) and "Commit-reveal VCG with Diffie-Hellman and Mixer" (MixerVCG).

The tests executed were composed of auctions between 5 bidders with the following set of bids, given as integers in some arbitrary unit: $[94, 95, 13, 17, 71]$. For the contracts that require passwords as part of their commitment, we used the following set of password strings: [*'0ho95rq4'*, *'84620e92'*, *'abw9eu56'*, *'srolni0n'*, *'c3kknvgf'*]. The auction is supposed to sell 3 items, corresponding each to different ctrs: [3, 2, 1;].

The decision to use 5 bidders in our test came from limitations with the integration of the SmartDHX implementation we used; the contracts and tests provided by the authors in their repository [167] are slow, and a higher number of bidders will, in fact, result in timeout errors in our JavaScript test files. So we decided to use 5 bidders in this approach to privacy-performance analysis, which is fast and already representative for the comparison we intend to make here. We leave more thorough testing of these issues as future work.

In contrast to the benchmark comparisons described in Chapter 5, the comparison we present here isn't focused on comparing different blockchain systems. Therefore, for these tests we didn't use testnets, opting to simply execute our tests in Truffle's testing framework, which is sufficient to obtain gas usage for our tests. Our tests were executed in Truffle version v5.3.14, with the JavaScript Solidity compiler Solc-js, version 0.8.1. All contracts and tests used for this comparison are available[16].

[16] github.com/LucasMSg/Smart-Contracts-For-Auctions-From-Experimental-Assessment-To-

#### 6.4.5.1 Gas comparison

Table 6.3 represents the gas used for our different proof-of-concept algorithms, excluding the gas used for the Diffie–Hellman key exchange, which is discussed in Section 6.4.5.2.

As previously discussed, *view* functions can be executed locally, without a transaction having to be submitted to the blockchain; therefore they don't consume gas and are considered "free" to execute. Nevertheless, in Table 6.3, we decided to nonetheless present the estimated "gas usage" of view functions, in the italics, since we consider this information is relevant to analyse the efficiency of our algorithms, since they do have a cost, even though it is a local one. The "reveal" transactions in the table relate as a whole to the "reveal" step of the commit-reveal scheme, which takes different forms in the various contract functions: "revealBid", for CRVCG and CRVCGSep, and "encryptedBidding", for both DHVCG and MixerVCG. "RetrieveAllBids" also refers to different functions within our contracts: while DHVCG and MixerVCG have a function "retrieveAllBids" that will recover the encrypted bids and decrypt them with the shared key, CRVCGSep, on the other hand, requires the auctioneer to read the bids from the "bids" table; in this case, the gas present in the table corresponds to the sum of gas for accessing all 5 bids.

From the table, one can observe the following fact: deployment costs increase with the algorithmic complexity of the contract. This should be expected, since this is directly related to the code size.

Also, the first bids are more gas consuming than the following ones, something we already noticed with the naive VCG implementation. In Section 5.3.5.2 we explained that this occurs due to the first bid setting up the storage for the arrays managed in the contract.

Note also that "closeAuction" for VCG and CRVCG are transactions, while for the other algorithms, they are views. For VCGSep and CRVCGSep, the gas sum of the views "closeAuction" and "publishResults" are similar to what happens with the "closeAuction" transactions of VCG and CRVCG, respectively.

MixerVCG has a different style of payment, since we make use of the mixer; the winners addresses are stored differently, thus the gas required for performing payment differ for each item.

#### 6.4.5.2 SmartDHX

We use the SmartDHX implementation [167] for the Diffie–Hellman key exchange necessary for DHVCG and MixerVCG. In practice, we followed the JavaScript test for Multi-party SmartDHX "2_TestMultiPartyDHX.js", made available by the authors[17], though, for our test runs, as previously explained, we used 6 participants, or clients, (5 bidders and an auctioneer) instead of the 5 used in the original SmartDHX test.

The number of transactions performed at run time by our test are the following:

- controller deployment, 1;

Privacy/tree/main/price%20of%20privacy

[17] github.com/robmuth/smart-dhx

| Transaction | VCG (gas) | VCGSep (gas) | CRVCG (gas) | CRVCGSep (gas) | DHVCG (gas) | MixerVCG (gas) |
|---|---|---|---|---|---|---|
| Deployment | 1887325 | 1890181 | 2600777 | 2422229 | 3147601 | 3583014 |
| openAuction | 125703 | 125703 | 128636 | 128636 | 128636 | 131010 |
| calculate hash | | | *23421* | *23443* | *23443* | *23465* |
| 1st Bid | 112907 | 112907 | 140016 | 139994 | 160454 | 160366 |
| Bid | 78707 | 78707 | 108604 | 108582 | 129042 | 128954 |
| stopCommitPhase | | | 27323 | 27323 | 27345 | 27345 |
| encryptBid | | | | | *60641* | *60685* |
| encryptAddress | | | | | | *36055* |
| reveal | | | 60626 | 60626 | 81351.6 | 127159.2 |
| retrieveAllBids | | | | *130533* | *267091* | *264869* |
| retrieveAddr | | | | | | *114781* |
| closeAuction | 171623 | *76471* | 172229 | *62832* | *62832* | *60544* |
| publishResults | | 103447 | | 98650 | 99405 | 187933 |
| payment 1 | 44935 | 44935 | 44973 | 45909 | 45909 | 46741 |
| payment 2 | 44935 | 44935 | 44973 | 45909 | 45909 | 48644 |
| payment 3 | 44935 | 44935 | 44973 | 45909 | 45909 | 50547 |

Table 6.1: Gas usage for the different VCG implementations. The values in *italics* represent view functions. There are 3 payments, corresponding to each of the auctioned items.

- client deployment, 6;
- controller start, 1;
- answers: 65 (Client 0, 1; Client 1, 2; Client 2, 4; Client 3, 8; Client 4, 16; and Client 5, 31).

We put these numbers in perspective in the rest of this section, noting that "Client 5" is considered the auctioneer.

In their article [167], the SmartDHX designers advertise a requirement of a total of $\sum_{k=1}^{n} k$ transactions for the execution of multi-party SmartDHX, with $n$ being the number of clients. For 6 clients, theoretically 21 transactions should be enough, but we obtained an quite different total number of 73 transactions. As it happens, in their GitHub repository[18], the authors provide an "helper" contract, with an "answerBatch" function that helps minimize the number of "answer" transactions, thus explaining the discrepancy. Of course, although the number of transactions is different, the gas used should be equivalent, since the batch function executes the same "answer" transactions in a loop. Note that the gas cost of "answerBatch" with 6 participants exceeds the gas block limit of Truffle, and thus we opted not to adopt the "helper" contract.

As explained in the SmartDHX original paper, the computation of the key exchanges before the final key calculation is optimized via some kind of distributed memoization, which renders the number of required "answer" transactions different for each participant , as we can see when looking at each client's number of "answers". Assuming that each client does execute their required "answer" transaction, the total gas used by each of them is listed in Table 6.2.

[18] github.com/robmuth/smart-dhx

| | Client 0 | Client 1 | Client 2 | Client 3 | Client 4 | Client 5 |
|---|---|---|---|---|---|---|
| Answers (gas) | 846688 | 1593725 | 2902234 | 5036715 | 7897951 | 8667425 |

Table 6.2: SmartDHX "answer" gas usage per client

As one can see, Client 5 has to execute considerably more code in terms of gas than the other participants (in fact, by more than a factor of 10). The execution of Multi-party SmartDHX necessitates the deployment of "MultipartySmartDiffieHellmanController" and the deployment of one "MultipartySmartDiffieHellmanClient" for each client. The controller deployment costs 2293839 gas, while the deployment cost varies for each client, with an average of $2332546.5 \pm 5292.36$ gas[19]. Finally, the execution of one "start" transaction by "MultipartySmartDiffieHellmanController" is required, which consumes 990450 gas.

#### 6.4.5.3 Transaction fees

For the monetary analysis of our privacy POCs, we estimated the prices of the different implementations in USD using the pricing calculations determined by EIP1559, as presented in Section 3.4.4.2 and analysed in Section 5.5.4.1. As already mentioned, EIP1559 overhauled the Ethereum fee mechanism. Since we judged it to be more representative of Ethereum's future, we adopted it for this analysis. Ethereum's (ETH) price dates from July 24th 2022, 11:40am CET, amounting 1,603.49 USD per ETH. On the same date, the gas costs from Eth Gas Station[20] are divided into a "Base Fee" of 5 Gwei and a "Max Priority Fee" of 2 Gwei.

Table 6.3 represents the different prices for the auction participants. The auctioneer is responsible for deploying the VCG smart contract and the auction control transactions, while the bidders are responsible for the bidding and committing and revealing of their bids. The payment fees indicated here are the prices winners are charged for transferring ETH during the "payment" transaction.

| | VCG (USD) | VCGSep (USD) | CRVCG (USD) | CRVCGSep (USD) | DHVCG (USD) | MixerVCG (USD) |
|---|---|---|---|---|---|---|
| auctioneer | 24.52 | 23.9 | 32.57 | 30.1 | 38.25 | 44.19 |
| 1st bidder | 1.19 | 1.19 | 2.25 | 2.25 | 2.71 | 3.22 |
| other bidders | 0.85 | 0.85 | 1.89 | 1.89 | 2.36 | 2.87 |
| payment | 0.5 | 0.54 | 0.54 | 0.51 | 0.51 | 0.54 |

Table 6.3: Fees, in US dollars, for the participants of the different POCs.

[19]The gas costs for deployment for the SmartDHX clients contract vary between them, for the client's constructor function calls "addClient" from the controller contract, which adds the new client to the controller "clients" table. In order to add a new client, the "addClient" function loops over the "clients" table, to be sure that the new client hasn't isn't there, resulting in different gas costs for clients depending on the number of clients already in the table. It's worth noting that the first client pays for the set up of the "clients" table, similarly to the higher price of the first bidders in our VCG contracts.

[20]`ethgasstation.info`

Since, for DHVCG and MixerVCG, a large part of the cost is related to the execution of the Multi-Party SmartDHX, Table 6.4 showcases the fees in USD for each client to deploy and execute the necessary transactions.

| | Cli 0 (USD) | Cli 1 (USD) | Client 2 (USD) | Cli 3 (USD) | Cli 4 (USD) | Cli 5 (USD) |
|---|---|---|---|---|---|---|
| Total prices | 35.77 | 43.99 | 58.71 | 82.7 | 114.84 | 160.37 |

Table 6.4: Fees in dollars for the 6 clients of Multi-Party SmartDHX.

As noted in the previous section, the number of "answer" transactions for the different clients is different, Client 5, i.e., the auctioneer, having the largest number of those. We judged that it would be fair for the auctioneer to bear the heaviest burden, although, of course, such a decision should be agreed upon prior to the auction launch, off-chain. Thus, in the table, Client 5 pays the fees related to both its "answer" transactions but also for the deployment of 'MultipartySmartDiffieHellmanController" and the necessary "start" transaction to this contract.

#### 6.4.5.4 Analysis

The type of items being auctioned, and more importantly their actual price, is ultimately what is going to determine the feasibility of our POCs. The prices might not justify the use of our smart contract implementation, due to its associated fees. But if one is considering auctioning, say bands of the radio-wave spectrum for new mobile operators, these fees would be minuscule compared to the future revenues.

Naive VCG and commit-reveal VCG lead to similar fees. The auctioneer fees see a 32.8% increase from VCG to CRVCG, and 25.9% from VCGSep to CRVCGSep, due to the increase in cost for the deployment and from the extra "stopCommit" transaction. On the bidders side, the commit-reveal scheme demands an extra "reveal" transaction from bidders, doubling the fees for bidders.

DHVCG and Mixer VCG, on the other hand, are considerably more costly, due to the need of executing Multi-Party SmartDHX. For the auctioneer, considering that they should bear the heavier load of transactions needed (as Client 5), the total fees increase from 23.9 USD, for VCGSep, to 198.62 USD, for DHVCG, and 204.56 USD ,for MixerVCG, an increase of 731.04% and 755.89%, respectively. Considering that the key generated by SmartDHX could be reused between different auctions, this monetary impact could be diluted over different auctions. The increase in fees for the auctioneer, disregarding the fees from the shared key exchange, in DHVCG and MixerVCG is of 60.04% and 85% by comparison to naive VCGSep.

Moving now to bidders, on average, they need to pay $67.2 \pm 32.04$ (for the clients' behavior is not uniform due to SmartDHX, as we saw above) USD in fees for the execution of Multi-party SmartDHX, amount to be added to the values presented in Table 6.3. By comparison to VCGSep, this results in an increase of $5774.8\% \pm 2592.43\%$ for the first bidder of DHVCG, and

8083.5%±3669.41% for the subsequent bidders, while, for MixerVCG, one finds 5817.64%±-2592.43%, for the first bidder, and 8143.52%±3669.41%, for the rest. Note though that, without the price induced by SmartDHX, the more reasonable increases are of 127.7% for the first bidder of DHVCG, 177.64% for the other bidders of DHVCG, 70.6% for the first bidder of MixerVCG and 237.64% for the other bidders.

As aforementioned, the values of the auctioned items and the importance of the information the origins and values of the bids might hold need to be put in perspective by the auctioneer and the possible participants before deciding to use one of the techniques introduced in this chapter. Privacy concerns might be relevant enough for the auctioneer and bidders to consider accepting to pay increased fees, and adopt one of our POCs, paying the price for the increased privacy. The techniques and analyses presented here enable an informed choice by the auction participants, and can be used to help them decide of a particular value-vs-fee trade-off for each auction.

## 6.5 Conclusion and discussion

In this section, we defined the concept of privacy, and analysed how it affects sealed-bid auctions, as an use case. We reviewed proposals for preserving privacy of auctions in centralized systems, which revealed to us that concerns about privacy aren't exclusive to decentralized auctions.

We then explored how privacy can be addressed in the context of blockchains, and showed that the transparency inherent to those systems implies the absence of secretive information.

We presented of the current approaches from the cryptosphere to preserve privacy inside of a blockchain; in most cases it involves centralized off-chain computation, which is undesirable to us, since we want to preserve trust in the whole auction process. After these initial analyses, we focused on auctions inside blockchain applications, and pointed out how mostly first-price solutions are at risk, with their biggest concern being front-running attacks. We also analysed our VCG for search auction from Chapter 5 and defined a few key privacy points that we judged being important to address. Finally, we proposed 3 new proof-of-concept implementations of VCG for search that incrementally address these privacy points.

Note, to conclude, that trying to solve the privacy problem for a VCG for search implementation is a bit a contradiction in terms. Indeed, what first enticed us to develop an auction system for decentralized applications was the trust that users can depose on them. With much of this trust deriving from the system's transparency, obscuring parts of such an auction would mean taking away from the trust. We tried, in this chapter, to find a balance between transparency and obscurity in order to maintain the user's trust, without exposing too much on the system.

What we ended up with, with our proposal in Section 6.4.4 was a an auction that can be executed without the aid of off-chain systems, manages to obscure bids from non-participants, as well as obscure the link between winners and participants. The proposal is based on secret key exchanges between all participants, which enables everyone to verify the validity of the auctions, but has the undesirable effect of giving every malignant participant the power to reveal the secret to outsiders. With the advancement of cryptographic tools and the interest in web3 and dApps, we

can hope that future developments will give us more possibilities to tackle out the privacy problem addressed here, without having to make big compromises.

CHAPTER 7

# CONCLUSION AND FUTURE WORK

We close this thesis with a recapitulation of our key findings, derived from both our benchmark comparisons and privacy assessment of the VCG for search smart contract, as well as the limitations of our research. We finish by presenting some prospects for future expansion of our research.

## 7.1 Summary of key findings and significance

We started this thesis with the intention to explore smart contracts as platforms for sealed-bid types of auctions, such as VCG for search, and our research led us through different discoveries and unveiled new and interesting research directions to expand our work. In this section, we describe the key findings achieved during this research work.

### 7.1.1 Benchmark comparison between Ethereum and Tezos

Our first benchmark analysis, presented in Chapter 5, is a comparison between the proof-of-work (PoW) Ethereum and proof-of-stake (PoS) Tezos, from the perspective of smart contracts. Our intent was to compare both chains in terms of programmability, performance and monetary cost (relative to transaction fees). For the comparison, we designed and implemented a naive VCG-for-search algorithm in both Solidity and SmartPy. Our analysis made clear the limitations of the PoW chain in face of PoS, though it also highlighted Ethereum's superiority as a platform for smart contract development, with more community support and better development tools.

This comparison was published in the article "Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos" [7] and presented in the Fourth International Symposium on Foundations and Applications of Blockchain 2021 (FAB '21).

### 7.1.2 Benchmark comparison between Ethereum's upgrades

Chapter 5 also presents a second benchmark, focused on Ethereum and two key proposals for addressing its proof-of-work related issues: the layer 2 "Polygon PoS" and Ethereum's own update, the "Merge". We believe this choice is pertinent, given the importance of these proposals in the community, and research-wise efficient, since these platforms are both EVM-compatible, which means we could reuse the same Solidity-based contract source code and infrastructure for our tests.

Once again, the comparison attested the superiority of the PoS chains in comparison to Ethereum PoW. Thus, the most compelling question raised by our benchmark data is the relevancy of layer 2 solutions in face of the Merge. Even though our benchmark revealed Polygon as the fastest and cheapest chain, even when compared to Ethereum post-Merge (Klin), this superiority is, we believe, not sufficient to declare that this layer 2 solution, and possibly others, will still be relevant as Ethereum evolves. Price is volatile, and if the Polygon PoS chain cannot offer a cheaper alternative to execute Ethereum transactions, its will be irrelevant as a layer 2 solution.

### 7.1.3 Smart-contract privacy analysis and privacy-preserving proof-of-concept proposals

Chapter 6 presents an analysis of the consequences of the lack of privacy inherent to smart contracts to a sealed-bid auction such as VCG for search. Our analysis started with a survey of the proposals for increasing the privacy for sealed-bid auctions in centralized solutions.

It became clear that privacy is a concern even for centralized systems, though this concern is different from ours, for the solutions presented focus on preserving the knowledge of the bids from the auctioneer and ensuring that the auction algorithm is executed correctly. Such concerns are not an issue in our VCG for search contract, since we are able to take advantage of the trust and transparency directly built into this type of distributed applications.

We also examined existing tools and proposals from the crypto community that have developed in order to increase privacy in smart contracts. The proposals use cryptography tools such as Paillier homomorphic encryption and zk-SNARKs, which unfortunately demand too much computation to be executed in-chain; thus the proposals surveyed opt for off-chain solutions. Since we strive to keep our system fully decentralized, we pursued a different approach.

In order to study the effectiveness of on-chain privacy solutions on our VCG contract, we determined 4 key privacy points that we judged important for our system to address: bidder's anonymity, bidder's confidentiality, records secrecy and transfer secrecy. We then presented three proof-of-concepts proposals that incrementally address our privacy points: "Commit-reveal VCG", "VCG for search with Diffie–Hellman key exchange" and "Diffie-Hellman mixer"

**Commit-reveal VCG** Our first proposal is based on the idea of adding a commit-reveal scheme on top of our VCG for search contract. The approach is efficient in ensuing that bidders are committed to their bids, and are not influenced by each other. It also provides some partial relief for bidder's confidentiality and record secrecy, as bidders can opt not to reveal their values in face of already revealed bids.

**VCG for search with Diffie–Hellman key exchange** In this proposal, auction participants generate a shared key via the "Diffie–Hellman key exchange" protocol in order to use symmetric encryption to preserve the bids' values from non-participant onlookers. The proposal is efficient in preserving the secrecy of the bids from non participants, though the prices of prizes are still revealed on-chain.

**Diffie-Helman mixer** In this improvement of our VCG for search with Diffie–Hellman key exchange approach, the link between the addresses for bids and for payments for the prizes are broken through a mixer, which takes advantage of the Diffie-Hellman-based shared key. This scheme helps to obscure the link between the addresses used for payment and those used for bids. The mixer helped thus increase the privacy in our four key privacy points, though in a limited matter, since the shared key is known by all participants.

## 7.2 Limitations

Here we present some limitations of our research work.

### 7.2.1 Working with Blockchains

A limitation for our benchmark studies was the monetary cost associated with blockchain transactions. Transactions can be quite costly, specially in Ethereum (before EIP-1559), thus it never made sense, in this research setting, to execute our tests in the corresponding mainnets of our targeted blockchains. We had to compromise in adopting a free alternative, while still keeping our tests results close to a real blockchain.

An option could have been to set up our own private network, and execute our tests in our own client, but that wouldn't replicate the competitive nature of blockchains, as we would be the sole user submitting transactions, which would reflect badly on the transaction acceptance latency. We ended up opting to use testnets for our tests. Testnets, advertised as test networks for developers, have different types of consensus, sometimes diverging from their corresponding mainnet, though we strove to use those most closely resembling their mainnet counterpart. In any case, our contribution here, in addition to the admittedly possibly somewhat biased data obtained, lies already in the protocols we designed to perform such tests.

One penalizing issue with testnets is that they often get unstable over time, as stated by Tim Beiko, in charge of Protocol Support for the Ethereum Foundation, during his interview for the podcast Epicenter on the subject of Ethereum's merge [172]: testnets, specially proof-of-work ones, with time, will present high volatility in terms of block-time values. We could indeed very clearly attest this volatility in the Ropsten testnet, hence our need to include block-time data retrieved from the the corresponding mainnets and compare those to our results.

### 7.2.2 Privacy-preserving proof-of-concept proposals

Our proof-of-concept proposals that rely on symmetric encryption are limited by our choice of symmetric-key algorithms. As discussed in Chapter 6, smart-contract developers are against the adoption of symmetric encryption, for a blockchain transaction would reveal the private key. We showed that "view" functions can be use to crypt and decrypt via symmetric-key encryption without risk of compromising the private key. For our POCs, we made use of a very simple one-time

pad cipher for the symmetric-key encryption. One-time pads, as their name state, have to be used only once, for re-use might compromise the private key used for encryption; thus it's, knowingly, being misused in our POCs. Programming a symmetric-key cypher was considered out of the scope of our research work, and since there aren't any Solidity implementation available, since they are frown upon by the community, we compromised in developing the simplest symmetric-key cipher that would fit our POC needs.

## 7.3 Opportunities for future research

Throughout this thesis, many questions and interesting leads for future expansion of this research arose. As blockchain technologies adoption's grows, our main topics of scalability and privacy become always more relevant subjects of interest, since they are a hindrance for mass adoption. Thus there are everyday new research and technologies being put in development in the field that can offer solutions to the problems we presented. Here we introduce some topics that we judge interesting to expand our research.

### 7.3.1 Benchmark study focused in scalability

As mentioned above, a worthwhile first endeavor for future research would be to perform our benchmarks on mainnets instead of the testnets we had to use; of course, the need for a somewhat significant funding for this type of real-world tests is an issue that should be handled prior to such a validation experiment.

Moreover, our benchmark studies aren't representative of the scalability issues that sometimes plague blockchains. A new study focusing on the scalability of the blockchains, in terms of transaction and block size, should be initiated on this important issue. The comparison could be done between different blockchains, akin to our Ethereum and Tezos benchmark.

Another proposal could be to conduct this scalability benchmark as a comparison of different layer 2 solutions, since they were first conceived for the purpose of escalating existing blockchains. Ethereum's next update after the Merge, "Sharding" [140], would be a great candidate for such a benchmark.

### 7.3.2 Privacy

A first interesting line of research related to our privacy work comes from one of the limitations presented in the last section: implementing and assessing the viability of a more robust symmetric-key cipher inside of a smart contract. Promising candidates are the popular Advanced Encryption Standard (AES) [173] and Chacha20 [174]. An additional study could be made in order to assess the strength of the symmetric cryptographic solution adopted.

Recent advancements in cryptography are popularising tools with very interesting properties for smart contracts and our VCG for search problem. We seriously considered in particular full

homomorphic encryption and zk-SNARKS, which look like promising tools to expand our privacy preserving POCs, before focusing on the solutions presented in this thesis. Yet, we believe that these approaches, once a bit more mature, present great opportunities in the times to come to distributed applications such as VCG.

In our research, we came across instances of adoption of additively homomorphic encryption (see Chapter 6), in which additions are allowed to be performed directly on ciphertext, with the decrypted result being the same as if the operation had been executed over plaintext. Fully homomorphic encryption (FHE) [175] allows more than addition to be performed this way; it could enable different operations and evaluations to be executed directly on ciphertext. FHE seems, once more mature, like a perfect solution for our VCG for search, since the whole VCG for search algorithm could be executed with crypted bids, and then users would need only to decrypt the processed result. Of course, costs and performance issues would need to be looked at carefully before embarking into such an approach.

Finally, zero-knowledge proofs are also quite promising for our research, more precisely zk-SNARKs (”zero-knowledge succinct arguments of knowledge”) [158]. zk-SNARKs are “succinct” for they don’t require extensive interactions between the “prover”, the actor who wants to prove some statement, and the “verifier”, the one who needs to be proven something. Zk-SNARKs proofs are fast to compute and, due to their succinct nature, are a good fit for smart contracts. For our VCG for search contract, the auctioneer could just prove to the bidders that the auction was executed correctly. During our research we experimented with ZoKrates [1], a toolbox for zk-SNARKs in Ethereum, but lack of time prevented us from digging further this promising line of research.

### 7.3.3 Limiting auctioneer’s participation in auction

VCG for search smart contracts are dependent on the participation of the auctioneer to function; it is their responsibility to submit the auctioned CRTs, as well as controlling the different phases of the auction, such as respecting the time period of bid acceptance, managing revelation (in the case of commit-reveal schemes) and deciding when to close the auction.

The auctioneers are complicated participants of auctions; one could argue that they have, in fact, the highest incentive to cheat, since they collect the prize money. And thus, there already exist several proposals for auctions that exclude the auctioneer from participating in the process (see, e.g., [144] and [149], discussed in Chapter 6).

While we believe that the transparency of smart contracts already somewhat addresses the lack of trust in the auctioneer, a key advantage of using blockchain technology for this type of applications, more could even be done to minimize the auctioneer’s participation and possible interference on the auction process. While some level participation from the auctioneer is unavoidable, such as setting up the CTRs and collecting the prize’s money, we would like to explore schemes that em-

[1] `zokrates.github.io/`

power honest bidders to even execute the auction themselves, bypassing the auctioneer altogether.

Some interesting prospects we want to explore in this context are:

- a voting mechanism among the participants to approve the payment and retrieval of items, since such a proposal would make the use of an auction process similar to multi-signature wallets [176], based on collecting bidders' signatures in order to approve the price transfers;

- a staking mechanism in which participants would need to put aside some coins in order to penalize dishonest behavior and ensure the proper execution of VCG; the staked coins could also work as an incentive for the revelation of bids during the "reveal" phase, even though an analysis of the impact this approach could have on the utility function of VCG for search would need to be performed.

## 7.4 Final thoughts

In this final section, we present some final thoughts on how we view research work in blockchain systems. The crypto-sphere, fuelled by competition and monetary success, is constantly pushing for technology advancements and protocol updates, without concern for scientific rigor to support and document their platforms. This characteristic makes researching the field, specially carrying out experiments, a constant effort of catching-up with the industry. During this thesis, the difficulty of working with fast-evolving systems became apparent in a few occasions. The first case occurred with Tezos; its self-amending properties made us have to rework tests during its protocol updates, which sometimes made our development tools faulty. This setback generated by Tezos' updates was both time-consuming and, at times, frustrating. This is an issue that designers of such systems should think about ahead of time, at least if they intend to provide a sound environment for scientifically based analysis.

Another drawback we encountered due to this inherent lack of scientific rigor relates to documentation. The chains we covered in this thesis offered documentations with varying levels of information and consistency. Some blockchains fail to maintain or provide a concise technical documentation, Bitcoin being a prime example. To compensate for such absence, there is a lot of community-made documentation, with "Bitcoin Wiki" [2] being one of the chain's main resources. The issue with community-made content is that it is seldom updated with the latest developments, which for us resulted in the need to constantly compare different documentations.

In some cases, such as Polygon, there is a main documentation provided by the related foundation, but it isn't totally transparent, with some data, such as the slash-penalty cost values and block-publishing rewards, not being revealed. One could argue that this information is available in the code of Polygon's clients, since Polygon is an open-source project, but avoiding this extra work is why documentations exists in the first place.

[2] `en.bitcoin.it`

Finally, we came to a situation analogue to Tezos', during the closing days of this document. Between September and October 2022, Ethereum Merge took place; on September 15, 2022 [6], Ethereum mainnet merged with the beacon chain. Immediately after that date, Ethereum' web site `ethereum.org/` provided an updated documentation, dropping all which was related to Ethereum PoW, which put us in a difficult position when reviewing some data for Chapter 3, specially for topics for which Ethereum's main documentation served as a resource. Luckily, Ethereum's yellow paper [36], which has a more scientific approach, preserving previous iterations of the document, was enough to provide all the information we were lacking.

We believe in the potential of blockchain systems for providing trust and security for auctions and markets, but, for these systems to be resilient enough to provide such features, scientific rigor is called for. We hope that this document will contribute on the endeavor of bringing science to blockchains, so that these systems can achieve their full potential, bringing many innovations in the years to come.

CHAPTER 8

# CONCLUSION ET TRAVAUX FUTURS

*Nous terminons cette thèse par une récapitulation de nos principales conclusions, dérivées à la fois de nos comparaisons de référence et de l'évaluation de la confidentialité du contrat intelligent VCG (Vickrey–Clarke–Groves) pour la recherche, ainsi que des limites de notre recherche. Nous terminons en présentant quelques perspectives pour l'évolution future de nos recherches.*

## 8.1 Résumé des principales conclusions et leur importance

*Nous avons entamé cette thèse avec l'intention d'explorer les contrats intelligents en tant que plateformes pour les enchères de types offres scellées, telles que VCG pour la recherche, et nos recherches nous ont conduits à différentes découvertes et ont révélé des intéressantes nouvelles directions de recherche pour élargir notre travail. Dans cette section, nous décrivons les principales conclusions obtenues au cours de ce travail de recherche.*

### 8.1.1 Comparaison des performances entre Ethereum et Tezos

*Notre première analyse comparative, présentée dans le chapitre 5, est une comparaison entre Ethereum, fondé sur le mécanisme de preuve de travail (PoW), et Tezos, fondé sur le mécanisme de preuve d'enjeu (PoS), du point de vue des contrats intelligents. Notre intention était de comparer les deux chaînes en termes de programmabilité, de performance et de coût monétaire (en ce qui concerne les frais de transaction). Pour la comparaison, nous avons conçu et mis en œuvre un algorithme VCG pour la recherche (naïf) à la fois en Solidity et en SmartPy. Notre analyse a clairement mis en évidence les limites de la chaîne PoW face à la chaîne PoS, mais elle a également souligné la supériorité d'Ethereum en tant que plateforme de développement de contrats intelligents, avec un soutien communautaire plus grand et de meilleurs outils de développement.*

*Cette comparaison a été publiée dans l'article intitulé "Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos" (Évaluation des performances des blockchains : un cas d'utilisation de contrat intelligent d'enchère VCG pour Ethereum et Tezos) [7] et présentée lors du Quatrième symposium international sur les fondements et les applications de la blockchain 2021 (Fourth International Symposium on Foundations and Applications of Blockchain, FAB '21).*

### 8.1.2 Comparaison des performances entre les mises à jour d'Ethereum

*Le chapitre 5 présente également un deuxième benchmark, axé sur Ethereum et deux propositions clés pour résoudre les problèmes liés à la preuve de travail : la couche 2 "Polygon PoS" et la mise à jour d'Ethereum, le "Merge". Nous estimons que ce choix est pertinent compte tenu de l'importance de ces propositions dans la communauté et qu'il est également efficace sur le plan de la recherche, car ces plateformes sont compatibles avec l'EVM (Ethereum Virtual Machine ou machine virtuelle Ethereum), ce qui signifie que nous pouvions réutiliser le même code source de contrat fondé sur Solidity et la même infrastructure pour nos tests.*

*Une fois de plus, la comparaison a confirmé la supériorité des chaînes PoS par rapport à la version d'Ethereum en PoW. Ainsi, la question la plus pertinente soulevée par nos données de benchmark est la pertinence des solutions de couche 2 face au "Merge". Bien que notre benchmark a révélé que Polygon était la chaîne la plus rapide et la moins chère, même par rapport à Ethereum après le Merge (Klin), cette supériorité, selon nous, n'est pas suffisante pour affirmer que cette solution de couche 2, et éventuellement autres alternatives, resteront pertinentes à mesure qu'Ethereum évolue. Le prix est volatile et si la chaîne PoS de Polygon ne peut pas offrir une alternative moins chère pour exécuter des transactions Ethereum, elle va perdre en pertinence en tant que solution de couche 2.*

### 8.1.3 Analyse de la confidentialité des contrats intelligents et propositions de preuves de concept pour préservant la confidentialité

*Le chapitre 6 présente une analyse des conséquences du manque de confidentialité inhérent aux contrats intelligents sur une enchère scellée telle que VCG pour la recherche. Notre analyse a débuté par une étude sur les propositions visant à accroître la confidentialité des enchères scellées dans les solutions centralisées.*

*Il est devenu évident que la confidentialité est une préoccupation même pour les systèmes centralisés, bien que cette préoccupation soit différente de la nôtre, car les solutions présentées se concentrent sur la préservation de la connaissance des offres de la part de l'organisateur d'enchères et sur la garantie que l'algorithme de l'enchère est correctement exécuté. De telles préoccupations ne se posent pas dans le contrat VCG pour la recherche, car nous sommes en mesure de tirer parti de la confiance et de la transparence directement intégrées à ce type d'applications distribuées.*

*Nous avons également examiné les outils et propositions existants de la communauté crypto qui ont été développés dans le but d'accroître la confidentialité des contrats intelligents. Les propositions utilisent des outils de cryptographie tels que le chiffrement homomorphique de Paillier et les zk-SNARKs, qui malheureusement demandent trop de calculs pour être exécutés sur la blockchain; ainsi, les propositions étudiées optent pour des solutions off-chain (hors chaîne). Étant donné que nous nous efforçons de maintenir notre système entièrement décentralisé, nous avons adopté une approche différente.*

*Afin d'étudier l'efficacité des solutions de confidentialité* on-chain *(sur chaîne) dans le contrat*

*VCG, nous avons déterminé 4 points clés de confidentialité que nous avons jugés importants pour notre système : l'anonymat de l'enchérisseur, la confidentialité de l'enchérisseur, la confidentialité des enregistrements et la confidentialité des transferts.*

*Nous avons ensuite présenté trois propositions de preuve de concept qui abordent progressivement ces points de confidentialité : "Commit-reveal VCG" (une sorte de mise en gage), "VCG pour la recherche avec échange de clés Diffie-Hellman" et "Mélangeur Diffie-Hellman".*

**Commit-reveal VCG** *Notre première proposition est fondée sur l'idée d'ajouter un schéma de mise en gage au contrat VCG pour la recherche. Cette approche garantit que les enchérisseurs s'engagent envers leurs offres et ne sont pas influencés les uns par les autres. Elle offre également un soulagement partiel en ce qui concerne la confidentialité des enchérisseurs et la confidentialité des enregistrements, car les enchérisseurs peuvent choisir de ne pas révéler leurs valeurs face aux offres déjà révélées.*

**VCG pour la recherche avec échange de clés Diffie-Hellman** *Dans cette proposition, les participants à l'enchère génèrent une clé partagée via le protocole "Diffie-Hellman" afin d'utiliser le chiffrement symétrique pour préserver la confidentialité des valeurs des offres vis-à-vis des spectateurs non participants. La proposition est efficace pour préserver le secret des offres vis-à-vis des non participants, bien que les prix des CTRs soient toujours révélés sur la chaîne.*

**Mélangeur Diffie-Hellman** *Dans cette amélioration de notre approche VCG pour la recherche avec échange de clés Diffie-Hellman, la liaison entre les adresses des offres et les adresses de paiement des prix est rompu grâce à un mélangeur (*mixer*), qui tire parti de la clé partagée fondée sur Diffie-Hellman. Ce schéma contribue à obscurcir le lien entre les adresses utilisées pour le paiement et celles utilisées pour les offres. Le mélangeur a ainsi permis d'augmenter la confidentialité concernant les quatre points clés de confidentialité, bien que de manière limitée, étant donné que la clé partagée est connue de tous les participants.*

## 8.2 Limitations

*Voici quelques limitations de notre travail de recherche.*

### 8.2.1 Travailler avec des blockchains

*Une limitation de nos études de référence était le coût financier associé aux transactions sur les blockchains. Les transactions pouvant être assez coûteuses, notamment sur Ethereum (avant l'EIP-1559), il n'a donc jamais été possible, dans ce cadre de recherche, d'exécuter nos tests sur les mainnets correspondants de nos blockchains ciblées. Nous avons dû faire un compromis en adoptant une alternative gratuite, tout en maintenant des résultats de tests proches d'une véritable blockchain.*

*Une option aurait été de mettre en place notre propre réseau privé et d'exécuter les tests sur notre propre client, mais cela ne reproduirait pas la nature compétitive des blockchains, car nous serions les seuls utilisateurs à soumettre des transactions, ce qui pourrait nuire à la latence d'acceptation des transactions. Nous avons finalement choisi d'utiliser des testnets pour nos tests. Les testnets, annoncés comme des réseaux de test pour les développeurs, ont différents types de consensus, parfois différents de leur mainnet correspondant, bien que nous nous soyons efforcés d'utiliser ceux qui se rapprochent le plus de leur équivalent mainnet. Quoi qu'il en soit, notre contribution ici, en plus des données admises et peut-être quelque peu biaisées obtenues, réside déjà dans la conception et la mise au point des protocoles expérimentaux que nous avons utilisés pour effectuer de tels tests.*

*Un problème pénalisant avec les testnets est qu'ils deviennent souvent instables avec le temps, comme l'a déclaré Tim Beiko, responsable du support du protocole pour la Ethereum Foundation, lors de son interview pour le podcast Epicenter sur le sujet du Merge d'Ethereum [172] : les testnets, en particulier ceux fondés sur la preuve de travail, présentent avec le temps une grande volatilité en termes de valeurs de temps de bloc. Nous avons en effet pu constater clairement cette volatilité dans le testnet Ropsten, d'où notre besoin d'inclure les données de temps de bloc récupérées à partir des mainnets correspondants et de les comparer à nos résultats.*

### 8.2.2 Propositions de preuves de concept pour la préservation de la confidentialité

*Nos propositions de preuve de concept qui reposent sur le chiffrement symétrique sont limitées par notre choix d'algorithmes à clé symétrique. Comme discuté dans le chapitre 6, les développeurs de contrats intelligents sont opposés à l'adoption du chiffrement symétrique, car une transaction sur une blockchain révélerait la clé privée. Nous avons montré que les fonctions de type 'view" peuvent être utilisées pour chiffrer et déchiffrer via un chiffrement à clé symétrique sans risque de compromettre la clé privée. Pour nos PoCs, nous avons utilisé un chiffre de Vernam très simple pour le chiffrement à clé symétrique. Les chiffres de Vernam (*one-time pad*, en anglais), doivent être utilisés une seule fois, car une réutilisation pourrait compromettre la clé privée utilisée pour le chiffrement ; nous les utilisons donc sciemment de manière inappropriée dans nos PoCs. La programmation d'un chiffrement symétrique a été considérée comme étant hors du champ d'application de notre travail de recherche, et étant donné qu'il n'existe pas d'implémentation Solidity disponible et que celle-ci est mal vue par la communauté, nous avons fait le compromis de développer le chiffre symétrique le plus simple qui correspondrait à nos besoins en PoC.*

## 8.3 Opportunités de recherche future

*Tout au long de cette thèse, de nombreuses questions et pistes intéressantes pour l'évolution future de cette recherche ont émergé. À mesure que l'adoption des technologies blockchain se développe, nos principaux sujets de scalabilité et de confidentialité deviennent des sujets d'intérêt de plus en plus pertinents, car ils constituent un obstacle à l'adoption en masse. Ainsi, de nouvelles*

*recherches et technologies sont constamment mises en développement dans le domaine, offrant des solutions aux problèmes que nous avons présentés. Nous introduisons ici quelques sujets que nous jugeons intéressants pour élargir notre recherche, répartis en deux catégories : les comparaisons futures de référence et les améliorations de la confidentialité.*

### 8.3.1 Étude de référence axée sur la scalabilité

*Comme mentionné précédemment, une première initiative intéressante pour les recherches futures consisterait à effectuer nos benchmarks sur les mainnets plutôt que sur les testnets que nous avons dû utiliser. Bien entendu, la nécessité d'un financement assez conséquent pour ce type de tests en conditions réelles est un problème qui doit être résolu avant de mener une telle expérience de validation.*

*De plus, nos études de référence ne sont pas représentatives des problèmes de scalabilité qui affectent parfois les blockchains. Une nouvelle étude axée sur la scalabilité des blockchains, en termes de taille des transactions et des blocs, devrait être lancée sur cette question importante. La comparaison pourrait être réalisée entre différentes blockchains, semblable à notre étude de référence sur Ethereum et Tezos.*

*Une autre proposition pourrait consister à réaliser ce benchmark de scalabilité en comparant différentes solutions de couche 2, étant donné qu'elles ont été initialement conçues dans le but d'augmenter les performances des blockchains existantes. La prochaine mise à jour d'Ethereum après Merge, "Sharding" [140], serait un excellent candidat pour un tel benchmark.*

### 8.3.2 Confidentialité

*Une première ligne de recherche intéressante liée à notre travail sur la confidentialité découle d'une des limitations présentées dans la dernière section : la mise en œuvre et l'évaluation de la viabilité d'un chiffrement symétrique plus robuste à l'intérieur d'un contrat intelligent. Des candidats prometteurs sont les populaire Advanced Encryption Standard (AES) [173] et Chacha20 [174]. Une étude supplémentaire pourrait être réalisée afin d'évaluer la solidité de la solution cryptographique symétrique adoptée.*

*Les avancées récentes en cryptographie popularisent des outils aux propriétés très intéressantes pour les contrats intelligents et notre problème de VCG pour la recherche. Nous avons sérieusement envisagé en particulier le chiffrement totalement homomorphe et les zk-SNARKS, qui semblent être des outils prometteurs pour étendre nos PoCs préservant la confidentialité, avant de nous concentrer sur les solutions présentées dans cette thèse. Cependant, nous croyons que ces approches, une fois un peu plus matures, offriront de grandes opportunités à l'avenir pour les applications distribuées telles que VCG.*

*Dans nos recherches, nous avons rencontré des cas d'adoption du chiffrement homomorphe additif (voir le chapitre 6), dans lequel les additions peuvent être effectuées directement sur le texte chiffré, le résultat déchiffré étant le même que si l'opération avait été effectuée sur le texte*

*en clair. Le chiffrement totalement homomorphe (FHE) [175] permet d'effectuer plus que des additions de cette manière ; il pourrait permettre l'exécution de différentes opérations et évaluations directement sur le texte chiffré. FHE semble, une fois de plus mature, être une solution parfaite pour VCG pour la recherche, car l'ensemble de l'algorithme VCG pour la recherche pourrait être exécuté avec des enchères chiffrées, et les utilisateurs n'auraient alors qu'à déchiffrer le résultat traité. Bien sûr, les coûts et les problèmes de performances devraient être examinés attentivement avant de se lancer dans une telle approche.*

*Enfin, les preuve à divulgation nulle de connaissance (*zero-knowledge proofs*) sont également très prometteuses pour nos recherches, plus précisément les zk-SNARKs ("zero-knowledge succinct arguments of knowledge") [158]. Les zk-SNARKs sont "succinctes", car elles ne nécessitent pas d'interactions étendues entre le "prouveur", l'acteur qui souhaite prouver une déclaration, et le "vérificateur", celui qui doit être convaincu de quelque chose. Les preuves zk-SNARKs sont rapides à calculer et, en raison de leur nature succincte, elles conviennent bien aux contrats intelligents. Pour le contrat VCG pour la recherche, l'organisateur de l'enchère pourrait simplement prouver aux enchérisseurs que l'enchère a été correctement exécutée. Au cours de nos recherches, nous avons expérimenté avec ZoKrates*[1]*, une boîte à outils pour les zk-SNARKs sur Ethereum, mais le manque de temps nous a empêchés d'approfondir davantage cette ligne de recherche prometteuse.*

### 8.3.3 Limitation de la participation de l'organisateur d'enchères

*Les contrats intelligents VCG pour la recherche dépendent de la participation de l'organisateur pour fonctionner ; il leur incombe de soumettre les CRTs (taux de clics) mis aux enchères, ainsi que de contrôler les différentes phases de l'enchère, telles que le respect de la période d'acceptation des offres, la gestion de la révélation (dans le cas des schémas Commit-reveal) et la décision de clôturer l'enchère.*

*La participation des organisateurs des enchères dans la enchère est compliquée; on pourrait soutenir qu'ils ont en réalité le plus grand incitatif à tricher, puisqu'ils collectent l'argent des prix. Et ainsi, il existe déjà plusieurs propositions d'enchères excluant les organizateurs de la participation au processus (voir, par exemple [144] et [149], discutés dans le chapitre 6).*

*Bien que nous croyions que la transparence des contrats intelligents contribue déjà en partie à résoudre le problème de confiance envers les organizateurs, un avantage clé de l'utilisation de la technologie blockchain pour ce type d'applications, il est possible de faire davantage pour minimiser la participation de l'organisateur d'enchères et les interférences éventuelles dans le processus d'enchères. Bien qu'une certaine participation de l'organisateur soit inévitable, comme la mise en place des règles de l'enchère et la collecte de l'argent des prix, nous souhaiterions explorer des schémas permettant aux enchérisseurs honnêtes d'exécuter eux-mêmes l'enchère, en contournant complètement l'organisateur.*

[1] `zokrates.github.io/`

*Quelques perspectives intéressantes que nous souhaitons explorer dans ce contexte sont :*

- *un mécanisme de vote parmi les participants pour approuver le paiement et la récupération des articles, car une telle proposition rendrait l'utilisation d'un processus d'enchères similaire aux portefeuilles multi-signatures [176], fondé sur la collecte des signatures des enchérisseurs afin d'approuver les transferts de prix ;*

- *un mécanisme de preuve d'enjeu dans lequel les participants devraient mettre de côté certaines pièces pour sanctionner les comportements malhonnêtes et garantir l'exécution appropriée de VCG (Vickrey-Clarke-Groves) ; les pièces mises en jeu pourraient également servir d'incitation à la révélation des offres pendant la phase de "révélation", bien qu'une analyse de l'impact que cette approche pourrait avoir sur la fonction d'utilité de VCG pour la recherche devrait être effectuée.*

## 8.4 Réflexions finales

*Dans cette dernière section, nous présentons quelques réflexions finales sur notre vision sur le travail de recherche dans les systèmes blockchain. La communauté de la crypto-sphère, alimentée par la concurrence et le succès financier, pousse constamment à des avancées technologiques et à des mises à jour de protocoles, sans se soucier de la rigueur scientifique nécessaire pour soutenir et documenter les plateformes. Cette caractéristique demande à la recherche dans ce domaine, en particulier la réalisation d'expériences, un effort constant de rattrapage par rapport à l'industrie. Au cours de cette thèse, la difficulté de travailler avec des systèmes en évolution rapide est devenue évidente à plusieurs reprises. Le premier cas s'est produit avec Tezos ; ses propriétés d'auto-amendement nous ont obligés à retravailler les tests lors de ses mises à jour de protocole, ce qui rendait parfois nos outils de développement défectueux. Ce retard causé par les mises à jour de Tezos était à la fois chronophage et parfois frustrant. Il s'agit d'un problème auquel les concepteurs de tels systèmes devraient réfléchir à l'avance, s'ils ont l'intention de fournir un environnement solide pour une analyse scientifique fondée sur des faits.*

*Un autre inconvénient que nous avons rencontré en raison de ce manque inhérent de rigueur scientifique concerne la documentation. Les chaînes que nous avons couvertes dans cette thèse offrent des documentations avec des niveaux d'information et de cohérence variables. Certaines blockchains ne parviennent pas à maintenir ou à fournir une documentation technique concise, Bitcoin étant un exemple parfait. Pour compenser cette absence, il existe de nombreuses documentations créées par la communauté, dont le Bitcoin Wiki [2] est l'une des principales ressources de la chaîne. Le problème avec le contenu créé par la communauté est qu'il est rarement mis à jour avec les derniers développements, ce qui nous a obligés à comparer constamment différentes documentations.*

[2] `en.bitcoin.it`

*Dans certains cas, comme celui de Polygon, une documentation principale est fournie par la fondation concernée, mais elle n'est pas totalement transparente, certaines données, telles que les valeurs des coûts de pénalité de slash et des récompenses de publication de bloc, ne sont pas révélées. On pourrait argumenter que ces informations sont disponibles dans le code des clients de Polygon, puisque Polygon est un projet open source, mais éviter ce travail supplémentaire est précisément la raison d'être des documentations.*

*Enfin, nous sommes arrivés à une situation similaire à celle liée à Tezos durant les derniers jours de ce document. Entre septembre et octobre 2022, Ethereum Merge a eu lieu ; le 15 septembre 2022 [6], Ethereum mainnet a fusionné avec la chaîne Beacon. Immédiatement après cette date, le site web d'Ethereum* `ethereum.org/` *a fourni une documentation mise à jour, supprimant tout ce qui était lié à Ethereum preuve de travail (PoW), ce qui nous a mis dans une situation difficile lors de l'examen de certaines données pour le chapitre 3, notamment pour les sujets pour lesquels la documentation principale d'Ethereum servait de ressource. Heureusement, le* yellow paper *d'Ethereum [36], qui adopte une approche plus scientifique en préservant les itérations précédentes du document, a suffi pour fournir toutes les informations qui nous manquaient.*

*Nous croyons au potentiel des systèmes blockchain pour fournir confiance et sécurité dans les enchères et les marchés, mais pour que ces systèmes soient suffisamment résilients pour offrir de telles fonctionnalités, il est nécessaire de faire preuve de rigueur scientifique. Nous espérons que ce document contribuera à l'effort de faire entrer la science dans les blockchains, afin que ces systèmes puissent atteindre leur plein potentiel et apporter de nombreuses innovations dans les années à venir.*

# BIBLIOGRAPHY

[1] M. De Witte and T. Ker. (2020). "Stanford economists Paul Milgrom and Robert Wilson win the Nobel in economic sciences," Stanford News, [Online]. Available: https://news.stanford.edu/2020/10/12/stanford-economists-paul-milgrom-robert-wilson-win-nobel-economic-sciences/. accessed: 11.10.2022.

[2] N. Confessore and M. Rosenberg. (2018). "Cambridge Analytica and Facebook: The Scandal and the Fallout So Far," [Online]. Available: https://nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html. accessed: 02.07.2022.

[3] C. Tabacco. (2022). "Facebook Ad Buyers Secure Class Certification in Facebook Inflated "Reach" Advertisement Case," [Online]. Available: https://lawstreetmedia.com/news/tech/facebook-ad-buyers-secure-class-certification-in-facebook-inflated-reach-advertisement-case/. accessed: 02.07.2022.

[4] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Cryptography Mailing list at https://metzdowd.com*, 2008.

[5] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2014.

[6] R. Nambiampurath. (2022). "Ethereum Finally Completes The Merge," Investopedia, [Online]. Available: https://www.investopedia.com/ethereum-completes-the-merge-6666337. accessed: 08.10.2022.

[7] L. Massoni Sguerra, P. Jouvelot, E. J. Gallego Arias, G. Memmi, and F. Coelho, "Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos," 2021. [Online]. Available: https://hal-mines-paristech.archives-ouvertes.fr/hal-03210222.

[8] JASHKOTHARI1. (2022). "Cryptography and its Types," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/cryptography-and-its-types/. accessed: 03.09.2022.

[9] (2022). “Cryptography,” IBM MQ documentation, [Online]. Available: https://www.ibm.com/docs/en/ibm-mq/7.5?topic=concepts-cryptography. accessed: 03.09.2022.

[10] (2022). “What is a cryptographic key?” Cloudflare News, [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-a-cryptographic-key/. accessed: 15.10.2022.

[11] S. Kaaru. (2022). “13 years ago, Satoshi Nakamoto sent Hal Finney 10 bitcoins in Bitcoin’s first transaction ever,” CoinGeek, [Online]. Available: https://coingeek.com/13-years-ago-satoshi-nakamoto-sent-hal-finney-10-bitcoins-in-bitcoins-first-transaction-ever/. accessed: 11.10.2022.

[12] Zehraina. (2022). “What is Ethereum Mempool?” GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/what-is-ethereum-mempool/. accessed: 01.08.2022.

[13] yashi4001. (2022). “Blockchain Merkle Trees,” GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/blockchain-merkle-trees/. accessed: 01.08.2022.

[14] G. Walker. (2020). “TXID,” Learn me a bitcoin, [Online]. Available: https://learnmeabitcoin.com/technical/txid. accessed: 01.08.2022.

[15] GR0KCHAIN. (2020). “Calculating the Merkle Root for a block,” Bitcoin Developer Network, [Online]. Available: https://bitcoindev.network/calculating-the-merkle-root-for-a-block/. accessed: 01.08.2022.

[16] (2022). “Ethereum Development Documentation,” Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/. accessed: 01.08.2022.

[17] O. Ifegwu. (2022). “Finality,” Binance, [Online]. Available: https://academy.binance.com/en/glossary/finality. accessed: 06.08.2022.

[18] N. Szabo, “Formalizing and Securing Relationships on Public Networks,” *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/548, accessed: 03.09.2022.

[19] M. Miller, “The Future of Law,” 1997, accessed: 03.09.2022.

[20] (2021). "Solidity," Ethereum, Solidity, [Online]. Available: https://docs.soliditylang.org/en/v0.8.15. accessed: 22.07.2022.

[21] (2022). "Anatomy of Smart Contracts," Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/anatomy/. accessed: 03.09.2022.

[22] (2022). "Solidity Documentation Contracts," Ethereum, [Online]. Available: https://docs.soliditylang.org/en/latest/contracts. accessed: 03.09.2022.

[23] (2022). "Anatomy of Smart Contracts," ankr, [Online]. Available: https://www.ankr.com/docs/game/extra/events-and-subscriptions/. accessed: 03.09.2022.

[24] *Ethereum Improvement Proposals*, Ethereum, 2022. [Online]. Available: https://eips.ethereum.org, accessed: 22.07.2022.

[25] M. Marcobello. (2022). "What Are EIP and ERC and How Are They Connected?" CoinDesk, [Online]. Available: https://www.coindesk.com/learn/what-are-eip-and-erc-and-how-are-they-connected/. accessed: 03.09.2022.

[26] (2022). "ERC-20 Token Standard," Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-20. accessed: 22.07.2022.

[27] F. Vogelsteller and V. Buterin. (2015). "Ethereum Improvement Proposals - EIP-20: Token Standard," Ethereum, [Online]. Available: https://eips.ethereum.org/EIPS/eip-20. accessed: 03.09.2022.

[28] (2022). "ERC-20 Non-Fungible Token Standard," Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-721. accessed: 22.07.2022.

[29] M. Sangwan. (2021). "The Ethereum Virtual Machine (EVM)," Data Science Central, [Online]. Available: https://www.datasciencecentral.com/the-ethereum-virtual-machine-evm/. accessed: 01.08.2022.

[30] J. Frankenfield. (2021). "Investopedia - Wei," Investopedia, [Online]. Available: https://investopedia.com/terms/w/wei.asp. accessed: 24.09.2022.

[31] (2022). “Deploying Smart Contracts,” Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/deploying/. accessed: 06.08.2022.

[32] (2022). “Deploying your contracts,” Hardhat, [Online]. Available: https://hardhat.org/hardhat-runner/docs/guides/deploying. accessed: 03.09.2022.

[33] (2022). “MetaMask: The crypto wallet for Defi, Web3 Dapps and NFTs,” MetaMask, [Online]. Available: https://metamask.io. accessed: 18.07.2022.

[34] (2021). “Ethereum Virtual Machine Opcodes,” Ethereum Stack Exchange, [Online]. Available: https://ethervm.io/. accessed: 25.09.2022.

[35] (2022). “Blocks - What is a block,” Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/blocks/. accessed: 06.08.2022.

[36] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger - BERLIN VERSION 8fea825 – 2022-08-22,” *Ethereum project yellow paper*, 2022, accessed: 25.09.2022.

[37] (2022). “Blocks - What is a block,” Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/mining-algorithms/ethash. accessed: 06.08.2022.

[38] B. Müller-Clostermann and T. Jonischkat, “Random Numbers - How Can We Create Randomness in Computers?” In *Algorithms Unplugged*, B. Vöcking, H. Alt, M. Dietzfelbinger, R. Reischuk, C. Scheideler, H. Vollmer, and D. Wagner, Eds. Springer Berlin Heidelberg, 2011, ISBN: 978-3-642-15328-0. [Online]. Available: https://doi.org/10.1007/978-3-642-15328-0_25, accessed: 12.11.2022.

[39] V. Buterin. (2016). “Ethereum Improvement Proposals - EIP-100: Change difficulty adjustment to target mean block time including uncles,” Ethereum, [Online]. Available: https://eips.ethereum.org/EIPS/eip-100. accessed: 06.08.2022.

[40] J. Poon and V. Buterin, “Plasma: Scalable Autonomous Smart Contracts,” 2017. [Online]. Available: https://www.plasma.io/plasma.pdf, accessed: 03.09.2022.

[41] (2022). “Cosmos SDK - The world’s most used framework for building blockchains,” Cosmos, [Online]. Available: https://v1.cosmos.network/sdk. accessed: 11.10.2022.

[42] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982. [Online]. Available: https://doi.org/10.1145/357172.357176, accessed: 15.10.2022.

[43] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," 2018. [Online]. Available: https://arxiv.org/abs/1807.04938, accessed: 12.10.2022.

[44] (2022). "Peppermint," Polygon Wiki, [Online]. Available: https://wiki.polygon.technology/docs/pos/peppermint/. accessed: 15.10.2022.

[45] (2022). "Go Ethereum - Official Go implementation of the Ethereum protocol," Ethereum, [Online]. Available: https://geth.ethereum.org/. accessed: 15.10.2022.

[46] (2022). "The Merge," Ethereum, [Online]. Available: https://ethereum.org/en/upgrades/merge/. accessed: 16.07.2022.

[47] S. Higgins. (2017). "$232 Million: Tezos Blockchain Project Finishes Record-Setting Token Sale," CoinDesk, [Online]. Available: https://www.coindesk.com/markets/2017/07/13/232-million-tezos-blockchain-project-finishes-record-setting-token-sale/. accessed: 06.08.2022.

[48] (2022). "Tezos Agora Wiki - What is baking?" Tezos, [Online]. Available: https://wiki.tezosagora.org/learn/baking. accessed: 03.09.2022.

[49] L. Aştefanoaei, P. Chambart, A. Del Pozzo, T. Rieutord, S. Tucci, and E. Zălinescu, "Tenderbake – A Solution to Dynamic Repeated Consensus for Blockchains," 2020. [Online]. Available: https://arxiv.org/abs/2001.11965, accessed: 03.09.2022.

[50] (2021). "Tezos Agora - Tezos Energy Consumption," Tezos, [Online]. Available: https://wiki.tezosagora.org/learn/baking/tezos-energy-consumption. accessed: 06.08.2022.

[51] (2021). "Governance and validation on Tezos," Tezos, [Online]. Available: https://tezos.com/governance/. accessed: 03.09.2022.

[52] (2021). "Tezos Agora Wiki - Formal Verification," Tezos, [Online]. Available: wiki.tezosagora.org/learn/smartcontracts/michelsonandcoq. accessed: 25.09.2022.

[53] (2021). “Tezos Agora - Smart Contracts,” OpenTezos, [Online]. Available: https://wiki.tezosagora.org/learn/smartcontracts. accessed: 03.09.2022.

[54] M. Hiron. (2022). “Open Tezos Smart Contracts,” OpenTezos, [Online]. Available: https://opentezos.com/michelson/smart-contracts. accessed: 03.09.2022.

[55] A. Kaur. (2021). “Indexing in Databases,” GeeksforGeeks, [Online]. Available: https://geeksforgeeks.org/indexing-in-databases-set-1/. accessed: 24.09.2022.

[56] (2022). “Truffle- Sweet Tools for Smart Contracts,” Truffle, [Online]. Available: https://trufflesuite.com. accessed: 18.07.2022.

[57] (2019). “Truffle blockchain developer tools to support Hyperledger Fabric, R3 Corda,” Ledger Insights, [Online]. Available: https://www.ledgerinsights.com/truffle-blockchain-developer-tools-hyperledger-fabric-r3-corda/. accessed: 04.10.2022.

[58] (2022). “Infura- The World’s Most Powerful Blockchain Development Suite,” Infura, [Online]. Available: https://infura.io. accessed: 18.07.2022.

[59] (2022). “We focus on the infrastructure,” Infura, [Online]. Available: https://infura.io/product/ethereum. accessed: 04.10.2022.

[60] S. Becker. (2022). “The 6 Best Crypto Wallets for Most Investors, According to Experts,” Ledger Insights, [Online]. Available: https://time.com/nextadvisor/investing/cryptocurrency/best-crypto-wallets/. accessed: 04.10.2022.

[61] P. Wang. (2020). “How MetaMask stores your wallet secret?” [Online]. Available: https://www.wispwisp.com/index.php/2020/12/25/how-metamask-stores-your-wallet-secret/. accessed: 11.10.2022.

[62] R. A. McNeal, “The Brides of Babylon: Herodotus 1.196,” *Historia: Zeitschrift für Alte Geschichte*, vol. 37, no. 1, pp. 54–71, 1988. [Online]. Available: https://jstor.org/stable/4436038?seq=1, accessed: 30.09.2022.

[63] S. Karp. (2008). “Google AdWords: A Brief History Of Online Advertising Innovation,” Publishing 2.0, [Online]. Available: https://publishing2.scottkarp.ai/2008/05/27/google-adwords-a-brief-history-of-online-advertising-innovation/. accessed: 09.07.2022.

[64] Statista. (2022). "Advertising revenue of Google from 2001 to 2021," [Online]. Available: https://statista.com/statistics/266249/advertising-revenue-of-google/. accessed: 02.07.2022.

[65] V. Krishna, *Auction Theory*, 2nd ed. Academic Press, 2009, ISBN: 9780123745071.

[66] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," 1961. [Online]. Available: https://cs.princeton.edu/courses/archive/spr09/cos444/papers/vickrey61.pdf, accessed: 30.09.2022.

[67] T. Roughgarden, *CS269I: Incentives in Computer Science - Lecture 13: Introduction to Auctions*, Nov. 2016. [Online]. Available: https://timroughgarden.org/f16/l/l13.pdf.

[68] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[69] E. H. Clarke, "Multipart pricing of public goods," 1971. [Online]. Available: https://https://www.jstor.org/stable/30022651?seq=1, accessed: 30.09.2022.

[70] T. Groves, "Incentives in Teams," *Econometrica*, vol. 41, no. 4, pp. 617–631, 1973. [Online]. Available: https://www.jstor.org/stable/1914085, accessed: 30.09.2022.

[71] P. Milgrom, Ed., *Putting Auction Theory to Work*. Cambridge U. Press, 2004, ISBN: 9780521536721.

[72] P. Jouvelot and E. J. Gallego Arias, *Mech.v*, 2022. [Online]. Available: https://github.com/jouvelot/mech.v, accessed: 25.09.2022.

[73] P. Jouvelot, L. Massoni Sguerra, and E. J. Gallego Arias, "Towards a Generic Coq Proof of the Truthfulness of Vickrey–Clarke–Groves Auctions for Search," 2021.

[74] P. Jouvelot and E. J. Gallego Arias, *A Foundational Framework for the Specification and Verification of Mechanism Design*, Poster, 2022. [Online]. Available: https://hal-mines-paristech.archives-ouvertes.fr/hal-03715847, accessed: 25.09.2022.

[75] M. H. Rothkopf, T. J. Teisberg, and E. P. Kahn, "Why Are Vickrey Auctions Rare?" *Journal of Political Economy*, vol. 98, no. 1, pp. 94–109, 1990. [Online]. Available: https://jstor.org/stable/2937643?seq=1, accessed: 30.09.2022.

[76] L. M. Ausubel and P. Milgrom, "The Lovely but Lonely Vickrey Auction," 2006. [Online]. Available: https://milgrom.people.stanford.edu/wp-content/uploads/2005/12/Lovely-but-Lonely-Vickrey-Auction-072404a.pdf, accessed: 30.09.2022.

[77] H. Varian and C. Harris, "VCG Auction in Theory and Practice," *American Economic Review*, vol. 104, no. 5, pp. 442–45, 2014. [Online]. Available: https://aeaweb.org/articles?id=10.1257/aer.104.5.442, accessed: 30.09.2022.

[78] Statista. (2022). "Advertising revenues generated by Facebook worldwide from 2017 to 2026," [Online]. Available: https://statista.com/statistics/544001/facebooks-advertising-revenue-worldwide-usa/. accessed: 02.07.2022.

[79] J. Guo. (2022). "How Facebook (Meta Platforms) Makes Money," [Online]. Available: https://seekingalpha.com/article/4471770-how-does-facebook-make-money. accessed: 02.07.2022.

[80] M. Prater. (2021). "25 Google Search Statistics to Bookmark ASAP," [Online]. Available: https://blog.hubspot.com/marketing/google-search-statistics. accessed: 02.07.2022.

[81] T. Roughgarden. (2016). "CS269I: Incentives in Computer Science - Lecture 13: Introduction to Auctions," [Online]. Available: http://timroughgarden.org/f16/l/l13.pdf. accessed: 12.11.2022.

[82] T. Roughgarden, *Twenty Lectures on Algorithmic Game Theory*. Cambridge U. Press, 2016, ISBN: 9781316779309.

[83] C. Metz. (2015). "Facebook Doesn't Make as Much Money as It Could — On Purpose," WIRED, [Online]. Available: https://www.wired.com/2015/09/facebook-doesnt-make-much-money-couldon-purpose/. accessed: 17.10.2022.

[84] (2022). "About Ad Auctions," Meta Business Help Center, [Online]. Available: https://www.facebook.com/business/help/430291176997542?id. accessed: 17.10.2022.

[85] P. Vigna. (2020). "Bitcoin Price Hits All-Time High Above $19,000, Topping 2017 Record," The Wall Street Journal, [Online]. Available: https://wsj.com/articles/bitcoin-hits-all-time-high-of-19-786-topping-record-from-december-2017-11606750573. accessed: 15.07.2022.

[86] A. Kharpal. (2017). "Bitcoin price hits another record high above $5,800, now up 480% this year," CNBC, [Online]. Available: https://cnbc.com/2017/10/13/bitcoin-price-hits-another-record-high-above-5800.html. accessed: 15.07.2022.

[87] J. Finneseth. (2021). "Ethereum price hits a new high above $4,500 right as Bitcoin recaptures $64k," Cointelegraph, [Online]. Available: https://cointelegraph.com/news/ethereum-price-hits-a-new-high-above-4-500-right-as-bitcoin-recaptures-64k. accessed: 15.07.2022.

[88] M. Zare. (2022). "NFT Sales Hit Record-breaking $25B in 2021: Report," Crypto Economy, [Online]. Available: https://crypto-economy.com/nft-sales-hit-record-breaking-25b-in-2021-report/. accessed: 15.07.2022.

[89] J. Kanani, S. Nailwal, and A. Arjun, "Matic Whitepaper," 2019. [Online]. Available: https://github.com/maticnetwork/whitepaper, accessed: 30.09.2022.

[90] "Comparison of Ethereum, Hyperledger Fabric and Corda," 2017. [Online]. Available: https://smallake.kr/wp-content/uploads/2017/07/2017_Comparison-of-Ethereum-Hyperledger-Corda.pdf, accessed: 30.09.2022.

[91] S. Jani, "An Overview of Ethereum  Its Comparison with Bitcoin," 2017. [Online]. Available: https://researchgate.net/profile/Shailak-Jani/publication/323078799_An_Overview_of_Ethereum_Its_Comparison_with_Bitcoin/links/5a7ea3c14585154d57d53d5d/An-Overview-of-Ethereum-Its-Comparison-with-Bitcoin.pdf, accessed: 30.09.2022.

[92] M. Rauchs, A. Blandin, K. Bear, and S. McKeon, "2nd Global Entreprise Blockchain Benchmarking Study," 2019. [Online]. Available: https://jbs.cam.ac.uk/wp-content/uploads/2020/08/2019-10-ccaf-second-global-enterprise-blockchain-report.pdf.

[93] D. Perez, J. Xu, and B. Livshits, "Revisiting Transactional Statistics of High-scalability Blockchains," 2020. [Online]. Available: https://arxiv.org/pdf/2003.02693.pdf%5C%C3%5C%82%5C%C2%5C%A0.

[94] (2021). "Tezos VS Ethereum: The ultimate comparison," Smartlink, [Online]. Available: https://smartlink.so/tezos-vs-ethereum-the-ultimate-comparison/. accessed: 16.07.2022.

[95] D. Hamilton. (2022). "Avalanche Vs. Ethereum – What's the Difference?" Securities.io, [Online]. Available: https://securities.io/avalanche-avax-vs-ethereum-eth-everything-you-need-to-know/. accessed: 16.07.2022.

[96] L. Kokoris-Kogias. (2022). "Understanding Blockchain Latency and Throughput," Paradigm, [Online]. Available: https://paradigm.xyz/2022/07/consensus-throughput. accessed: 16.07.2022.

[97] P. Prajapati, N. Bhatt, and N. Bhatt, "Performance comparison of different sorting algorithms," *vol. VI, no. Vi*, pp. 39–41, 2017.

[98] J. Goddard. (2022). "Sorting in Solidity without Comparison," Medium, [Online]. Available: https://medium.com/coinmonks/sorting-in-solidity-without-comparison-4eb47e04ff0d. accessed: 18.11.2022.

[99] *Remix Ethereum IDE Deploy & Run*, Ethereum, 2019. [Online]. Available: https://remix-ide.readthedocs.io/en/latest/run.html, accessed: 30.09.2022.

[100] (2022). "Anatomy of Smart Contracts - Memory," Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/anatomy/#memory. accessed: 17.07.2022.

[101] (2022). "Visibility Quantifiers," TutorialsPoint, [Online]. Available: https://tutorialspoint.com/solidity/solidity_contracts.htm. accessed: 18.07.2022.

[102] (2022). "Hashing Algorithms," Ethers, [Online]. Available: https://docs.ethers.io/v5/api/utils/hashing. accessed: 18.07.2022.

[103] A. M. Antonopoulos and G. Wood, *Mastering Ethereum*. O'Reilly Media, Inc, 2018, ch. 4, ISBN: 9781491971949. [Online]. Available: https://oreilly.com/library/view/mastering-ethereum/9781491971932/ch04.html, accessed: 18.07.2022.

[104] (2019). "Truffle-hdwallet-provider," Truffle, [Online]. Available: https://github.com/trufflesuite/truffle-hdwallet-provider. accessed: 18.07.2022.

[105] (2022). "Ethereum Accounts," Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/accounts/. accessed: 25.09.2022.

[106] (2021). “Tezos Agora Wiki - Smart Contracts on Tezos,” Tezos, SmartPy, [Online]. Available: https://wiki.tezosagora.org/learn/smartcontracts. accessed: 22.07.2022.

[107] (2021). “Tezos Agora Wiki - High-Level Languages of Tezo,” Tezos, [Online]. Available: https://wiki.tezosagora.org/learn/smartcontracts#high-level-languages-of-tezos. accessed: 22.07.2022.

[108] (2021). “Tezos Agora Wiki - Michelson,” Tezos, [Online]. Available: https://wiki.tezosagora.org/learn/smartcontracts/michelson. accessed: 22.07.2022.

[109] (2021). “SmartPy Overview - What is SmartPy?” Tezos, SmartPy, [Online]. Available: https://smartpy.io/docs. accessed: 22.07.2022.

[110] (2021). “SmartPy - Typing,” Tezos, SmartPy, [Online]. Available: https://smartpy.io/docs/general/types/. accessed: 22.07.2022.

[111] (2022). “TzKT Tezos Blockchain Explorer - Tezos Accounts,” TzKT, [Online]. Available: https://tzkt.io/accounts. accessed: 22.07.2022.

[112] (2022). “Ethereum Cumulative Unique Addresses,” YCharts, [Online]. Available: https://ycharts.com/indicators/ethereum_cumulative_unique_addresses. accessed: 22.07.2022.

[113] (2021). “Tezster-CLI,” Tezos, [Online]. Available: https://github.com/Tezsure/Tezster-CLI. accessed: 20.09.2022.

[114] (2022). “Tezos Governance Overview,” Tezos, [Online]. Available: https://wiki.tezos.com/learn/governance/tezos-governance-overview. accessed: 20.09.2022.

[115] Paradigm. (2019). “Tezos Governance Overview,” Tezos, [Online]. Available: https://medium.com/paradigm-research/tezos-the-carthage-proposal-the-carthagenet-test-network-the-first-version-of-a-baker-registry-10e455075d93. accessed: 20.09.2022.

[116] (2021). “Protocol Delphi,” Tezos, [Online]. Available: https://tezos.gitlab.io/protocols/007_delphi.html. accessed: 20.09.2022.

[117] C. Wiser. (2021). "Tezos Edo Upgrade + Network Effects," Tezos, [Online]. Available: https://medium.com/tqtezos/tezos-edo-upgrade-network-effects-5adc20075de4. accessed: 20.09.2022.

[118] *Ethereum Improvement Proposals*, Ethereum, 2022. [Online]. Available: https://eips.ethereum.org/erc, accessed: 22.07.2022.

[119] (2020). "The standard for secure blockchain applications," OpenZeppelin, [Online]. Available: https://www.openzeppelin.com/. accessed: 11.10.2022.

[120] (2021). "Top 10 Ethereum development tools in 2021," Analytics Insight, [Online]. Available: https://analyticsinsight.net/top-10-ethereum-development-tools-in-2021. accessed: 20.09.2022.

[121] (2022). "Top 10 Ethereum development tools in 2021," Hardhat - Ethereum development environment for professionals, [Online]. Available: https://hardhat.org. accessed: 20.09.2022.

[122] *Ethereum improvement proposal 1559*, Ethereum-bad, 2019. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md, accessed: 30.09.2022.

[123] (2021). "2020 Dapp Industry Report," DappRadar, [Online]. Available: https://static.coindesk.com/wp-content/uploads/2021/01/DappRadar-2020-Dapp-Industry-Report.pdf. accessed: 08.07.2022.

[124] J. Truby, R. D. Brown, A. Dahdal, and I. Ibrahim, "Blockchain, climate damage, and death: Policy interventions to reduce the carbon emissions, mortality, and net-zero implications of non-fungible tokens and Bitcoin," *Energy Research Social Science*, vol. 88, p. 102 499, 2022. [Online]. Available: https://sciencedirect.com/science/article/pii/S221462962200007X, accessed: 30.09.2022.

[125] F. Rousselot. (2022). "Cryptocurrency, NFTs and the metaverse threaten an environmental nightmare – here's how to avoid it," [Online]. Available: https://theconversation.com/cryptocurrency-nfts-and-the-metaverse-threaten-an-environmental-nightmare-heres-how-to-avoid-it-175761. accessed: 08.07.2022.

[126] E. J. Beyer. (2022). "NFTs and the Environment: Why the Anger Is Unjustified," [Online]. Available: https://nftnow.com/features/nfts-and-the-environment-why-the-anger-is-unjustified/. accessed: 08.07.2022.

[127] (2022). "2021 Dapp Industry Report," DappRadar, [Online]. Available: https://dappradar.com/blog/2021-dapp-industry-report. accessed: 08.07.2022.

[128] F. Rousselot, *Ethereum for everyone*, Ethereum, 2022. [Online]. Available: https://ethereum.org/en/layer-2/, accessed: 08.07.2022.

[129] O. Adejumo. (2022). "Ethereum developers pick September 19 for Merge," CryptoSlate, [Online]. Available: https://cryptoslate.com/ethereum-developers-pick-september-19-for-merge. accessed: 20.09.2022.

[130] (2022). "Mumbai PoS Testnet," Polygon, [Online]. Available: https://docs.polygon.technology/docs/develop/network-details/network/#mumbai-pos-testnet. accessed: 10.07.2022.

[131] (2022). "Announcing the Kiln Merge Testnet," Ethereum, [Online]. Available: https://log.ethereum.org/2022/03/14/kiln-merge-testnet/. accessed: 10.07.2022.

[132] (2022). "Ethereum Blocks - Block Time," Ethereum, [Online]. Available: https://ethereum.org/gl/developers/docs/blocks. accessed: 20.09.2022.

[133] (2022). "WHAT IS GAS?" Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/gas/#what-is-gas. accessed: 14.07.2022.

[134] (2020). "EVM OPCODE Gas Costs," [Online]. Available: https://github.com/djrtwo/evm-opcode-gas-costs. accessed: 14.07.2022.

[135] (2022). "What is Polygon? (MATIC)," Kraken, [Online]. Available: https://www.kraken.com/en-gb/learn/what-is-polygon-matic. accessed: 01.08.2022.

[136] (2022). "Polygon PoS Chain Average Block Time Chart," Polygonscan, [Online]. Available: https://polygonscan.com/chart/blocktime. accessed: 20.09.2022.

[137] (2022). "Gas And Fees - EIP-1559," Ethereum, [Online]. Available: https://ethereum.org/en/developers/docs/gas/#eip-1559. accessed: 20.09.2022.

[138] Y. Liu, Y. Lu, K. Nayak, F. Zhang, L. Zhang, and Y. Zhao, "Empirical Analysis of EIP-1559: Transaction Fees, Waiting Time, and Consensus Security," *arXiv e-prints*, arXiv:2201.05574, arXiv:2201.05574, Jan. 2022. arXiv: 2201.05574 [econ.GN], accessed: 22.10.2022.

[139] L. Dobos. (2022). "Gas fees: Ethereum is now cheaper than Polygon," CryptoSlate, [Online]. Available: https://cryptoslate.com/gas-fees-ethereum-is-now-cheaper-than-polygon/. accessed: 01.08.2022.

[140] (2022). "Ethereum Sharding," Ethereum, [Online]. Available: https://ethereum.org/en/upgrades/sharding/55. accessed: 03.09.2022.

[141] (2022). "Criteo," Criteo, [Online]. Available: https://www.criteo.com/. accessed: 01.08.2022.

[142] F. Schoeman, Ed., *Auction Theory, An Anthology*. Cambridge U. Press, 1984, ISBN: 9780511625138.

[143] D. Lucking-Reiley, "Vickrey Auctions in Practice: From Nineteenth-Century Philately to Twenty-First-Century E-Commerce," *Journal of Economic Perspectives*, vol. 14, no. 3, pp. 183–192, Sep. 2000. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/jep.14.3.183, accessed: 01.08.2022.

[144] J. Montenegro, M. Fischer, J. Lopez, and R. Peralta, "Secure sealed-bid online auctions using discreet cryptographic proofs," *Mathematical and Computer Modelling - MATH COMPUT MODELLING*, vol. 57, Jun. 2013, accessed: 03.09.2022.

[145] A. Chouayakh, A. Bechler, I. Amigo, L. Nuaymi, and P. Maillé, *An Ascending Implementation of the Vickrey-Clarke-Groves Mechanism for the Licensed Shared Access*. Sep. 2021, pp. 87–100, ISBN: 978-3-030-87472-8. [Online]. Available: https://www.researchgate.net/publication/354798506_An_Ascending_Implementation_of_the_Vickrey-Clarke-Groves_Mechanism_for_the_Licensed_Shared_Access, accessed: 01.08.2022.

[146] L. Ausubel, "An Efficient Ascending-Bid Auction for Multiple Objects," *American Economic Review*, vol. 94, pp. 1452–1475, Dec. 2004. [Online]. Available: https://www.researchgate.net/publication/4901671_An_Efficient_Ascending-Bid_Auction_for_Multiple_Objects, accessed: 01.08.2022.

[147] J. Boyar, I. Damgård, and R. Peralta, "Short Non-Interactive Cryptographic Proofs," 2000. [Online]. Available: doi.org/10.1007/s001450010011, accessed: 25.09.2022.

[148] X. Yi, R. Paulet, and E. Bertino, *New Directions in Cryptography*. Springer, Cham, 2014, ISBN: 978-3-319-12228-1. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-12229-8_2, accessed: 01.08.2022.

[149] F. Brandt and T. Sandholm, *Efficient Privacy-Preserving Protocols for Multi-unit Auctions*. Feb. 2005, vol. 3570, pp. 298–312, ISBN: 978-3-540-26656-3. [Online]. Available: https://www.researchgate.net/publication/220797039_Efficient_Privacy-Preserving_Protocols_for_Multi-unit_Auctions, accessed: 01.08.2022.

[150] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85, Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 291–304, ISBN: 0897911512. [Online]. Available: https://doi.org/10.1145/22145.22178, accessed: 22.10.2022.

[151] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," 2018. [Online]. Available: https://eprint.iacr.org/2017/1066.pdf, accessed: 30.09.2022.

[152] (2022). "Polygon zkEVM Public Testnet: The Next Chapter for Ethereum," Polygon, [Online]. Available: https://blog.polygon.technology/polygon-zkevm-public-testnet-the-next-chapter-for-ethereum/. accessed: 22.10.2022.

[153] (2022). "AZTEC NETWORK The Privacy Layer for Web3," Aztec, [Online]. Available: https://aztec.network/. accessed: 22.10.2022.

[154] K. Mesquita. (2022). "On-chain privacy is key to the wider mass adoption of crypto," Cointelegraph, [Online]. Available: https://cointelegraph.com/news/on-chain-privacy-is-key-to-the-wider-mass-adoption-of-crypto. accessed: 22.10.2022.

[155] P. Angell. (2022). "Why Web3 Is Not Yet Safe Enough For Mass Adoption," Screen Rant, [Online]. Available: https://screenrant.com/web3-mass-adoption-safety-privacy-issues/. accessed: 22.10.2022.

[156] (2022). "Tornado Cash," [Online]. Available: https://github.com/tornadocash. accessed: 01.08.2022.

[157] J. Southurst. (2022). "Dutch police arrest Tornado Cash developer, highlighting illegality of 'coin mixers'," CoinGeek, [Online]. Available: https://coingeek.com/dutch-police-arrest-tornado-cash-developer-highlighting-illegality-of-coin-mixers/. accessed: 03.09.2022.

[158] V. Buterin. (2021). "An approximate introduction to how zk-SNARKs are possible," [Online]. Available: https://vitalik.ca/general/2021/01/26/snarks.html. accessed: 01.08.2022.

[159] G. Dagher, P. Marella, M. Matea, and J. Mohler, "BroncoVote: Secure Voting System Using Ethereum's Blockchain," 2018. [Online]. Available: https://researchgate.net/publication/322874160_BroncoVote_Secure_Voting_System_using_Ethereum's_Blockchain, accessed: 30.09.2022.

[160] C. Mitchell. (2022). "Front-Running," [Online]. Available: https://www.investopedia.com/terms/f/frontrunning.asp. accessed: 01.08.2022.

[161] S. Häfner and A. Stewart, "Front-Running, Smart Contracts, and Candle Auctions," 2021. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3846363, accessed: 01.08.2022.

[162] E. Onica. and C.-I. Schifirneţ., "Towards Efficient Hashing in Ethereum Smart Contracts," in *Proceedings of the 16th International Conference on Software Technologies - ICSOFT,*, INSTICC, SciTePress, 2021, pp. 660–666, ISBN: 978-989-758-523-4. [Online]. Available: https://www.scitepress.org/PublicationsDetail.aspx?ID=uXzmScb67p8=&t=1, accessed: 01.08.2022.

[163] A. Sobol, *Frontrunning on Automated Decentralized Exchange in Proof Of Stake Environment*, Cryptology ePrint Archive, Paper 2020/1206, 2020. [Online]. Available: https://ia.cr/2020/1206, accessed: 01.08.2022.

[164] T. Nabi. (2022). "Pure vs view in solidity," [Online]. Available: https://hashnode.com/post/pure-vs-view-in-solidity-cl04tbzlh07kaudnv1ial1gio. accessed: 16.11.2022.

[165] (2022). "What is a brute force attack?" Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/bots/brute-force-attack/. accessed: 03.09.2022.

[166] W. Diffie and M. E. Hellman, "New Directions in Cryptography," 2020. [Online]. Available: https://ee.stanford.edu/~hellman/publications/24.pdf, accessed: 01.08.2022.

[167] R. Muth and F. Tschorsch, "SmartDHX: Diffie-Hellman Key Exchange with Smart Contracts," 2020. [Online]. Available: https://eprint.iacr.org/2020/325.pdf, accessed: 01.08.2022.

[168] (2018). "Cryptography in a smart contract," [Online]. Available: https://ethereum.org/en/developers/docs/accounts/. accessed: 25.09.2022.

[169] A. Froehlich. (2022). "One-Time Pad," TechTarget, [Online]. Available: https://www.techtarget.com/searchsecurity/definition/one-time-pad. accessed: 01.08.2022.

[170] B. WhiteHat, K. Gurkan, and K. Wei Jie, *MicroMix: A noncustodial Ethereum mixer based on zero-knowledge signalling*, 2019. [Online]. Available: https://github.com/weijiekoh/mixer, accessed: 03.09.2022.

[171] W.-M. Lee. (2021). "Blockchain Series — How MetaMask Creates Accounts," Meta Business Help Center, [Online]. Available: https://levelup.gitconnected.com/blockchain-series-how-metamask-creates-accounts-a8971b21a74b. accessed: 21.10.2022.

[172] J. Schweitzer, S. Couture, and T. Beiko, *Epicenter - Episode 451 Ethereum Foundation – The Ethereum Merge*, Epicenter, 2022. [Online]. Available: https://epicenter.tv/episodes/451/, accessed: 10.10.2022.

[173] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, *Advanced Encryption Standard (AES)*, 2001-11-26 2001, accessed: 21.10.2022.

[174] J. P. Degabriele, J. Govinden, F. Günther, and K. G. Paterson, "The Security of ChaCha20-Poly1305 in the Multi-User Setting," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 1981–2003, ISBN: 9781450384544. [Online]. Available: https://doi.org/10.1145/3460120.3484814, accessed: 21.10.2022.

[175] M. Creeger, "The Rise of Fully Homomorphic Encryption: Often Called the Holy Grail of Cryptography, Commercial FHE is Near," *Queue*, vol. 20, no. 4, pp. 39–60, Aug. 2022. [Online]. Available: https://doi.org/10.1145/3561800, accessed: 21.10.2022.

[176] O. Pomerantz. (2022). “Security choices and multi-signature wallets,” [Online]. Available: https://blog.logrocket.com/security-choices-multi-signature-wallets. accessed: 16.11.2022.

## APPENDIX A

# NAIVE VCG FOR SEARCH IN SOLIDITY

```
1  //Author: Lucas Massoni Sguerra
2  //Institution: CRI - MINES ParisTech
3  //contact: lucas.sguerra@mines-paristech.fr
4
5  pragma solidity >=0.7.0 <0.8.0;
6
7  //Contract owned, from which VCG will inherit ownership, a basic access
8  //control mechanism
9  contract owned {
10     constructor() {
11         //setting owner as the contract deployer
12         owner = msg.sender;
13     }
14
15     address payable owner;
16
17     //modifier that only allows the owner address to call certain functions
18     modifier onlyOwner {
19         require(msg.sender == owner, "Only owner can call this function.");
20         _;
21     }
22
23     function transferOwnership(address payable newOwner) external onlyOwner {
24         if (newOwner != address(0)) {
25             owner = newOwner;
26         }
27     }
28 }
29
30 //VCG contract inheriting ownership from owned contract
31 contract VCG is owned {
32     //Flag to signalize if an auction is underway
33     bool isOpen;
34     //auctions click-through rate values
35     uint256[] private ctrs;
36     //bids values
37     uint256[] private bids;
38     //bidders' addresses
```

```
39      address[] private agents;
40      //winners' prices
41      uint256[] private prices;
42
43      //Events
44      //Event emmited when an auction is opened, broadcasting the ctrs
45      event Open(uint256[] ctrs);
46      //Event emmited when an auction is ended, broadcasting winner agents and
        corresponding prices
47      event EndAuction(address[] agents, uint256[] prices);
48      //Error event, with was substitued by an require in our updated contract
49      event ErrorEvent(string error);
50
51      //Constructior, set owner and isOpen as false
52      constructor() {
53          owner = msg.sender;
54          isOpen = false;
55      }
56
57      //Function open auction, updating the ctr values
58      function openAuction(uint256[] calldata newCTRs) external onlyOwner {
59          if (!isOpen) {
60              isOpen = true;
61              delete prices;
62              delete bids;
63              delete agents;
64              ctrs = newCTRs;
65              emit Open(ctrs);
66          } else {
67              emit ErrorEvent("Ongoing auction");
68          }
69      }
70
71      //Unused (for test only)
72      function updateCTRs(uint256[] calldata newCTRs) external onlyOwner {
73          if (!isOpen) {
74              ctrs = newCTRs;
75          }
76      }
77
78      //Function for bidding, stores the bidder's address and bid value
79      function bid(uint256 amount) external returns (uint256 numberOfBids) {
80          if (!isOpen) {
81              emit ErrorEvent("auction not open yet");
82              return 0;
83          } else {
84              bids.push(amount);
85              agents.push(msg.sender);
```

```
86              return bids.length;
87          }
88      }
89
90      //Function for closing auction
91      //sort bids and calculate winners and corresponding prices
92      function closeAuction() public onlyOwner returns (uint256[] memory Prices)
        {
93          isOpen = false;
94          uint256 length = bids.length;
95          if (bids.length > 0) {
96              uint256[] memory data = bids;
97              uint256[] memory labels = bids;
98
99              for (uint256 j = 0; j < length; j++) {
100                 labels[j] = j;
101             }
102
103             for (uint256 j = 0; j < length; j++) {
104                 uint256 i = j;
105                 while ((i > 0) && (data[i] >= data[i - 1])) {
106                     swap(i, data, labels);
107                     i--;
108                 }
109             }
110
111             uint256[] memory result = calculatePrice(labels);
112
113             return result;
114         } else {
115             emit ErrorEvent("No bids to be auctioned");
116         }
117     }
118
119     //Internal function to calculate winner's prices following VCG's algorithm
120     function calculatePrice(uint256[] memory labels) internal
121     returns (uint256[] memory) {
122         for (uint256 i = 0; (i < ctrs.length && i < agents.length); i++) {
123             uint256 price_i = 0;
124
125             for (uint256 j = (i + 1); j < (ctrs.length + 1); j++) {
126                 price_i =
127                 price_i + (getElement(bids, labels[j]) *
128                 (getElement(ctrs, j - 1) - getElement(ctrs, j)));
129             }
130             prices.push(price_i);
131         }
132         emit EndAuction(agentsSlice(labels), prices);
```

```
133         return prices;
134     }
135
136     //Swap the positions of two elemements in the data and
137     //labels table, part of the insert sort of closeAuction
138     function swap(
139         uint256 i,
140         uint256[] memory data,
141         uint256[] memory labels
142     ) internal pure {
143         uint256 tempData = data[i];
144         uint256 tempLabels = labels[i];
145         data[i] = data[i - 1];
146         labels[i] = labels[i - 1];
147         data[i - 1] = tempData;
148         labels[i - 1] = tempLabels;
149     }
150
151     //Slices agents array, to generate a winners table
152     function agentsSlice(uint256[] memory labels) private view returns (
    address[] memory winners) {
153         winners = new address[](ctrs.length);
154         if (ctrs.length < agents.length) {
155             for (uint256 i = 0; i < ctrs.length; i++) {
156                 winners[i] = (agents[labels[i]]);
157             }
158             return winners;
159         } else {
160             return agents;
161         }
162     }
163
164     //Function to cancel auction and reset parameters
165     function cancelAuction() public {
166         if (isOpen) {
167             isOpen = false;
168             if (bids.length > 0) {
169                 delete bids;
170                 delete agents;
171                 delete ctrs;
172             }
173         }
174     }
175
176     //Get element from list at position i
177     //if there isn't an element, returns zero
178     function getElement(uint256[] storage list, uint256 i) internal view
    returns (uint256 value) {
```

```
179         if (i < list.length) return list[i];
180         else return 0;
181     }
182 
183     //View function to check if an auction in underway
184     function isOPen() public view returns (bool) {
185         return isOpen;
186     }
187 }
```

## APPENDIX B

# NAIVE VCG FOR SEARCH IN SMARTPY

```
#Author: Lucas Massoni Sguerra
#Institution: CRI - MINES ParisTech
#contact: lucas.sguerra@mines-paristech.fr

#Importing SmartPy
import smartpy as sp

#Defining our VCG contract
class SponsoredVCG(sp.Contract):
    def __init__(self, owner):
        self.init(
            owner = owner,
            isOpen = sp.bool(False),
            #initiaizing click-through rates map
            ctrs = sp.map(l = {}),
            #initiaizing bids map
            bids = sp.map(l = {}),
            #initiaizing agents map
            agents = sp.map(l = {}),
            #initiaizing prices map
            prices = sp.map(l = {}),
         )

    #transfer ownership entry point
    @sp.entry_point
    def transferOwnership(self, params):
        sp.verify(sp.sender == self.data.owner)
        self.data.owner = params

    #Update CTRs entry point, unused (for test only)
    @sp.entry_point
    def updateCTRs(self, params):
        sp.verify(sp.sender == self.data.owner)
        sp.if (self.data.isOpen == False) :
            self.data.ctrs = params


    #Open auction entrypoint, with ctrs values as input
```

```
39      @sp.entry_point
40      def openAuction(self, params):
41          sp.verify(sp.sender == self.data.owner)
42          sp.if (self.data.isOpen == False) :
43              self.data.isOpen = True
44              self.data.prices = ({})
45              self.data.bids = ({})
46              self.data.agents = ({})
47              self.data.ctrs = params
48
49      #Entrypoint for bidding
50      @sp.entry_point
51      def bid(self, params):
52          #length of bids map
53          l = sp.local('l', sp.len(self.data.bids))
54          #store bid value
55          self.data.bids[l.value] = (params)
56          #store bidder's address
57          self.data.agents[l.value] = sp.sender
58
59      #Entrypoint that cancels ongoin auction and reset parameters
60      @sp.entry_point
61      def cancelAuction(self):
62          sp.verify(sp.sender == self.data.owner)
63          sp.if (self.data.isOpen) :
64              self.data.isOpen = False
65              self.data.bids = ({})
66              self.data.agents = ({})
67              self.data.ctrs = ({})
68
69      #Endpoint to close auction, and calculate winners and prices
70      @sp.entry_point
71      def closeAuction(self):
72          sp.verify(sp.sender == self.data.owner)
73          l = sp.local('l', sp.len(self.data.bids))
74          j = sp.local('j', l.value - sp.nat(1))
75          sp.for x in self.data.bids.items():
76              #Sorting bids
77              self.insertSort( x.key )
78          self.calculatePrice(2)
79
80      #Sub entrypoint, similar to an internal function.
81      #Here used for insertion sort of the bids
82      @sp.sub_entry_point
83      def insertSort(self, param):
84          self.data.isOpen = False
85          jx = sp.local('jx', param)
86          jint = sp.local('jint', sp.to_int(param))
```

```
87
88          def swap(i):
89              tempBid = sp.local("tempBid", self.data.bids[nat(i)])
90              tempAgent = sp.local("tempAgent", self.data.agents[nat(i)])
91              self.data.bids[nat(i)] = self.data.bids[nat(i - 1)]
92              self.data.agents[nat(i)] = self.data.agents[nat(i - 1)]
93              self.data.bids[nat(i - 1)] = tempBid.value
94              self.data.agents[nat(i - 1)] = tempAgent.value
95
96          def nat(x):
97              return sp.as_nat(x)
98
99          sp.while ((nat(jint.value) > 0) &
100         (self.data.bids[nat(jint.value)] >=
101         self.data.bids[nat(jint.value - 1)]) ) :
102             swap(jint.value)
103             jint.value -= 1
104
105     #Sub entrypoint for calculating prices
106     @sp.sub_entry_point
107     def calculatePrice(self, param):
108         sp.if ( sp.len(self.data.bids) > 0 ) :
109             lenCtr = sp.local("lenctr", sp.len(self.data.ctrs))
110             self.data.ctrs[lenCtr.value] = 0
111             i = sp.local('i', 0)
112             jc = sp.local('jc', 0)
113             price_i = sp.local("price_i", 0)
114             oneNat = sp.as_nat(1)
115             sp.while ( (i.value < lenCtr.value) &
116             (i.value < sp.len(self.data.bids)) ):
117                 price_i.value =  0
118                 jc.value =  i.value + oneNat
119                 sp.while ( jc.value < (lenCtr.value  + 1) ):
120                     price_i.value = price_i.value + (self.data.bids[jc.value]
    * (self.data.ctrs[sp.as_nat(sp.to_int(jc.value) - 1)] - self.data.ctrs[jc.
    value]))
121                     jc.value += 1
122                 self.data.prices[i.value] = price_i.value
123                 i.value += 1
124             del self.data.ctrs[lenCtr.value]
```

## RÉSUMÉ

La sécurité et la transparence des blockchains semblent fournir un environnement adapté pour les enchères. Nous nous focalisons sur l'enchère Vickrey-Clarke-Groves pour la recherche sponsorisée (VCG) pour évaluer cette hypothèse. Nous proposons et utilisons une méthodologie pour la comparaison de différents blockchains du point de vue des contrats intelligents (*smart contracts*). En utilisant VCG, nous avons comparé Ethereum et Tezos ainsi que les mises à jour récentes d'Ethereum sous la forme d'Ethereum Merge et Polygon POS. Enfin, nous analysons les conséquences du manque de confidentialité des blockchains dans une enchère telle que VCG, en proposant trois nouveaux algorithmes pour en atténuer les effets négatifs.

## MOTS CLÉS

*Smart contracts*, Théorie des enchères, Cryptomonnaies, Théorie des jeux, Confidentialité, Mécanisme de Vickrey-Clarke-Groves.

## ABSTRACT

The security and transparency of blockchain systems should provide an attractive environment for auctions. We focus on the auction known as Vickrey–Clarke–Groves for sponsored search (VCG for search) to assess this claim. We proposed and made use of a methodology for the comparison of different blockchain systems from the perspective of smart contracts. Using VCG for search, we compared Ethereum and Tezos as well as the recent updates to the Ethereum protocol in the form of the Ethereum Merge and Polygon POS. Finally we analyze the consequences of the inherent lack of privacy of blockchains to a truthful auction such as VCG for search, proposing three new algorithms to mitigate these negative effects.

## KEYWORDS

Smart contracts, Auction theory, Cryptocurrency, Game theory, Privacy, Vickrey–Clarke–Groves mechanism.