The Price of Smart Contract Privacy

Lucas Massoni Sguerra, Pierre Jouvelot, Fabien Coelho, Emilio Jesús Gallego Arias, Gérard Memmi

The Price of Smart Contract Privacy

(*CRI, Mines Paris, PSL University, Tech. Rep. E/465/CRI, 6/6/2024*)

LUCAS MASSONI SGUERRA, Mines Paris, PSL University, France
PIERRE JOUVELOT*, Mines Paris, PSL University, France
FABIEN COELHO, Mines Paris, PSL University, France
EMILIO J. GALLEGO ARIAS, Inria Paris, France
GÉRARD MEMMI, LTCI, Télécom Paris, IP Paris, France

Smart contracts face a significant challenge regarding the data transparency inherent to the blockchain-based decentralized systems on which they run. This transparency can limit the potential applications and use cases of smart contracts, especially when privacy and confidentiality are paramount. Presently, blockchain applications that require a certain level of privacy will tend to rely on off-chain, centralized solutions. However, this approach introduces trade-offs, potentially compromising the trust and security provided by blockchain technology.

In this article, we advocate for the integration of cryptographic tools into smart contracts, aiming to enhance privacy and address transparency concerns in applications. We introduce the notion of a *Privacy Framework* (PF) as the general building block that addresses privacy issues in smart contracts by linking privacy requirements and adequate implementations. Since auction are important applications that strongly rely on privacy for reaching their full potential, we adopt in this paper the auction known as Vickrey–Clarke–Groves (VCG) Auction for Sponsored Search as a use case to develop the notion of PFs. In practice, we provide three PF *instances*, of increasing complexity, to improve the privacy assurances of specific auction smart contracts. Our experimental assessment of these PF instances suggest they are efficient, not only in terms of privacy preservation, but also in gas and monetary cost, two crucial factors for the viability of smart contracts.

Additional Key Words and Phrases: blockchain, distributed ledger technology, Ethereum, smart contracts, performance, privacy, auctions, Vickrey-Clarke-Groves auction

1 INTRODUCTION

Blockchain systems, an instance of distributed ledger technologies, such as Bitcoin, ensure the security, integrity, and fairness of financial transaction execution in the absence of a governing authority trough the transparency of its data. With the advent of the so-called "smart contracts" , i.e., Turing-complete executable programs run on blockchain-based platforms, as introduced in 2015 by Ethereum [4][37], another important public blockchain, and now generalized to other distributed ledgers such as Tezos[1], this transparency was inherited by general programs, i.e., code and storage. The security and decentralization of public blockchains are paramount to smart contracts, which may be susceptible to collusion or dishonest behavior from participants. Conversely, although what ultimately bestows security to the system, this inherent transparency of blockchains is a severe handicap for any process demanding even a small amount of privacy.

In this paper, we introduce new proposals that manage to preserve privacy in smart contracts while avoiding to compromise the decentralization and transparency of their applications. We follow up with a discussion of the trade-offs of these approaches through an analysis of their performance via different use-case implementations .

*Also with LTCI, Télécom Paris, IP Paris, France
[1]https://tezos.com

Authors' addresses: Lucas Massoni Sguerra, lucas.sguerra@minesparis.psl.eu, Mines Paris, PSL University, Paris, France; Pierre Jouvelot, pierre.jouvelot@minesparis.psl.eu, Mines Paris, PSL University, Paris, France; Fabien Coelho, fabien.coelho@minesparis.psl.eu, Mines Paris, PSL University, Paris, France; Emilio J. Gallego Arias, e@x80.org, Inria Paris, Paris, France; Gérard Memmi, gerard.memmi@telecom-paris.fr, LTCI, Télécom Paris, IP Paris, Paris, France.

1.1 Context

Ethereum expanded on Bitcoin's distributed payment functionalities with the addition of the scripting language Solidity [4], whose programs are popularized under the term "smart contracts". The advent of smart contracts expanded the distributed ledger technology from a decentralized payment system to a decentralized computer, which in turn made possible the appearance of decentralized applications (or "dApps"). These applications enjoy the same security and decentralization properties as any other data stored in the blockchain.

Due to the nature of public blockchains, smart contract execution allows for complete transparency in terms of data and code. Such property provides a secure and tamper-proof environment for applications that require honesty or are particularly susceptible to collusion. For instance, for voting systems, a smart contracts implementation could ensure that each vote is accurately counted and that there is no manipulation of the results. Another example are public auctions, in which a a dedicated smart contract would guarantee that the bids are recorded accurately, that the outcome is properly determined and that there is no collusion between bidders to manipulate the auction results. The transparency and security of smart contract execution make it an ideal solution for a range of applications that do not want to rely upon honesty, fairness, and trust from a central authority.

Nevertheless, these processes may also require some amount of privacy to ensure their correct operation. For instance, elections need to keep participant identities in secret to prevent voters' manipulation. Also, in some types of auctions, such as so-called "truthful" auctions, privacy is paramount to ensure the correct functioning of the auction mechanism. Thus these applications tread into a fine balance of needing both transparency and some level of privacy. This is precisely this balance that we explore in this article and for which we introduce new solutions.

1.2 Contributions

The main contributions of this paper are as follows.

- We introduce the notion of *privacy frameworks* (PF) for smart contracts that link key privacy requirements with possible implementable solutions, or *instances*, which we illustrate here via the case of smart contracts for auctions.
- We provide examples of PFs such as the basic Blockchain PF or Smart Contract PFs, based on four key PF requirements: anonymity, confidentiality, record secrecy, and transfer secrecy.
- We design and implement in Solidity three PF instances for a particular Smart Contract PF, namely VCG PF, for a particular auction case, the Vickrey-Clarke-Groves (VCG) Auction for Sponsored Search, showcasing incremental privacy-preserving capabilities.
- We analyze the run-time performance efficiency of these three PF instances in terms of gas usage and price.

1.3 Structure of paper

This paper is structured in 5 sections. After this initial introduction, which sets the context of this work, Section 2 introduces the notion of privacy frameworks and describe existing partial solutions for privacy preservation in Ethereum. Section 3 details four instances of a privacy framework dedicated to the VCG for Sponsored Search auction smart contract, seen here as a use case for the general class of smart contracts. Section 4 explores the experimental impact of these privacy instances, comparing their efficiency and evaluating their practical usability. To conclude, Section 5 introduces possible future improvements and discusses other future research perspectives.

2 PRIVACY

Privacy is a complex and subjective notion. In the words of 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.*" [31]. 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.

2.1 Privacy frameworks

One can envision a whole continuum of privacy policies, depending on the nature, amount or scope of the privacy-endowed data. We define a *privacy framework* (PF) as a general reference building block that links a specific set of privacy requirements and (possibly partial) solutions to implement them.

The first concrete example of a PF is the *Blockchain* PF based on privacy requirements defined via two key properties that we believe are of utmost importance for privacy in all blockchains. Of course, other more specific PFs can be defined for dedicated applications (as we see below for smart contracts for auctions). These two properties are build upon the conventional concepts of anonymity and confidentiality:

- anonymity, the capability of hiding the identity of someone;
- confidentiality, capability of hiding the contents of a message or data.

In the context of public blockchains, which 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.

Within the Blockchain PF, the common concepts of anonymity and confidentiality are refined as:

- 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 state changes.

Of course, public blockchains, such as Ethereum, already provide some form of weak anonymity. It is not possible to link accounts' 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 grant any confidentiality to their transactions save for Zexe [21] (see below). 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.

This lack of confidentiality also 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 the following section.

2.2 Privacy instances

As the usage of smart contracts expands beyond simple token transfers, there has been significant efforts in the business and research communities to try to mitigate the inherent lack of privacy in blockchains system [6] [5]due to its transparency principle. This issue is seen as a significance hindrance for the widespread adoption of blockchain-based systems [24] [9]. Here, we present some interesting approaches for bestowing some secrecy in Ethereum's smart contracts. They correspond to particular solutions, or *PF instances*, to enforce the Blockchain PF requirements.

2.2.1 Tornado Cash. Tornado Cash [7] is a (now infamous, due to the controversy over the use of this mixing software technology raised [34]) 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 non-interactive succinct arguments of knowledge") proofs to break the link between the depositor and the withdrawer of a coin. A zk-SNARK [12] is a proof that a certain computation has been performed; this proof is both difficult to create and easy to verify, while its zero-knowledge trait means that the corresponding property can be proved without "knowledge", or full revelation of information [14]. Note that the zk-SNARK technology is still growing in popularity, also being used by other systems such as Zerocash [11] or Zexe [13].

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.

2.2.2 BroncoVote. This proposal for a secure and private e-voting system is based on Ethereum's smart contracts. BroncoVote [15] 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 actual 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.

3 SMART CONTRACT PRIVACY FRAMEWORKS

We explore here techniques to maintain the transparency of smart contracts while withholding information that may be deemed sensitive to be shared on-chain. For this purpose, we suggest to build upon cryptography-based algorithms and techniques to be integrated to smart contracts. In this section, we introduce three new different cryptography-based schemes for smart contracts, providing increasing levels of privacy: "Commit-reveal", "Commit-reveal with Diffie–Hellman key exchange" and "Commit-reveal with Diffie–Hellman key exchange and Mixer".

These techniques can be incorporated in many types of algorithms. To both illustrate and experimentally assess these new techniques, we chose here, as a typical use case, the Vickrey–Clarke–Groves Auction for Sponsored Search scheme, for which we present and discuss four implementations, a base case and three privacy-protecting versions.

3.1 Vickrey–Clarke–Groves (VCG)

A VCG auction for sponsored search (abbreviated as "VCG for search") is an instance of the more general Vickrey–Clarke–Groves (or VCG) mechanism, adapted for sponsored search auctions.

3.1.1 Vickrey–Clarke–Groves (VCG) mechanism. Named after William Vickrey, Edward H. Clarke, and Theodore Groves, the VCG mechanism is the golden standard for mechanism design, mechanism design being the science of rule making, a field of economics and game theory that reasons over games with incomplete information to generate a desirable outcome. The VCG mechanism provides some interesting properties such as maximizing the welfare of all participants and being truthful, i.e., being such that participants will receive their best outcome by submitting their actual, or "true", evaluations of the items at stake to the mechanism.

More precisely, in a sales mechanism such as an auction, buyer participants, or bidders, will have a private true evaluation of how much they value the sold item [20]. Depending on what is being auctioned, this private evaluation could be a very sensitive type of data.

In a truthful auction, bidders are guaranteed to receive the best possible outcome for them if they submit their true evaluation, though, because of the sensitivity of this data, this type of mechanism has been severely underutilized in marketplaces [29] [10] [35], at least until Facebook adopted an instance of the VCG mechanism for the sale of advertisement links in the late 2000s, known as Vickrey–Clarke–Groves auctions for sponsored search.

3.1.2 Vickrey–Clarke–Groves (VCG) for sponsored search. In a sponsored search auction, the seller (or auctioneer) wants to sell a certain number of advertisement slots. The participating buyers are publishers interested in the keywords related to a query performed by the user, query which is going to lead to the display of an advertisement-enhanced page. 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 known as their Click-Through Rates (CTRs). 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 up on screen for its target audience (such an appearance is also known as an "impression").

Sponsored-search auctions happen in milliseconds, before the query result page is loaded on the client device, and this occurs millions of times a day, depending on the popularity of the web site. Auction participants report to the auctioneers keywords and bids, on a "cost-per-click" basis. When the advertiser's keyword matches a user's query, the advertiser automatically enters in the auction for these query's advertisements.

The VCG for search algorithm, in which n bidders vie for k slots, each characterized by a CTR α_j, can be outlined as follows, where the CTRs α_j are assumed down-sorted [30]:

(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 $\alpha_{k+1} = 0$, when a click is performed on the ad.

3.2 VCG privacy framework

For the VCG for search auction to behave as initially intended, a certain amount of privacy is necessary; the auction is a truthful sealed-bid auction[2], 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 light of the aforementioned transparency inherent to blockchain systems and of the privacy needs of VCG for search, we define below the four main privacy requirement properties of a new PF, the *VCG privacy framework*, that we judge needed to be addressed in any instance 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;
- record 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

This paper introduces below three instances for the VCG PF for privacy enhancement on VCG for search in smart-contract form that address the previous requirements; their corresponding Solidity implementation and analysis are also provided.

The main intent here is to address the four privacy properties presented without relying on centralized solutions, while preserving some of the needed transparency in order not to compromise the bidders' trust on the auction. The real challenge is to find the proper balance between transparency and obscurity. We also make use of four framework implementations to check the practical feasibility of the proposed PF solutions.

These proposals combine existing modern techniques to improve privacy in the blockchain. A key contribution of this paper is to analyse their relevance to the VCG mechanism, and discuss what type of changes their use would require on smart contracts implementing such an important application. Of course, these techniques could be used in other applications that require the same or similar levels of privacy; our privacy-increased VCG solutions can thus be seen as significant use cases for all such contracts.

3.3 Base case

The base contract that serves as a basis for experimentally assessing privacy is a direct implementation of the VCG algorithm in Solidity; this is a direct translation of the VCG for search algorithm to smart-contract form that doesn't take into account the repercussions that a distributed/public implementation can have on the auction algorithm.

A Solidity smart contract is characterized by its storage and functions. The following subsection discuss both for the base VCG contract.

3.3.1 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 the 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;

[2]In sealed bid auctions, bidders communicate their bids to the auctioneer in secret, for example in a sealed envelope. The idea is to make sure, for privacy reasons, that no bidder knows how much the other auction participants have bid.

[3]The `bool` type is used for booleans; `address` is the Solidity builtin type for identifying participants; the `uint` type denotes unsigned integers; the `[]` notation is used to introduce arrays of variable length.

- `agents(address[])`, the array of addresses of the bidders;
- `winnersAndPrices(address => Price)`, a map from an agent address to a Price structure that contains an uint "price", representing the winners' prices in ETH, and a boolean "payed", that records if the price was already payed or not.

3.3.2 Public functions. Here we specify the main public functions of the VCG for search contract (only the `bid` and `payment` functions are not reserved to the contract-owner role):

- `transferOwnership` transfers the contract ownership;
- `openAuction` opens an auction, providing it an initial array of CTRs as 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[4], and updates the `winnersAndPrices` map with the proper VCG for search values.
- `payment` is used by the participants who won the auction to submit their payment, which will update the boolean inside their Price structure in `winnersAndPrices`, registering that they concluded their payment.

A typical contract execution can be described by a sequence diagram (see Figure 1). There are two types of actors involved, auctioneer and bidders, the same as in every auction. The whole process is divided into three parts. The first is the auction setup; for it, the auctioneer calls the function[5] `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, enabling the winners to pay via the `payment` function.

3.4 Commit-reveal instance

A technique in smart-contract development that is commonly used to prevent front-running attacks is the so-called "commit-reveal" scheme [27] [33], 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 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 instance of the VCG for search contract with the 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 to prevent them to be influenced by

[4]In order to perform the sorting of bids required when closing a VCG for search auction, the base 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, so as to make the auctioneer bear the costs of sorting. Although insertion sort is not adapted to large sets [28] [18], we decided to keep this already existing implementation, since we are mainly interested here in relative comparisons.

[5]The keyword `calldata` is used to indicate that a value is located in the argument storage.

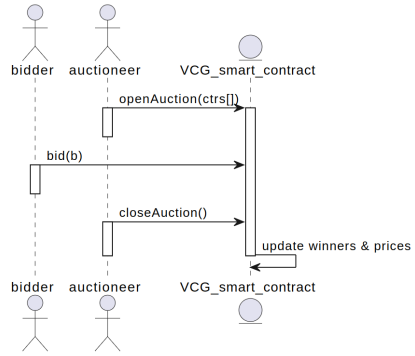

Fig. 1. Sequence diagram of the base VCG for search smart contract

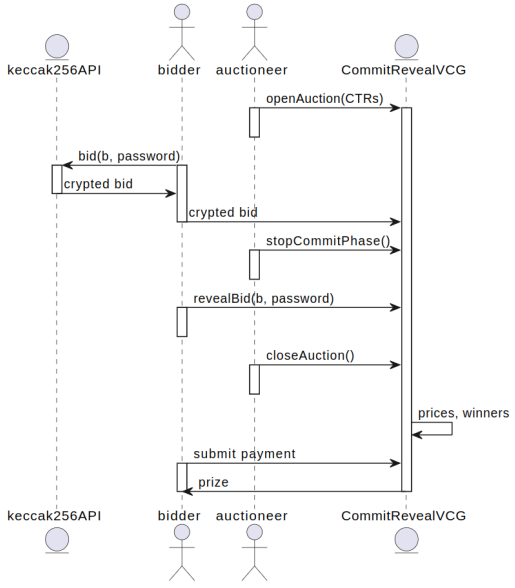

Fig. 2. Sequence diagram of VCG for search with commit-reveal smart contract

or take advantage of the existing bids. The sequence diagram provided in Figure 2 illustrates the basic process of a VCG for search smart contract integrating the commit-reveal scheme for bids.

3.4.1 VCG smart with commit-reveal contract implementation. This new auction contract operates similarly to the base implementation, 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.

Listing 1. View function for computing Keccak-256 hashes

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

Listing 2. Stage control

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

The `hashedBids` table stores the hashed value resulting from the Keccak-256 hash value of a bidder's bid, appended to a user-selected password[6] and the user's 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[7] `hash`, presented in Listing 1.

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

Note that, due to the added complexity of the commit-reveal scheme, phase control requires here more than the single `isOpen` boolean present in the previous VCG implementation. Phase, also called "stage", control needs to be made more explicit into the contract, to regulate its flow through the different steps of the auction. The stages introduced here are enumerated values, seen as integers: `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 2 presents the code associated with contract stage control.

After the reception of the encrypted bids, the commit phase is over; in this 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[8].

In the next stage, the `revealBid` function receives, from each bidder, their `uint256` bid value and password as inputs; the function then checks that the hash of these inputs, once appended

[6]We enforce here the use of one-time string passwords to enhance the security of hashes; otherwise it would be potentially possible, in a relatively easy manner, to decipher a bid value via brute force [8].

[7] *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 [26].

[8]In the implementation, due to the sequential nature of the test protocol we use, 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.

Listing 3. Verifying the validity of a bid value and a password

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

to the message sender's address, is equal to the previously committed encrypted value, as shown in the code extract in Listing 3. 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.

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 `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, available on GitHub[9].

3.4.2 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 the intent with this approach.

Yet, analysing the efficiency of this approach under the scope of the PF characteristics previously specified shows that the commit-reveal enhanced contract is still ineffective on other grounds. Indeed, this implementation offers 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 provides 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.

3.4.3 Trade-offs. The nature of the commit-reveal process imposes a split of the 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, the last winner will have to pay a price

[9] github.com/LucasMSg/The-Price-of-Smart-Contract-Privacy

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.

3.5 Shared key instance

Diffie–Hellman key exchange (DHKE) is a scheme for the secure cryptographic exchange of keys over a public channel introduced by Whitfield Diffie and Martin Hellman [16]. 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 DHKE an interesting match for public blockchain systems, specially since one of the goals addressed in this paper is the preservation of the decentralized nature of the underlying system. Here we propose an improved version for the VCG for search smart contract; it builds upon DHKE to create a shared key between bidders and the auctioneer, in an attempt to introduce some privacy to the system, in accordance to the privacy requirements introduced above.

3.5.1 Multi-Party SmartDHX smart contract. Our implementation of Diffie–Hellman key exchange builds upon the Multi-Party DHKE smart contract SmartDHX introduced by Robert Muth and Florian Tschorsch [25]. With SmartDHX, the authors show that it is possible to implement DHKE 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 DHKE, Solidity being unequipped to do so, since its semantics is 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. 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[10] *SolCrypto*, in particular its relevant function *BigMod*, for modulo calculations.

Readers who want to better understand the functioning can refer to the original SmartDHX paper [25].

3.5.2 VCG for search with shared key smart contract. There are multiple possible approaches for integrating the DHKE algorithm into the 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 3. 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*

[10] github.com/HarryR/solcrypto)

Listing 4. One-time pad decipher via “decrypt” function

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

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 the commit-reveal PF instance. 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 when to stop the commit phase and when to decide to compute the results. Also, as for the commit-reveal contract version, the bids are stored 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 [2], with the claim that users should make their secret keys public once they are in a blockchain transaction. With the privacy framework 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 this PF instance, a simple one-time pad cipher [17] was used 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 cipher 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 ambiguity. For testing, we imposed that the bids need to be written in two digits, and the passwords in 8 characters, though other rules could 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 4 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 decyphers all bids with the secret key. It’s the auctioneer’s responsibility to ensure that the decrypted bids are compatible with the committed hashes from “hashedBids”.

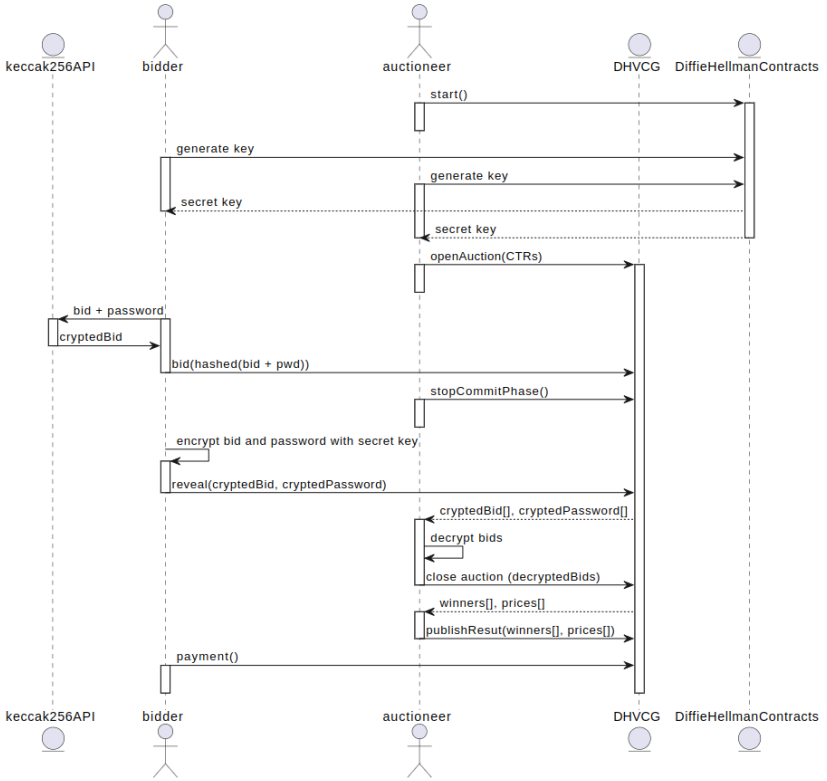

Fig. 3. Shared key PF instance for VCG

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, during testing, the auctioneer is always honest in their calculations; we didn't concern ourselves with a possible misbehavior from their part, although 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 the Commit-reveal VCG with DHKE contract is available[11]. A JavaScript test file for the contract is also available at the same location.

3.5.3 Privacy enhancements. We assess here the performance of this Shared key PF instance for VCG, taking the privacy properties of the VCG PF outlined above as guiding rules.

bidder's anonymity Our implementation offers no anonymity for the bidders; on the contrary, it partially aggravates it, by requiring each participant to have an associated Diffie–Hellman

[11] github.com/LucasMSg/The-Price-of-Smart-Contract-Privacy

client contract (*MultipartySmartDiffieHellmanClient*) in order to participate in the shared-key creation.

bidder's confidentiality In terms of bidder's confidentiality, the Diffie-Hellman VCG contract is efficient in preserving the secrecy of the bids from onlookers who 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 change in transfer secrecy with this approach; all transactions can be traced back to their originators.

3.5.4 Trade-offs. The consequences of adopting 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 [25], the SmartDHX multi-Party key-generation process requires performing at least $\sum_{k=1}^{n} k$ transactions, n being the number of participants (in this case, the number of bidders plus one auctioneer)[12].

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 decided to make it public; in this case, the privacy levels would 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 increases.

3.6 Mixer PF instance

The final PF instance to privacy improvement for the VCG for search smart contract that we introduce here is an upgrade of the VCG with DHKE 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 privacy framework, we're interested in hiding the source of payment for the "prizes", which are, in the case of VCG for search, the auctioned slots.

3.6.1 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" (briefly described in Section 2.2.1) allow multiple users to deposit fixed amounts of coins in a shared pool, where the coins are mixed together; then the users can retrieve those coins with different accounts than those used for deposit.

Mixers such as Tornado Cash and MicroMix [36] use zero-knowledge proofs generated off-chain upon deposit; the proof can then be revealed for retrieving the tokens.

3.6.2 Diffie-Hellman mixer. The idea for this implementation is loosely based on the mixer proposed by MicroMix [36], a mixer for transferring coins using zk-SNARKs, a type of zero-knowledge proof

[12] In the next section, we see that the number of transactions is actually much higher.

that MicroMix generates off-chain. For our solution, we wanted to make use of the shared key generated by DHKE in such a way that all computations could be realized on-chain, a feature that wouldn't compromise the decentralization of the system.

This improvement uses a mixer to list and receive payments from bidders, who, this way, can use accounts different from those used for bidding; this added level of indirection increases the transfers-secrecy level of the contract. This solution relies on both the easiness for users to create new accounts [22] and the use of a Diffie–Hellman secret key, created by multi-party key generation between the bidders and the auctioneer.

3.6.3 Mixer smart contract implementation. The implementation of this PF instance expands upon the previous VCG with Diffie–Hellman version. A typical execution of VCG with DH and mixer is represented in the sequence diagram of Figure 4.

In this new version, as part of the "encryptBid" function, where bidders submit their bids encrypted by the secret shared key, bidders will also have to 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 previously presented, except for the introduction of the necessary changes for the adoption of a 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 the "publishResults" function.

As with the previous implementations, the full code of the VCG with mixer PF instance is available online[13], with an associated JavaScript test file.

3.6.4 Privacy enhancements. The improvements on the VCG PF privacy requirements 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 the previous privacy framework, 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 the winners' addresses to the 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.

3.6.5 Trade-offs. The trade-offs with this mixer PF instance are similar to the side-effects of the shared key PF instance 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

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

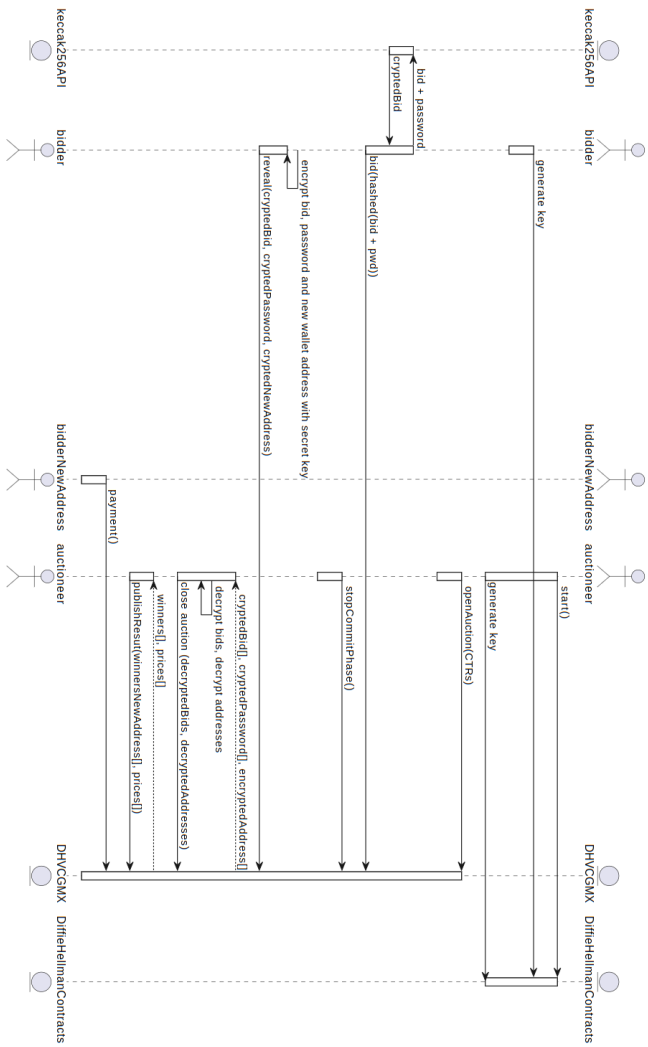

Fig. 4. VCG with mixer PF instance sequence diagram

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 responsibility enforced on the auctioneer increases 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.

4 EXPERIMENTAL ASSESSMENT OF VCG PRIVACY INSTANCES

In this section, we compare and analyse the base VCG contract with the VCG PF instances presented above. The metrics of concerns are the number of transactions and gas usage, together with the associated monetary cost. With this comprehensive benchmarking experiment, we intend to

analyse the practical applicability of these PF instances as well as provide a first idea on the "price of privacy", using incremental approaches that represent a spectrum of possibilities that developers can choose from to select the best trade-off for their particular use case, VCG for search being just one illuminating example to illustrate which various traits can be used in this regard.

4.1 Protocol

As previously discussed, the "Shared key PF instance" and "Mixer PF instance" versions perform the computation and the publishing 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 base VCG and "Commit-reveal PF instance" with a similar structure; there, the results are generated by a *view* function "closeAuction", and the results are published via "publishResults". Thus, the contracts dealt with in this comparison are: "base VCG" (VCG), "base VCG with separated results" (VCGSep), "VCG with commit-reveal PF instance" (CRVCG), "VCG with commit-reveal PF instance with separated results" (CRVCGSep), 'VCG with shared key PF instance" (SKVCG) and "VCG with mixer PF instance" (MixerVCG).

The tests executed were composed of auctions between 5 bidders with the following arbitrary 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 random password strings: [*'0ho95rq4'*, *'84620e92'*, *'abw9eu56'*, *'srolni0n'*, *'c3kknvgf'*]. The auction is supposed to sell 3 items, corresponding each to different click-through rates: [3, 2, 1;].

The decision to use 5 bidders in this test came from limitations with the integration of the preexisting SmartDHX implementation we used; the contracts and tests provided by the authors in their repository [25] 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 meaningful for the performance comparison we intend to make here. We leave more thorough testing of these issues as future work.

The 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[14].

4.2 Gas comparison

Tables 1 and 2 represent the gas used for the VCG privacy framework algorithms, excluding the gas used for DHKE, an issue that is addressed in the following subsection. Table 1 contains the gas usage for functions for which gas consumption is the same between different executions of the same transaction. Table 2 it lists the gas usage for the transactions based on functions where gas consumption can vary depending on its input or on the contract storage.

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 1, we decided to present the estimated "gas usage" of view functions, in the italics, since we consider this information to be relevant to analyse the efficiency of smart contracts, 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 "encryptedB", for both SKVCG and MixerVCG. "RetrieveAllBids" also refers to different functions within the contracts: while SKVCG 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

[14] github.com/LucasMSg/The-Price-of-Smart-Contract-Privacy

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 tables, one can make the immediate following simple observations.

- The deployment costs (Table 1) increase with the algorithmic complexity of the contract. This should be expected, since this metrics is directly related to the code size.
- The first bids (Table 2) are more gas-consuming than the following ones; this occurs due to the first bid setting up the storage for the arrays managed in the contract, an issue we already mentioned.
- Note 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" (Table 1) are, logically, similar to what happens with the "closeAuction" transactions of VCG and CRVCG, respectively.
- MixerVCG has a different style of payment, due to the use of a mixer; the winners addresses are stored differently, and thus the gas required for performing payment differs for each item (Table 2).

Transaction	VCG (gas)	VCGSep (gas)	CRVCG (gas)	CRVCGSep (gas)	SKVCG (gas)	MixerVCG (gas)
Deployment	1,887,325	1,890,181	2,600,777	2,422,229	3,147,601	3,583,014
openAuction	125,703	125,703	128,636	128,636	128,636	131,010
calculate hash			*23,421*	*23,443*	*23,443*	*23,465*
stopCommitPhase			27,323	27,323	27,345	27,345
encryptBid					*60,641*	*60,685*
encryptAddress						*36,055*
retrieveAllBids				*130,533*	*267,091*	*264,869*
retrieveAddr						*114,781*
closeAuction	171,623	*76,471*	172,229	*62,832*	*62,832*	*60,544*
publishResults		103,447		98,650	99,405	187,933

Table 1. Gas usage for the VCG implementations (constant consumption). The values in *italics* correspond to view functions.

Transaction	VCG (gas)	VCGSep (gas)	CRVCG (gas)	CRVCGSep (gas)	SKVCG (gas)	MixerVCG (gas)
1st Bid	112,907	112,907	140,016	139,994	160,454	160,366
Bid	78,707	78,707	108,604	108,582	129,042	128,954
reveal 1			60,626	60,626	81,354	127,164
reveal 2			60,626	60,626	81,354	127,164
reveal 3			60,626	60,626	81,354	127,152
reveal 4			60,626	60,626	81,342	127,152
reveal 5			60,626	60,626	813,54	127,164
payment 1	44,935	44,935	44,973	45,909	45,909	46,741
payment 2	44,935	44,935	44,973	45,909	45,909	48,644
payment 3	44,935	44,935	44,973	45,909	45,909	50,547

Table 2. Gas usage for the VCG implementations (variable consumption).

4.3 SmartDHX

We use the SmartDHX implementation [25] for performing the DHKE required by SKVCG and MixerVCG. In practice, we followed the JavaScript test for Multi-party SmartDHX "2_TestMultiPartyDHX.js" made available by the authors[15], 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 during testing are the following:

- controller deployment, 1;
- 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 [25], 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 a quite different total number of 73 transactions. As it happens, in their GitHub repository[16], 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 one can see when considering each client's number of "answers", as mentioned above. Assuming that each client does execute their required "answer" transaction, the total gas used by each of them is listed in Table 3.

	Client 0	Client 1	Client 2	Client 3	Client 4	Client 5
Answers (gas)	846,688	1,593,725	2,902,234	5,036,715	7,897,951	8,667,425

Table 3. 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 2,293,839 gas, while the deployment cost varies for each client, with an average of 2,332,546.5±5,292.36 gas[17]. Finally, the execution of one "start" transaction by *MultipartySmartDiffieHellmanController* is required, which consumes 990,450 gas.

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

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

[17] The gas costs for the deployment of SmartDHX clients' contracts 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 isn't there already, resulting in different gas costs for clients depending on the number of clients already present in the table (if need be, hash maps could be used to smooth out these differences). It's worth noting that the first client also pays for the initial set up of the "clients" table, similarly to what led to the higher price for the first bidders in the VCG contracts.

4.4 Transaction fees

For the monetary analysis of the PF instances, we estimated the prices of the different implementations in USD using the pricing calculations determined by EIP1559 [3].

Ethereum's (ETH) price dates from August 9th 2023, 19:40 UTC, amounting 1,848.3 USD per ETH[18]. On the same date, the gas costs from Etherscan[19] are divided into a "Base Fee" of 19 Gwei and a "Max Priority Fee" of 1 Gwei.

Transaction fees in fiat currencies, such as US dollars, for Ethereum transactions are subject to variability based on the number of active users. These fees are calculated using two important metrics: gas price and Ether price. Gas price reflects the cost of computational resources required to execute a transaction, while Ether price represents the value of Ethereum's native cryptocurrency. Both of these metrics fluctuate in response to the level of blockchain activity and market interest.

It is worth noting that representing transaction fees in a familiar fiat currency like US dollars can provide a useful reference point for readers who are new to Ethereum and unfamiliar with gas metrics. However, it's important to recognize that this approach fixes the metrics at a specific point in time, and as the blockchain evolves and market conditions change, the presented results will become outdated. Therefore, it's essential to consider the dynamic nature of gas prices and Ether prices when evaluating transaction fees.

Table 1 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, 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)	SKVCG (USD)	MixerVCG (USD)
auctioneer	81.85	78.23	108.38	98.95	125.79	145.23
1st bidder	3.92	3.92	6.6	6.6	8.22	9.79
other bidders	2.82	2.82	5.65	5.65	7.27	8.83
payment	1.66	1.66	1.66	1.69	1.69	1.66

Table 4. Fees, in US dollars, for the participants of the different VCG contracts.

Since, for SKVCG and MixerVCG, a large part of the cost is related to the execution of the Multi-Party SmartDHX, Table 5 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	116.06	142.11	188.81	265.37	368.56	518.21

Table 5. 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.

[18]Pricing from coinbase.com
[19]etherscan.io/gastracker

4.5 Comparative Analysis

The here presented PF instances can be compared by two different metrics, first through the privacy properties previously presented, and finally by the gas usage ans monetary aspect of smart-contract execution.

The comparison between the different PF instances in term of bidder's anonymity, bidder's confidentiality, records secrecy and transfer secrecy is summed up in Table 6.

PF instance	Bidder's anonymity	Bidder's confidentiality	Records secrecy	Transfer secrecy
Base	None	None	None	None
Commit-reveal	None	Partial	Partial	None
Shared key	None	Partial	Partial	None
Mixer	Partial	Partial	Partial	Partial

Table 6. Privacy properties of the different VCG PF instance contracts.

Base VCG and Commit-reveal PF instances 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.

SKVCG and MixerVCG, 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 731.04% from VCGSep to SKVCG, and 755.89% from VCGSep to MixerVCG.

Considering that the key generated by SmartDHX could be reused between different auctions, this monetary impact could be diluted over successive auctions. The increase in fees for the auctioneer, disregarding the fees from the shared key exchange, in SKVCG and MixerVCG is of 60% and 85% by comparison to VCGSep.

As for bidders, by comparison to VCGSep, the results indicate an increase of 5,774.8%±2,592.43%, for the first bidder of SKVCG, and 8,083.5%±3,669.41%, for the subsequent bidders, while, for MixerVCG, one finds 5,817.64%±2,592.43%, for the first bidder, and 8,143.52%±3669.41%, for the rest. Note though that, without the price increase induced by SmartDHX, a more reasonable increase of 127.7%, for the first bidder of SKVCG, 177.64%, for the other bidders of SKVCG, 70.6%, for the first bidder of MixerVCG and 237.64%, for the other bidders would have been recorded.

4.6 Discussion

To assess the practical impact of the previous results, the intrinsic 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 above. Privacy concerns might be significant enough for the auctioneer and bidders to consider accepting to pay increased fees, and adopt one of the proposed PF instances, 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.

Trying to solve the privacy problem for a VCG for search implementation is, in fact, a bit a contradiction in terms. Indeed, what one can find first enticing in the development of an auction system for decentralized applications is 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 show, in this paper, that a balance between transparency and obscurity in order to maintain the user's trust without incurring too much additional system overhead can be designed and implemented.

What one ends up with the proposed PF instances introduced above are auction contracts that can be executed without the aid of off-chain systems and that can obscure bids from non-participants, as well as obscure the link between winners and participants. Admittedly, SKVCG and MixerVCG, which are proposals based on secret key exchanges between all participants enabling everyone to verify the validity of the auctions, 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, one can hope that future developments will give more possibilities to tackle out the privacy problem addressed here, without having to make big compromises.

5 CONCLUSION AND FUTURE WORK

We close this paper with an overview of its main contributions, followed by a presentation of future perspectives for the progress of privacy research on public blockchains.

5.1 Main contributions

This paper deals with a critical concern associated with blockchain technologies, namely the challenge posed by the inherent transparency of blockchains on smart contracts. In line with one of the core design principles of the blockchain paradigm, solutions to such an issue should avoid to resort to centralized techniques, which are all so present in the field and which may compromise the decentralization and security traits of the blockchain technology.

Our proposal builds upon the concept of privacy frameworks (PF). PFs gather the essential building blocks that stem from a system's specific privacy requirements, along with the corresponding (possibly partial) solutions for their implementation.

Within the Blockchain PF, we reformulate the key properties of anonymity and confidentiality to align with the unique characteristics of distributed ledger technologies. We apply then these concepts in an actual use case analysis, namely the Vickrey-Clarke-Groves auction for sponsored search. VCG for search is a special type of auction in which privacy is paramount for its correct functioning and to preserve its truthfulness properties.

The VCG for search PF is defined to handle the privacy requirements specifically linked to the sealed-bid aspect of this auction mechanism. Framing such a mechanism as a contract, we introduce four specific privacy requirements, i.e., bidder's anonymity, bidder's confidentiality, record secrecy, and transfer secrecy. In order to address the defined privacy properties specified in the VCG for search PF, we introduce instances of the VCG for search smart contract building upon cryptographic tools integrated into the contracts' protocol. This results into four different contracts: the VCG base case, and its commit-reveal instance, shared key instance, and mixer instance.

These contracts provide partial effectiveness in meeting the privacy requirements outlined by the VCG for search PF. The most successful instance is the mixer VCG, which provided relief to all the privacy points. Thus, these previous proposals show the theoretical feasibility of achieving privacy within blockchain systems without the need of relying on centralized solutions. However, it is important to be aware, on a more practical note, that this enhanced privacy comes with increased costs, which we detail in the paper, both in terms of complexity and execution expenses.

5.2 Privacy properties comparison

There are a myriad of public blockchain systems in the market today, with a lack of scientific rigour to compare them [23]; an interesting development for our research is to execute a comparison of different public blockchain systems in terms of privacy. Some public blockchains employ methods

of privacy preservation; for instance Monero[1] was conceived for anonymity and confidentiality. Another interesting candidate is Tezos[19], which added privacy capabilities to its public chain in 2020 [32]. A comparison in terms of the privacy instances presented in this article, resulting from the execution of the base VCG contract in these 2 blockchains, compared with the control case of Ethereum, could reveal interesting insights that might be important when trying to choose which blockchain system to adopt.

5.3 Possible enhancements

Another appealing direction we intend to take is the further development and improvement of the VCG PF framework by applying other cryptography technologies that, at face value, seem well adapted for the goal of preserving privacy while conserving the contract's transparency.

A first interesting approach would be the use of homomorphic encryption. Homomorphic encryption is a special form of encryption that allows specific types of computations to be carried out on ciphertexts, so that when decrypted, they match the results of the same operations being performed on the originating plaintexts [38]. Such encryption would be perfect for auctions; ideally one would be able to sort bid values without having to decrypt then, thus preserving bid confidentiality, while maintaining the computation over the data transparent, so that users could attest the good behavior of the contract. Zero-knowledge(ZK) proofs are another appealing cryptographic technique for PFs. There are already dApps that make use of Zero-knowledge proofs to protect information on-chain, such as the infamous Tornado-cash[7], Zerocash or Zexe. Adding ZK-proofs to a VCG contract to prevent information to be published on-chain and to guarantee the proper execution of VCG to the bidders sounds compelling for future advancements.

REFERENCES

[1] Monero [n. d.]. *What is Monero (XMR)?* Monero. https://www.getmonero.org/get-started/what-is-monero/
[2] 2018. *Cryptography in a smart contract.* https://ethereum.org/en/developers/docs/accounts/
[3] 2019. Ethereum improvement proposal 1559. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md
[4] Ethereum, Solidity 2021. *Solidity.* Ethereum, Solidity. https://docs.soliditylang.org/en/v0.8.15
[5] Aztec 2022. *AZTEC NETWORK The Privacy Layer for Web3.* Aztec. https://aztec.network/
[6] Polygon 2022. *Polygon zkEVM Public Testnet: The Next Chapter for Ethereum.* Polygon. https://blog.polygon.technology/polygon-zkevm-public-testnet-the-next-chapter-for-ethereum/
[7] 2022. *Tornado Cash.* https://github.com/tornadocash
[8] Cloudflare 2022. *What is a brute force attack?* Cloudflare. https://www.cloudflare.com/learning/bots/brute-force-attack/
[9] Phoenix Angell. 2022. *Why Web3 Is Not Yet Safe Enough For Mass Adoption.* Screen Rant. https://screenrant.com/web3-mass-adoption-safety-privacy-issues/
[10] Lawrence M. Ausubel and Paul Milgrom. 2006. The Lovely but Lonely Vickrey Auction. (2006). https://milgrom.people.stanford.edu/wp-content/uploads/2005/12/Lovely-but-Lonely-Vickrey-Auction-072404a.pdf
[11] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. 2014. *Zerocash: Decentralized Anonymous Payments from Bitcoin.* Cryptology ePrint Archive, Paper 2014/349. https://eprint.iacr.org/2014/349
[12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 326–349. https://doi.org/10.1145/2090236.2090263
[13] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. 2020. ZEXE: Enabling Decentralized Private Computation. In *2020 IEEE Symp. on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 947–964. https://doi.org/10.1109/SP40000.2020.00050
[14] Vitalik Buterin. 2021. *An approximate introduction to how zk-SNARKs are possible.* https://vitalik.ca/general/2021/01/26/snarks.html
[15] Gaby Dagher, Praneeth Marella, Milojkovic Matea, and Jordan Mohler. 2018. BroncoVote: Secure Voting System Using Ethereum's Blockchain. (2018). https://researchgate.net/publication/322874160_BroncoVote_Secure_Voting_System_using_Ethereum's_Blockchain

[16] Whitfield Diffie and Martin E. Hellman. 2020. New Directions in Cryptography. (2020). https://ee.stanford.edu/~hellman/publications/24.pdf

[17] Andrew Froehlich. 2022. *One-Time Pad.* TechTarget. https://www.techtarget.com/searchsecurity/definition/one-time-pad

[18] Jules Goddard. 2022. *Sorting in Solidity without Comparison.* Medium. https://medium.com/coinmonks/sorting-in-solidity-without-comparison-4eb47e04ff0d

[19] L.M Goodman. 2014. Tezos — a self-amending crypto-ledger. (2014). https://academy.bit2me.com/wp-content/uploads/2021/04/tezos-whitepaper.pdf

[20] Vijay Krishna. 2009. *Auction Theory* (2 ed.). Academic Press.

[21] V. Languille, G. Memmi, and D. Menga. 2024. Fairness-Privacy Issue in Adaptation of ZEXE Protocol for Exchange Between Untrusted Parties. https://hal.science/hal-04445164v1

[22] Wei-Meng Lee. 2021. *Blockchain Series — How MetaMask Creates Accounts.* Meta Business Help Center. https://levelup.gitconnected.com/blockchain-series-how-metamask-creates-accounts-a8971b21a74b

[23] Lucas Massoni Sguerra, Pierre Jouvelot, Emilio J. Gallego Arias, Gérard Memmi, and Fabien Coelho. 2021. Blockchain Performance Benchmarking: a VCG Auction Smart Contract Use Case for Ethereum and Tezos. (2021). https://hal.science/hal-03210222v1

[24] Kieran Mesquita. 2022. *On-chain privacy is key to the wider mass adoption of crypto.* Cointelegraph. https://cointelegraph.com/news/on-chain-privacy-is-key-to-the-wider-mass-adoption-of-crypto

[25] Robert Muth and Florian Tschorsch. 2020. SmartDHX: Diffie-Hellman Key Exchange with Smart Contracts. (2020). https://doi.org/10.1109/DAPPS49028.2020.00022

[26] Tawseef Nabi. 2022. *Pure vs view in solidity.* https://hashnode.com/post/pure-vs-view-in-solidity-cl04tbzlh07kaudnv1ial1gio

[27] Emanuel Onica. and Cosmin-Ionuţ Schifirneţ. 2021. Towards Efficient Hashing in Ethereum Smart Contracts. In *Proceedings of the 16th International Conference on Software Technologies - ICSOFT,*. INSTICC, SciTePress, 660–666. https://doi.org/10.5220/0010606606600666

[28] Purvi Prajapati, Nikita Bhatt, and Nirav Bhatt. 2017. Performance comparison of different sorting algorithms. *vol. VI, no. Vi* (2017), 39–41.

[29] Michael H. Rothkopf, Thomas J. Teisberg, and Edward P. Kahn. 1990. Why Are Vickrey Auctions Rare? *Journal of Political Economy* 98, 1 (1990), 94–109. https://jstor.org/stable/2937643?seq=1

[30] Tim Roughgarden. 2016. *Twenty Lectures on Algorithmic Game Theory.* Cambridge U. Press.

[31] Ferdinand Schoeman (Ed.). 1984. *Auction Theory, An Anthology.* Cambridge U. Press.

[32] Adam Shinder. 2022. *Sapling and Shielded Transactions on Tezos.* Tezos Israel. https://ethereum.org/en/developers/docs/accounts/#account-creation

[33] Andrey Sobol. 2020. Frontrunning on Automated Decentralized Exchange in Proof Of Stake Environment. Cryptology ePrint Archive, Paper 2020/1206. https://ia.cr/2020/1206

[34] Jon Southurst. 2022. *Dutch police arrest Tornado Cash developer, highlighting illegality of 'coin mixers'.* CoinGeek. https://coingeek.com/dutch-police-arrest-tornado-cash-developer-highlighting-illegality-of-coin-mixers/

[35] Hal Varian and Christopher Harris. 2014. VCG Auction in Theory and Practice. *American Economic Review* 104, 5 (2014), 442–45. https://doi.org/10.1257/aer.104.5.442

[36] Barry WhiteHat, Kobi Gurkan, and Koh Wei Jie. 2019. MicroMix: A noncustodial Ethereum mixer based on zero-knowledge signalling. https://github.com/weijiekoh/mixer

[37] Gavin Wood. 2022. Ethereum: A secure decentralised generalised transaction ledger - BERLIN VERSION 8fea825 – 2022-08-22. *Ethereum project yellow paper* (2022).

[38] Xun Yi, Russell Paulet, and Elisa Bertino. 2014. *New Directions in Cryptography.* Springer, Cham. https://link.springer.com/chapter/10.1007/978-3-319-12229-8_2