Enabling Monetization of Depreciating Data on Blockchains

Christian Dahdah, Coline Van Leeuwen, Ziad Kheil, Jérôme Lacan[a], Jonathan Detchart[b] and Thibault Gateau[c]

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, France
{*al-cheikh-christian.el-dahdah, coline.van-leeuwen, ziad.kheil*}*@student.isae-supaero.fr,*

Keywords: Depreciative Data, Fair Exchange, Public Ledger, Ethereum, Solidity, Space Debris.

Abstract: In this paper, we introduce a protocol to securely exchange data on chain while varying its price according to their freshness, maturity and lifetime. The exchange protocol, implemented as a smart contract, is best applied to crowdsourcing systems for fast depreciating digital goods, in which information is publicly shared after a given delay. The smart contract acts as a trusted intermediary to make sure that the funds of a client are delivered to the provider if and only if the data were really transferred. It also ensures that the data will be freely shared on the blockchain when the data has sufficiently depreciated. We demonstrate our work with an available prototype for specific space tracking data exchange.[4]

1 INTRODUCTION

The interest of blockchains is demonstrated for many contexts. Cryptocurrencies are indeed the main applications but the intrinsic properties of blockchains let alone traceability, immutability and decentralization open new opportunities in many contexts such as supply chains or marketplaces. Smart contracts, which allow blockchains to perform transparent computations, led to the development of Distributed applications (Dapps) resulting in powerful services through complementary onchain and offchain procedures.

Crowdsourcing is one of many domains that can take great advantage of distributed ledgers. The underlying principle is to allow users to share data with others. Traditionally, crowdsourcing platforms are managed in a centralized way, prompting several potential concerns about security, trust and privacy. The decentralized structure of blockchains allows to seamlessly handle such issues, therefore, they are natural candidates to deploy crowdsourcing platforms (Kogias et al., 2019; Ma et al., 2020).

Although each crowdsourcing system presents its own specificities, their common issue is that the data is often shared freely. However, this hinders commercial entities to participate in aforesaid platforms. In this paper, we propose an incentive mechanism allowing users to sell digital goods on blockchain based crowdsourcing systems, all the while respecting the golden idea of sharing by publishing them for free after a given time.

The proposed mechanism relies on publishing encrypted data on the blockchain, and selling the decryption key to clients interested by an immediate access to the information.

But in order to stay in the spirit of crowdsourcing data, the provider pledges to publicly release the decryption key, free of charge, after sufficient time has passed.

Thus the mechanism is best suited for data with fast depreciation rates, with an interest to publish information publicly after a certain time in order to build a reputation and establish trust. To illustrate this, the protocol can be used to sell encrypted forex trade signals to clients. When these signals decay the key is publicly shared, and the information is available to everyone, and can be evaluated to establish reputation thus attracting future audience. In our implementation[1] we applied this to space debris positions which also lose precision over time. In this particular case, Trusat (ConsenSys-Space, 2020), a crowdsourcing project, aiming to store space debris and satellite observations to promote space safety can benefit from this Dapp.

[a] https://orcid.org/0000-0002-3121-4824
[b] https://orcid.org/0000-0002-4237-5981
[c] https://orcid.org/0000-0002-8719-5044

[1] https://github.com/ChristianDahdah/Monetization-of-Depreciating-Data-Through-Smart-Contracts

Dahdah, C., Van Leeuwen, C., Kheil, Z., Lacan, J., Detchart, J. and Gateau, T.
Enabling Monetization of Depreciating Data on Blockchains.
DOI: 10.5220/0010252105000507
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 500-507
ISBN: 978-989-758-491-6

Note that, since we are dealing with depreciating data, the price integrated in the contract is also subject to decline. Unrushed clients can therefore wait until the price suits their need. The protocol implemented in a smart contract acts as a judge by solving potential disputes between sellers and buyers. The Claim-and-refund procedures guarantee a private and fair exchange.

To this end, Section 2 presents our protocol enabling the monetization of depreciating data. Algorithms for fair data exchange and for Dispute and Refund are described. An implementation in an Ethereum smart contract is described in Section 3. One of the main objectives is the minimization of the gas spent which could lead to significant contract fees. This implementation is benchmarked in Section 4.

2 PROTOCOL DESCRIPTION

2.1 Protocol Overview

The global context of this paper is a crowdsourcing system implemented over a blockchain where the users share some data freely. Thanks to the transparency provided by the blockchain, we assume that the system implements a reputation mechanism allowing to evaluate the behavior and the quality of the data shared by users (Dennis and Owen, 2015).

The main contribution of this paper is a complete protocol allowing the transfer of depreciating data to different clients through a smart contract. In the case of honest provider and clients, each client pays the amount corresponding to the indicated price at the time she acquired the data.

The global process is summarized in Fig. 1 and detailed in the next paragraphs. Globally, when a provider wants to share data, she first encrypts it and stores it in the smart contract. The provider also announces the type of data, the price and the way it depreciates in function of time. The Smart Contract makes it mandatory to reveal the decryption key of the data before a given deadline in order to allow the provider to retrieve his earnings. This condition is essential to be able to evaluate the correctness of the provided key and the quality of the sold data.

If a client wants to access this data, she transfers the funds corresponding to the current cost of the data, and through the protocol, receives a decryption key. During the procedure, the smart contract does not learn the value of the key but is able to verify the fairness of the exchange once the key is revealed.

Since this procedure requires several exchanges, the contract takes into account any significant time delay in transactions and reimburses clients to fit the depreciated value of the data and not the current price at the time of purchase.

After the revelation of the key by the provider, a certain delay is given to the client to potentially launch a dispute procedure in case of fraud. The Dapp verifies the exchanges and can reimburse the client.

On the other hand, if no dispute was initiated, the smart contract allows the provider to retrieve his funds.

Note that our system does not verify the quality of the transferred data or that the exchanged key allows the decryption of the message but it can be extended by a reputation system (such as the one developed in the Trusat project). With this kind of system, it is not in the best interest of the seller to sell wrong key or data because of the potential reputation repercussions.

2.2 Security Model and Objectives

We consider that the blockchain functionalities ensure user authentication and correct executions of smart contracts. We also consider the following hypotheses on the cryptographic building blocks: [H1] the hash function returns a perfectly random value (random oracle model) and is collision-resistant; [H2] the random generator is assumed to be perfect and [H3] the pseudo-random generator function *PRG* used to generate the stream added to the data is unpredictable. The security objectives are:

Provider Fairness. An honest provider receives the client's funds corresponding to the date of purchase.
Client Fairness. An honest client receives the key that will be publicly shared at the end of the process or a part of the insurance funds provided by the provider if the latter is not honest.

Note that the second objective does not protect against a provider that shares a key different from the one she used to encrypt the data. This case is managed by a reputation system which will detect the fraud at the end of the process and thus will degrade the rating of the provider.

The potential attackers are [A1] the provider that wants to get the client's funds without revealing the data, [A2] a client that wants the data without paying the funds, [A3] a third party who observes the blockchain and wants to recover data without paying.

2.3 Description

Suppose a provider $\mathcal{P}$ wants to sell some information I deemed depreciative. Let $|I|$ be the length of

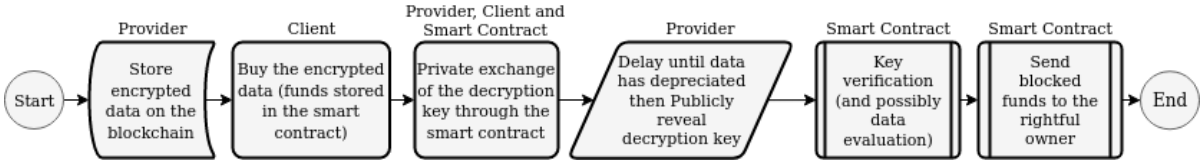

Figure 1: Protocol overview.

I. Transparency and reliability are required keystones for such a transaction.

In view of such objectives, a protocol to safely exchange encrypted data and their keys, over a certain validity period, is presented in Alg.1 and is detailed in the next paragraphs. The management of the funds during the process is illustrated in Fig. 2.

2.3.1 Store Encrypted Data on the Blockchain

To encrypt the data, $\mathcal{P}$ creates a random seed $\mathcal{S}$ and generates $\mathcal{K}$ with $|\mathcal{K}| = |I|$, in order to use stream cipher encryption: $I^{'} = \mathcal{K} \oplus I$. $I^{'}$ is then uploaded to the contract. With this method, any user having $\mathcal{K}$ is able to recover I from $I^{'}$. To save on gas storage cost, the provider can use any PRG to generate $\mathcal{K} = PRG(\mathcal{S})$ and exchange $\mathcal{S}$, since this does not fundamentally change the mechanism of exchange, the following protocol is used to transfer $\mathcal{S}$ which generally has a length smaller that $|\mathcal{K}|$.

To safely exchange information on the blockchain, the Diffie-Hellman (DH) protocol is used with each client, thus $\mathcal{P}$ generates public and private DH keys (P_{priv}, P_{pub}).

$\mathcal{P}$ contacts the smart-contract to sell his product: she chooses a certain initial price, a function Δ detailing the value decrease of his data, a duration of validity for his data, and the minimum quantity of data she pledges to upload. Furthermore, $\mathcal{P}$ can deliberately choose to offer a stake in Ether as an insurance fund of his trustworthiness. These variables are stored in the contract, meanwhile his DH keys are emitted as events to cut on gas cost. The contract assigns an identification Id to this offer.

2.3.2 Exchange Protocol

The steps of the decryption key exchange are the following:

1. At a given time t_0^i, a client $\mathcal{C}^i$ interested in buying this depreciating data must generate (offchain) his DH keys : (C^i_{priv}, C^i_{pub}), and send the funds C^i_{funds} in ether to the contract depending on the current price $\Delta(t_0^i)$ (computed from the initial price by the price decrease function Δ). The funds sent are blocked on the contract and can be withdrawn by the client if step 4 is not accomplished. His public key is emitted through a specific event.
2. With P_{pub} and C^i_{pub}, $\mathcal{P}$ and $\mathcal{C}^i$ generate a shared secret key $\mathcal{K}_3^i$ of length $|\mathcal{S}|$. Offchain $\mathcal{P}$ generates a random binary sequence $\mathcal{K}_2^i$ of length $|\mathcal{S}|$ then, through the contract, emits $\mathcal{S} \oplus \mathcal{K}_2^i \oplus \mathcal{K}_3^i$ as an event on the contract.
3. Then, $\mathcal{C}^i$ computes the hash: $h^i = H(\mathcal{S} \oplus \mathcal{K}_2^i \oplus \mathcal{K}_3^i \oplus \mathcal{K}_3^i) = H(\mathcal{S} \oplus \mathcal{K}_2^i)$ and sends it to the contract.
4. $\mathcal{P}$ can then check that $\mathcal{C}^i$ has $\mathcal{S} \oplus \mathcal{K}_2^i$ by verifying that $h^i = H(\mathcal{S} \oplus \mathcal{K}_2^i)$. If it is wrong, $\mathcal{P}$ stops the selling process. Else $\mathcal{P}$ sends $\mathcal{K}_2^i$ to $\mathcal{C}^i$ through the contract , which stores the new current price value $\Delta(t_1^i)$.
5. Finally $\mathcal{C}^i$ now has $\mathcal{K}_2^i$ and can compute $\mathcal{S}$, then $\mathcal{K}$ and eventually I. Moreover, the smart-contract computes and reimburses the price difference $\Delta(t_0^i) - \Delta(t_1^i)$ between the initiation of the exchange (step 1) and its end (step 4), to ensure $\mathcal{C}^i$ has access to the data at the current depreciation value.

These 5 steps lead to a client having access to the encryption key $\mathcal{S}$, meanwhile eavesdroppers can not conclude anything. Note that another client also cannot circumnavigate the logic behind this, by using the same $\mathcal{K}_2^i$ as the first client for example, because steps 2 to 5 need to be done independently for each client, thus the provider generates a different key $\mathcal{K}_2^i$ for each client.

2.3.3 Finalization

If the provider does not reveal $\mathcal{S}$, she will not be able to withdraw his/her earnings and insurance funds, and the clients will be refunded. If the provider does reveal the key, she must wait an additional time. During this time window, any client can choose to set a *dispute* (see next paragraph).

Finally after the fixed time window, the provider can withdraw his remaining unclaimed funds and the remaining funds from the insurance deposit. Note that this operation can only be done once, and only after the time window has passed. No client can set a dispute after the provider withdraws his/her money.

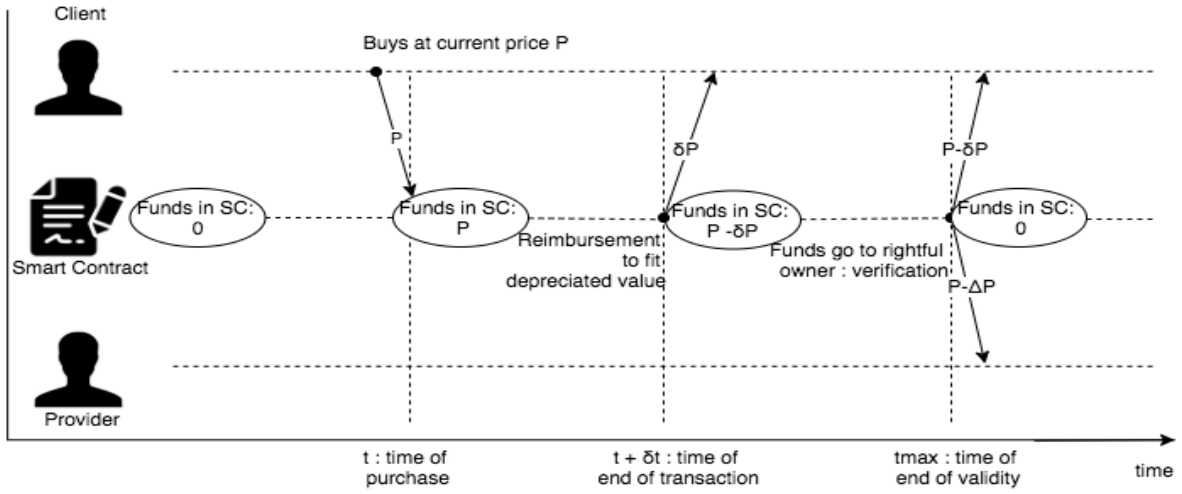

Figure 2: Funds transactions over contract validity.

2.3.4 Dispute and Refund

If client $\mathcal{C}^i$ deems that the seed received does not correspond to $\mathcal{S}$ publicly published by the provider, he calls the contract to raise a dispute (see Alg. 2) by simply specifying the Id bought. The contract checks whether $\mathcal{S}$ was released by the provider or not. Then, the contract must check if a full-on fraud has happened on $\mathcal{S}$ by checking if $H(\mathcal{S} \oplus \mathcal{K}_2^i) = h^i$. The funds paid by the client, and the insurance deposited by the provider (divided by the total number of clients), can be retrieved by the party who is right.

2.4 Security Analysis

The analysis presented in this Section is a summary of the main arguments demonstrating the security of our proposal.

In attack [A1] (see Section 2.2), the funds are transferred to the provider only if she wins the dispute and refund procedure. This is equivalent to the provider finding values of $\mathcal{S}$, $\mathcal{S}'$ and $\mathcal{K}'^i_2$ such that $H(\mathcal{S}' \oplus \mathcal{K}_2^i) = h^i = H(\mathcal{S} \oplus \mathcal{K}_2^i)$, which is implies finding a collision in the hash function. This is not possible from hypothesis [H1]. For attack [A2] to occur, the first possibility is that the client recovers directly I by guessing $\mathcal{S}$. However, this is impossible according to hypotheses [H2] and [H3]. The second possibility is to deduce $H(\mathcal{S})$ from $H(\mathcal{S} \oplus \mathcal{K}_2^i)$ by guessing $\mathcal{K}_2^i$, which is impossible considering hypothesis [H2]. There are no over possibilities to recover I without paying the funds. Attack [A3] can succeed only if the user guesses $\mathcal{S}$ which is impossible in view of hypotheses [H2] and [H3].

2.5 Related Work

Our work is connected to several existing projects in the domains of security, cryptography and blockchains. More specifically, it is related to some well-known problems.

The first one is the *time commitment* (Boneh and Naor, 2000): a user presents a commitment of a hidden data and promises to reveal it before a given time. Solutions were already proposed on Bitcoin (Andrychowicz et al., 2014) and Ethereum (Li and Palanisamy, 2018). In our context, the encrypted data is stored on the blockchain, and thus the smart contract is able to check whether the seed is revealed and whether this seed allows the data decryption. Note that this justifies our choice to store data on-chain, enabling the implementation of reputation systems through data validation in a Dapp.

The second problem is the *fair exchange of digital goods*. A third party is mandatory (Pagnia and Gärtner, 1999) and blockchains can assume this role (Bentov and Kumaresan, 2014). An efficient solution, called Zero Knowledge Contingent Payment protocol, has been proposed (BitcoinWiki, 2016), but a practical and secure solution was not proposed for data exchanges until 2017 (Campanelli et al., 2017). The ingenuity of this solution lies in the fact that the seller and the buyer make offchain exchanges using zero-knowledge (ZK) proofs. One should notice that some smart contracts avoid quite complex ZK computations and are able to manage disputes (Dziembowski et al., 2018). These proposals can not be used directly in our context because we need to manage time related aspects relative to the data depreciation (depreciation function, time of interest of the client, time delay in reception of the decryption key).

Algorithm 1: Exchange protocol.

while *CurrentTime* $<$ *TimeLimit* **do**
  Do in parallel
    Provider uploads data ;
  end
  Do in parallel
    Clients buy data at the current price $\Delta(t_0^i)$;
    for $i \leftarrow 1$ **to** $|Clients|$ **do**
      DH exchange and computation of the secret key $\mathcal{K}_3^i$;
      Provider generates random key $\mathcal{K}_2^i$ offchain;
      Provider emits $\mathcal{S} \oplus \mathcal{K}_2^i \oplus \mathcal{K}_3^i$ through the contract;
      Client i stores $h^i = H(\mathcal{S} \oplus \mathcal{K}_2^i)$ on the contract;
      if h^i *is validated by the provider* **then**
        Provider stores $\mathcal{K}_2^i$ on the contract;
        Smart Contract reimburses the client by the price difference $(\Delta(t_0^i) - \Delta(t_1^i))$ to ensure the data is bought at the current depreciated rate ;
      end
    end
  end
  Wait;
  if *TimeLimit almost reached* **then**
    Provider reveals $\mathcal{S}$;
  end
end
if *CurrentTime* $>$ *TimeLimit* + *DisputeTime* **then**
  Provider withdraws the remaining funds that hasn't been claimed by clients ;
end

The last point considered in the design of our protocol is the (possible) *depreciative value* of some data. Indeed, according to the type of data, their value can decrease more or less quickly. If we follow the example of Trusat data (ConsenSys-Space, 2020), the computation of the localization of a space debris - position and speed vector at a given date - lose accuracy over time, which may render old observations quickly obsolete depending on usage (e.g. in collision avoidance, accuracy is critical). This concept of perishable or depreciating data was recently studied in the context of Internet of Things (Jiao et al., 2018) and in blockchain-based vehicular networks (Xi et al., 2019), but these contexts focus mainly on the determination of the best price of the data. However, our proposal is a contribution to integrate the depreciating aspect of the data in the data exchange protocol.

Algorithm 2: Dispute and refund protocol.

if $\mathcal{K}_2^i$ *is not revealed* **then**
  Refund client with $C^i_{funds} (= \Delta(t_1^i))$;
else if $\mathcal{S}$ *is revealed* **then**
  if $H(\mathcal{S} \oplus \mathcal{K}_2^i) \neq h^i$ **then**
    Refund client with $C^i_{funds} + InsuranceFunds/\#OfClients$;
  else if $ProvidedData < MinData$ **then**
    Refund client with $C^i_{funds} + InsuranceFunds/\#OfClients$;
  end
else if $CurrentTime > TimeLimit$ ***and*** $\mathcal{S}$ *is not revealed* **then**
  Refund client with $C^i_{funds} + InsuranceFunds/\#OfClients$;
end

3 SMART CONTRACT IMPLEMENTATION

The Solidity code of our implementation can be found on Github: https://github.com/ChristianDahdah/Monetization-of-Depreciating-Data-Through-Smart-Contracts

3.1 Gas Price and Gas Limit

One of the main limitations that greatly impact the contract's logic is the gas usage. The use of for-loops with an unknown number of iterations or searching in a table should be avoided in functions that do require gas. Consequently, when a provider $\mathcal{P}$ sets the seed $\mathcal{S}$, the hashes of $\mathcal{S} \oplus \mathcal{K}_2^i$ ($i : 1 \rightarrow$ number of clients) are not automatically compared with the ones provided by each client to settle disputes.

Moreover, selling a product requires the initialisation and storage of a lot of variables: deploy time, product end time, provider's address, initial price, depreciation type, insurance deposit, $\mathcal{S}$, clients' addresses and their respective $\mathcal{K}_2$, hashes and funds. This sets a significant gas price to sell data which needs to be updated often. For this matter it was best seen to group a set of information into one reference, all encrypted with the same $\mathcal{S}$. This would be conve-

nient for both the provider and the client since it implies exchanging less keys and calling less functions.

3.2 Block Validation Time and Transaction Delay

The process of creating a new reference of products, exchanging keys and uploading data, can take several minutes or even hours on Ethereum. Therefore the contract cannot be functional for products that have a really fast depreciation rate, unless the client buys (or subscribes to) the reference beforehand and gets the keys. Afterwards the provider progressively uploads the data as soon as it is available. In this setting the client will be able to access critical information in real time without any delay. At the end of the offer time, if the provider did not upload at least the minimum number **N** of information agreed upon, the client can raise a dispute and claim his funds.

Furthermore, getting the absolute time in a smart contract depends on the block it was mined in. This could raise security problems and multiple exploitation opportunities. For example a malicious node could mine a block and alter the time it was mined, making a dispute in favor of a client instead of a provider. Fortunately block timestamps cannot be tempered a lot. A block with an abnormal timestamp is rejected by the network (Goldberg, 2018). On the long run the timestamp could be offset by a few minutes which could be acceptable.

3.3 Smart Contract Versatility and Organisation

For clarity and ease of use, the protocol mechanism was distributed over three solidity files:

- "Depreciation_Contract.sol" contains mainly all the needed variables for the protocol inside the structure "DataReference".
- "Client_Depreciation_Contract.sol" inherits from the previous contract, and implements all the functions needed for the client.
- "Provider_Depreciation_Contract.sol" inherits from "Client_ Depreciation_ Contract.sol" (consequently from "Depreciation_Contract.sol") and implements all the functions related to the provider, such as creating a new reference, withdrawing funds, setting keys, and viewing the clients' addresses.

To make use of the protocol, it is sufficient to only deploy the "Provider_Depreciation_Contract.sol" **once** and define the contract's interface and address in a fourth contract that contains the data types for our application. In our case we simulated an exchange of space tracking data. The fourth contract "TLE.sol" helps to store Two Line Elements data (TLEs)(Kelso, 2019) which is equivalent to a representation of a space object's location and velocity vector at a given time. The satellite's name (or line 0) is not encrypted and the Two-Line Orbital Element Set Format (lines 1-2) are encrypted and stored in 49 bytes for minimal gas usage. For developing and testing purposes, "TLE.sol" inherits but does not implement the interface of "Provider_Depreciation_Contract.sol". Nevertheless, defining the interface is straightforward and will drastically cut gas usage for deployment (refer to section 4). Hence it is possible to include multiple structures and data types inside one reference. In our case this could be useful to share TLEs along with telescope images using IPFS (Benet, 2014).

4 BENCHMARKS AND RESULTS

All possible scenarios proved to be correctly functional and gas usage was recorded on Ethereum-Client Besu. The initial deployment of the smart contract costs 3 675 449 gas. Note that deployment occurs only once, meaning one common contract can handle selling all future products from any provider for any future application.

The only functions a client needs are detailed in Fig. 3 with their gas costs. Keep in mind that a client only needs to call each function once. This brings the client to a grand total of 181 600 gas (roughly 1.02 USD at the time of writing). But keep in mind that most probably clients will not need to raise a dispute.

On the other hand the provider has to call some functions several times (indicated in fig-4) depending on what she intends to sell and to how many clients. The general case is described in Fig. 4. Furthermore, the gas usage for uploading data varies depending on its length.

Ideally $\mathcal{P}$ can sell information in bulk to multiple clients. All in all, an example of selling 10 TLEs ($\sim$ 96 bytes each) to 50 clients, would cost $\mathcal{P}$ roughly 6 300 000 gas (40.19 USD at the time of writing) which averages out to 126 000 gas per client (0.8 USD at the time of writing). The results seem to compare favorably to other solutions (Dziembowski et al., 2018) which needs to deploy a new contract for each pair provider/client.

Finally, keep in mind that these tests were done for the TLE crowdsourcing usecase. Gas costs can be greatly optimized, a simple idea would be to imple-

ment DH on Elliptic Curves (ECDH) to shorten the keys.

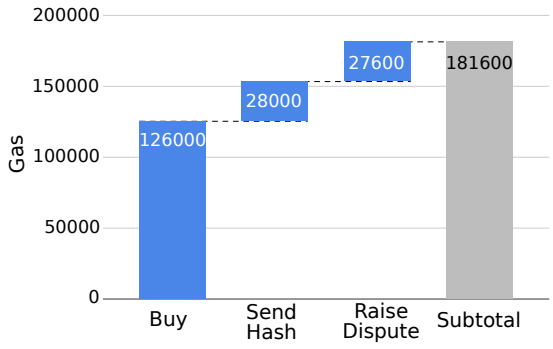

Figure 3: Client Gas Usage on benchmarks.

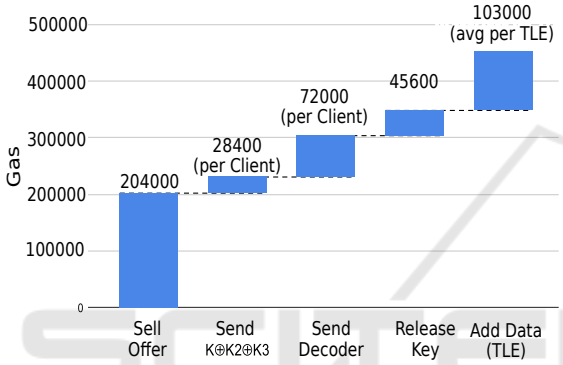

Figure 4: Provider Gas Usage on benchmarks.

5 LIMITATIONS

This protocol is based on the veracity/value of the traded information, in other words a non-honest provider could sell worthless information, a fraud that the smart-contract cannot detect. To counteract this we elucidated the importance of an on-blockchain reputation system.

We should also note that on a permissioned blockchain, gas cost is less of an issue, storage and computation are therefore less restrictive. This can eliminate the need of a PRG since the whole Key can be stored for the exchange of heavy sized data. Moreover, a reputation system is less necessary due to the fact that the providers would be more known and the information's value more trustworthy.

On the other hand, exchanging on the mainnet promotes the reach of a greater range of actors, audience, and information, which is necessary for crowdsourcing purposes. This requires the implementation of the full protocol (i.e. usage of PRGs to cut gas costs) along with a reputation based system to ensure the safety of both parties.

A reputation system could take the form of an automatic (such as in Trusat (ConsenSys-Space, 2020)), on-chain verification (if the nature of the information empowers this such as comparing numbers from one provider to another), or user ratings indicating the honesty of a provider. The penultimate idea justifies the necessity of Xor operations, since complex encryption/decryption methods cannot be used in a smart contract and the decryption must be computed on-chain.

Finally, the whole paper targets volatile and depreciating information because it binds the provider to publicly reveal the Key before a fixed time and thus reveal the published information. Outside this scope, the protocol can be burdensome, and it may be best to use other on-chain exchange protocols.

6 CONCLUSION

This paper introduces a full solution to monetize depreciating data in blockchain-based crowdsourcing systems. The system ensures a fair exchange of digital goods while ensuring that the data will be released freely before a given time. The algorithm described, has been implemented on an Ethereum smart contract, and benchmarked. An example was applied to a space tracking data system and is available on Github[1].

In future work, we plan to extend the concepts and algorithms introduced here, for example to manage larger data files. The formalization of the algorithm and of the security proof will further be developed in future publications.

REFERENCES

Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. (2014). Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458.

Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

Bentov, I. and Kumaresan, R. (2014). How to use bitcoin to design fair protocols. In Garay, J. A. and Gennaro, R., editors, *Advances in Cryptology – CRYPTO 2014*, pages 421–439, Berlin, Heidelberg. Springer Berlin Heidelberg.

BitcoinWiki (2016). Zero knowledge contingent payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment.

Boneh, D. and Naor, M. (2000). Timed commitments. In Bellare, M., editor, *Advances in Cryptology —*

[1]https://github.com/ChristianDahdah/Monetization-of-Depreciating-Data-Through-Smart-Contracts

CRYPTO 2000, pages 236–254. Springer Berlin Heidelberg.

Campanelli, M., Gennaro, R., Goldfeder, S., and Nizzardo, L. (2017). Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proc. of the 2017 ACM SIGSAC Conference on Comp. and Comm. Security*, CCS '17, page 229–243.

ConsenSys-Space (2020). Trusat white paper. https://trusat-assets.s3.amazonaws.com/TruSat+White+Paper_v3.0.pdf.

Dennis, R. and Owen, G. (2015). Rep on the block: A next generation reputation system based on the blockchain. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 131–138.

Dziembowski, S., Eckey, L., and Faust, S. (2018). Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 967–984.

Goldberg, P. (2018). Smart contract best practices revisited: Block number vs. timestamp. https://medium.com/@phillipgoldberg/smart-contract-best-practices-revisited-block-number-vs-timestamp-648905104323.

Jiao, Y., Wang, P., Feng, S., and Niyato, D. (2018). Profit maximization mechanism and data management for data analytics services. *IEEE Internet of Things Journal*, 5(3):2001–2014.

Kelso, T. (2019). Norad two-line element set format. https://www.celestrak.com/NORAD/documentation/tle-fmt.php.

Kogias, D. G., Leligou, H. C., Xevgenis, M., Polychronaki, M., Katsadouros, E., Loukas, G., Heartfield, R., and Patrikakis, C. Z. (2019). Toward a blockchain-enabled crowdsourcing platform. *IT Professional*, 21(5):18–25.

Li, C. and Palanisamy, B. (2018). Decentralized release of self-emerging data using smart contracts. In *37th IEEE Symposium on Reliable Distributed Systems, SRDS 2018, Salvador, Brazil, October 2-5, 2018*, pages 213–220. IEEE Computer Society.

Ma, Y., Sun, Y., Lei, Y., Qin, N., and Lu, J. (2020). A survey of blockchain technology on security, privacy, and trust in crowdsourcing services. *World Wide Web*, 23:393–419.

Pagnia, H. and Gärtner, F. C. (1999). On the impossibility of fair exchange without a trusted third party. Technical report, Darmstadt University of Technology.

Xi, R., Liu, K., Liu, S., Chen, W., and Li, S. (2019). Perishable digital goods trading mechanism for blockchain-based vehicular network. In *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 147–154.