



Title	Towards a Blockchain Contract-for-Difference Financial Instrument for Hedging Renewable Electricity Transactions
Authors(s)	Alao, Olakunle, Cuffe, Paul
Publication date	2020-10-01
Publication information	Alao, Olakunle, and Paul Cuffe. "Towards a Blockchain Contract-for-Difference Financial Instrument for Hedging Renewable Electricity Transactions." IEEE, October 1, 2020. https://doi.org/10.1109/energycon48941.2020.9236436 .
Conference details	The 6th IEEE International Energy Conference, Gammarth, Tunisia (held online due to coronavirus outbreak), 28 September - 1 October 2020
Publisher	IEEE
Item record/more information	http://hdl.handle.net/10197/11728
Publisher's statement	© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Publisher's version (DOI)	10.1109/energycon48941.2020.9236436

Downloaded 2026-05-02 01:12:42

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)



© Some rights reserved. For more information

Towards a Blockchain Contract-for-Difference Financial Instrument for Hedging Renewable Electricity Transactions

Olakunle Alao, *Member, IEEE* and Paul Cuffe, *Member, IEEE*

Abstract—Contract-for-Difference financial instruments are available to renewable electricity generators in day-ahead electricity markets to allow them to hedge against price and volume risk, together known as *revenue risk*. Traditional CfDs while designed to hedge revenue risk, introduce other new risks such as counterparty credit, margining and third-party risks. We therefore propose a novel financial instrument – an Ethereum blockchain-based dual escrow smart contract, to serve as the mediator in a CfD agreement between a renewable electricity generator and supplier. This financial instrument addresses hedging related risks that result from traditional CfD agreements in day-ahead electricity markets. In this paper, we design the logic of the financial instrument, translate this logic to smart contract codes and demonstrate its expected performance. Overall, the proposed financial instrument has the benefits of reducing hedging related risks inherent in traditional CfDs. Likewise, it enables secure, efficient, cost-effective, consistent, reliable, transparent and frictionless transactions between contracting parties in a CfD agreement.

Key words—Blockchain, Smart contracts, Renewable electricity, and Electricity derivatives

I. INTRODUCTION

THE diminishing levels of fossil fuels, as well as greenhouse gas emissions that accompany their exploitation, has elicited the need for renewable electricity. The world’s abundant renewable electricity resource is significantly untapped, as only about 25% of global energy demand is met by renewable electricity. With global energy demand expected to continue to exponentially increase, more investment in renewable electricity is therefore required in order for clean and green supply to keep pace with demand [1].

Revenue risk is the major risk posed to renewable electricity generators and can adversely affect the bankability of renewable electricity projects. Debt often accounts for the highest share of capital required to finance renewable electricity projects [2], [3]. Risk-averse debt financiers assess the bankability of a project before making investment decisions, and high-risk projects usually incur expensive finance (i.e. debt at high-interest rates). A bankable renewable electricity project is one where debt interest and principal repayments are feasible because of low revenue risk exposure [3]. Hedging against renewable

electricity generator’s revenue risk is therefore paramount, as it enhances the bankability of renewable electricity projects; hence, unlocking more debt capacity at low interest rates. The revenue of a typical renewable electricity generator in a day-ahead electricity market is mathematically shown in Equation (1):

$$\sum_{t=1}^{t=n} \left((C_t^{pm} \times P_t^{pm}) + (C_t^{fm} \times P_t^{fm}) \right) \quad (1)$$

In Equation (1), C is the contracted capacity in MWh and P is the price per MWh. Superscript pm represents the physical market and fm represents the financial market. A renewable electricity generator will primarily participate in the physical market, to generate revenue to cover its marginal cost of electricity production. The physical market is however highly volatile with respect to price and volume. A rational renewable electricity generator will therefore secure its position in the financial market, by hedging itself against price fluctuations on a trading period [4]. A two-way CfD is an electricity derivative financial instrument available in day-ahead electricity markets, that guarantees stable revenues for renewable electricity generators. CfD contracting parties (a renewable electricity generator and typically a supplier) are obliged to pay or receive the difference between the strike price and spot price on a trading period [4]. The financial market (second term) in Equation (1) can then be expressed as:

$$\sum_{t=1}^{t=n} \left(C_t^{st} \times \max(0, P_t^{st} - P_t^{sp}) \right) \quad (2)$$

In Equation (2), C and P retain their definitions as Equation (1), and superscripts st and sp represent the strike price and spot price respectively. Equation (2) is the CfD payoff per MWh that is due the generator, when the strike price is greater than the spot price. If the strike price is less than the spot price, then the payoff per MWh becomes: $\max(0, P^{sp} - P^{st})$, and the generator becomes obliged to pay the supplier the difference [5], [6].

CfDs like other bilateral contracts are traded over-the-counter [5]. They are therefore susceptible to counterparty credit, margining and third-party risks; slow settlement time; overhead costs; high collateral requirements; bilateral friction; and human errors [2], [5], [6]. We address these problems by proposing a novel financial instrument – SHARK v1.0 (*Smart Hedging to Ameliorate Risk*) – a blockchain-based dual escrow smart contract, that serves as the mediator of an escrow in a

This publication has been funded by the Sustainable Energy Authority of Ireland under the SEAI Research, Development & Demonstration Funding Programme 2018, grant number 18/RDD/373.

O. Alao and P. Cuffe (paul.cuffe@ucd.ie) are with the School of Electrical and Electronic Engineering, University College Dublin.

Interests disclosure: The Authors hold cryptographic assets.

CfD agreement between a renewable electricity generator and supplier.

Although earlier blockchain applications were limited to the banking and finance industry, modern blockchains enabled by smart contracts are slowly gaining traction in other industries, such as supply chain management, utilities, education, health care, etc [7]–[9]. These new use cases have also been facilitated by *stable coins*, which are digital assets whose underlying value is pegged to relatively stable fiat currencies, unlike highly volatile traditional cryptocurrencies [10].

Blockchain has been proposed for applications in the electricity sector: electricity trading, financing of renewable electricity projects, green certificates, carbon trading, smart grids, electric vehicles, etc. [8], [11]–[16]. The greatest number of blockchain applications in the electricity sector so far has been in the retail market. Decentralised peer-to-peer electricity trading at the retail market can be facilitated by blockchain connected smart meters and tokenisation of electricity on modern blockchain networks [8], [12], [13]. Blockchain may also disrupt the traditional way renewable electricity projects are financed, through Initial Coin Offerings [8], [15], [16], whose financing structure is similar to Initial Public Offerings.

Blockchain has been proposed in financial derivatives and securities market [17]–[21], but is yet to be explored in the electricity derivatives sector. Given the volatility of the day-ahead physical market, renewable electricity generators have to apply the lowest risk, efficient and cost-effective hedge. Considering blockchains already developed use cases in financial derivatives [17]–[20] and for bilateral wholesale and retail energy trading [12]–[14], [22], it is apparent that this technology could lend itself to reducing hedging related risks in electricity derivative transactions. Moreover, at the time of writing, no industry blockchain project or academic literature speaks to bespoke blockchain-based smart contracts for electricity derivative transactions. This gap therefore reveals the untapped potential of blockchain in electricity derivatives, and it is hoped that this paper kick-starts discussions around this salient topic.

II. BLOCKCHAIN AND SMART CONTRACTS

A. Blockchain fundamentals

A blockchain is an open, decentralized, distributed, immutable, transparent and consistent digital ledger. It records transactions, assets and data between participants in a verifiable and permanent manner [8], [9], [23]. Blocks in a blockchain consists of several transactions that are chronologically appended to each other via a cryptographic hash function. The main features of a blockchain are its data structures, consensus rules, cryptographic security and incentive mechanisms [7]–[9], [23]. In typical blockchains like Bitcoin and Ethereum, when a transaction is completed between participants, it is broadcasted to all mining nodes on the network. Mining nodes gather the transaction between these participants, and other transactions into a block. Thereafter, mining nodes compete to add this block to the blockchain by attempting to find a solution to a difficult *proof-of-work* problem, which is computationally and energy intensive. When a mining node finds the proof-of-work, it broadcasts the block to all other mining nodes on the network.

These mining nodes accept the block if all the transactions in it are valid. When more than 50% of the mining nodes on the network accept this block, this block is added to the blockchain. Mining nodes begin to work on creating the next block on the blockchain, including the cryptographic hash of the recently accepted block as the previous hash. Non-mining nodes, on the other hand, support the decentralization of the network by holding and distributing copies of the blockchain ledger [7]–[9], [23].

B. Smart contracts

Even though the two most popular blockchains – Bitcoin and Ethereum – have a similar design with respect to consensus and cryptographic rules, they have different objectives. While Bitcoin was proposed by Satoshi Nakamoto [23] for a single application – that is to serve as a secure peer-to-peer decentralized payment system; Ethereum, championed by Vitalik Buterin [21] was designed to support many applications other than payment. Ethereum, unlike Bitcoin supports any decentralized application (dApp) and is *Turing Complete* – implying that it can be programmed to solve any computational problem of arbitrary complexity. The business logic of these decentralized applications are embedded in *smart contracts*, which are complex set of software codes written on the blockchain with the ability to autonomously manage transactions [8], [9], [21], [24]. Smart contracts can seamlessly handle funds between different participants based on pre-agreed conditions, and when these conditions are met, the smart contract automatically pays the appropriate participants. In its simplest form, they are a translation of contract laws into computer protocols, albeit they provide security that surpasses classical contract laws. This additional layer of security is because transactions that emanate from smart contracts are verified and mined by the same rules that underpins transactions between externally owned accounts on a blockchain [21], [24].

Smart contracts change the state of Ethereum through the *Ethereum Virtual Machine* (EVM), which can be viewed as a global computer that cannot be interfered with. They are typically developed using *Solidity*, a contract-oriented high-level programming language. The EVM only understands *bytecode*, so the high-level Solidity code is translated to EVM executable bytecode by the Solidity compiler [21], [24]. Ethereum’s smart contracts are deployed on the blockchain via a transaction that submits their EVM executable bytecode on the blockchain. After deployment, they possess an Ethereum address and their code and subsequent transactions become accessible to the public on the blockchain. Thereafter, the smart contracts run in the EVM, whenever a transaction is sent to its address. Every transaction, including the contract deployment transaction incurs a fee called *gas*, which is payable to the applicable mining node on the network [8], [9], [21], [24]. In the Ethereum network, miners are rewarded for processing transactions between externally owned accounts and executing smart contracts [7], [24]. Smart contracts can be invoked either through transaction or calls. While transactions alter the state of the blockchain and incur gas, calls allow state reading and are free. Transactions sent to the smart contract address can contain Ether, data or both. Transactions containing Ether lodges Ether to the smart contract; whereas, transactions containing data usually serve as an

argument to a function in the smart contract [21], [24]. Overall, Ethereum’s ecosystem is flexible and mature, which is why its smart contract technology will be leveraged on for the rest of the paper.

III. METHODOLOGY

As already discussed, traditional CfDs hedges against the revenue risk exposure of renewable electricity generators, but themselves introduces other new risks. In this section, we define the business logic of SHARK that addresses the hedging related risks posed by traditional CfDs. Developing smart contracts can be very challenging since they must retain the same legal, technical and commercial functionality as contract laws [20], while being cheat-proof. This illuminates the importance of the business logic – the *brain* of the smart contract, where all the game-theoretic incentive mechanisms that encourages rational behaviour are housed.

In a broad sense (see Fig. 1), SHARK is designed to be a dual escrow bilateral smart contract, such as in [18]. It allows both contracting CfD parties to deposit into and withdraw from their respective escrow accounts. When exposed to a specific input that serves as a trigger, SHARK autonomously pays either party based on pre-agreed conditions. SHARK is designed to incentivize a neutral but trustless party to regularly invoke functions of the contract at certain time windows. Access to all functions is restricted so that only permitted parties can call certain functions. Contracting parties are motivated to remain in the contract to hedge their revenue risk, and maintain their escrow accounts to avoid liquidation and incurring a termination penalty.

A. Assumptions

1) *Low volatility*: The volatility problem of Ether has now been resolved by the introduction of stable coins (e.g. the Dai stable coin [10]) into the Ethereum ecosystem. Dai, developed by the Maker team, is a collateral-backed cryptocurrency whose value is softly pegged to the US Dollars (USD). Dai can be generated in exchange for Ether, through Maker – a smart contract on Ethereum, whose business logic is built on decentralized finance game-theoretic principles to maintain a 1:1 pegging with USD [10]. For convenience, we maintain that 1 Ether = €200, and we use this exchange rate all through the rest of this paper, even though DAI would be used in practise.

2) *Oracle service*: The spot price and strike price are required to determine the CfD payoff at every trading period. Although the strike price is fixed and registered on-chain following SHARK’s deployment, the spot price fluctuates at every trading period and is published off-chain. Unfortunately, smart contracts are unable to autonomously access off-chain data. This is where the role of Oracles come about, as they are used to provide external data that can trigger pre-defined actions of the smart contract. Oracles can be either software, via online sources that send data automatically to the smart contract; or hardware, via participants who are granted access to input data that serve as arguments to certain functions of the smart contract [24]. Oracles require trust [7], [24], and in this paper, we assume that a Market Operator has been delegated to serve as an Oracle in SHARK.

B. Incentives

Given that smart contract functions are not self-invoking and incurs gas (infrastructure fee) for every call, both contracting parties agree to allow a neutral but trustless party to invoke some of the functions of SHARK, for a small bounty. Like any derivative financial instrument, SHARK requires a maintenance margin to limit counterparty credit risk exposure. Its margin rules are in the form of a bilateral American option [19], since any of the contracting parties can exit the contract any time they wish. Therefore, a termination penalty premium is introduced to disincentivize counterparties from terminating the contract or refusing to refill their escrow account. The design problem becomes deriving the minimum required maintenance margin that removes counterparty credit risk; and the termination penalty that encourages contracting parties to remain in the contract and maintain their escrow accounts. Equation (3) shows the minimum required maintenance margin for the generator:

$$\sum_{t=1}^{t=n} \left(M_t^m \right) \geq \sum_{t=1}^{t=n} \left(C_t^{st} \times \max(0, P_t^{st} - P_t^{sp}) \right) \quad (3)$$

In Equation (3), M_t^m is the maintenance margin; while C and P, st and sp , retain their definitions as in Equation (1) and Equation (2) respectively. From Equation (3), we can assert that the maintenance margin must be equals to or greater than the CfD payoff per trading period. The maximum payoff at every trading period occurs when: $sp = \pm\infty$. We can hence make a realistic assumption that $\min sp = 0$ and $\max sp = (st \times 2)$. On the other hand, the termination penalty can be chosen probabilistically, based on many factors such as the jurisdiction, termination and default probability of contracting parties, etc [19]. However, it must be sufficient to deter either party from exiting the contract.

C. Functions

1) *Account*: SHARK provides two escrow accounts for both the generator and supplier. Both parties alone are able to withdraw money from their respective escrow account up to the maintenance margin. For demonstration, the following Solidity code snippet shows the Deposit and Withdraw function of the generator:

```
function generator_Deposit () public onlyGenerator payable {
    account_Balance [generator] += msg.value;
}
function generator_Withdraw (uint amount) public onlyGenerator payable {
    require (account_Balance [generator] >= maintenanceMargin);
    account_Balance [generator] -= amount;
    msg.sender.transfer(amount); }
```

2) *Difference*: The Difference payment function is called by a trustless party who is unable to affect the outcome of SHARK but is financially incentivized to invoke the function once every hour. This incentive mechanism is critical to the operation of decentralized networks, such as in [10]. For every transaction, the trustless party is paid a bounty, we term *trustless party difference fee*. This fee is higher than the typical gas cost of the transaction and is equally borne by both contracting

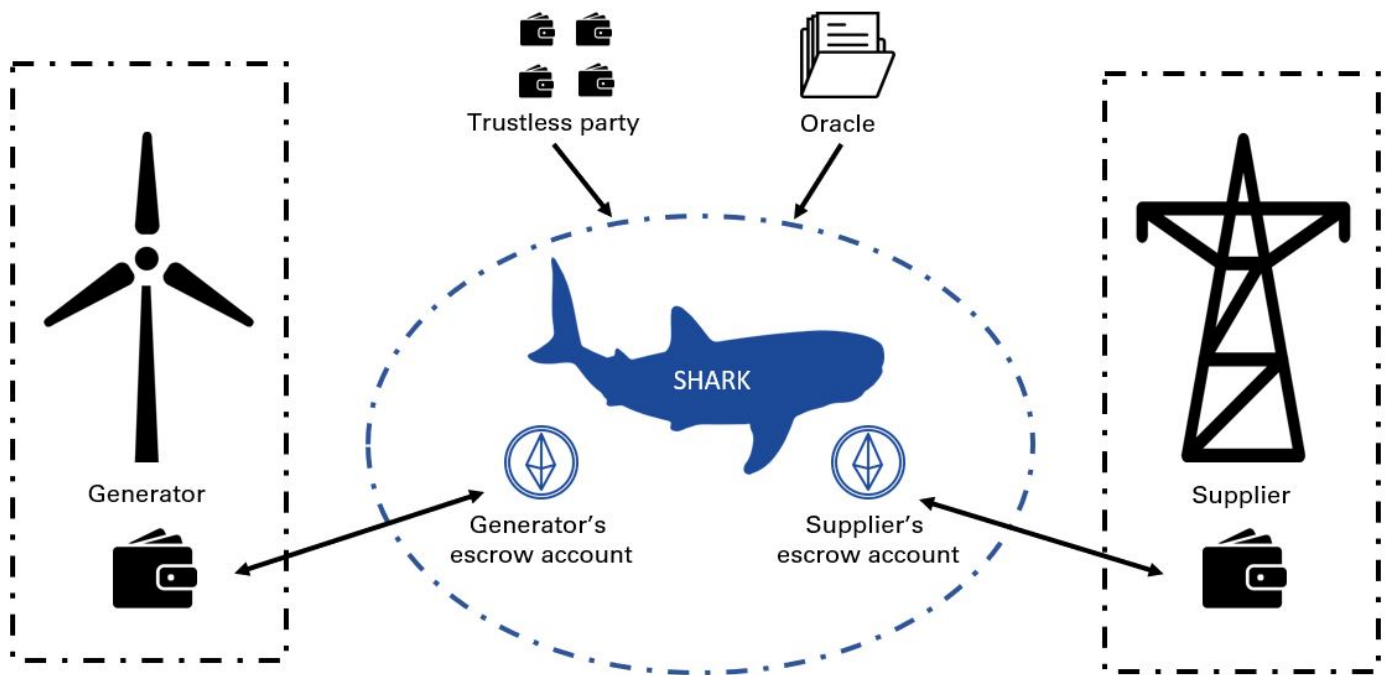


Fig. 1. SHARK v1.0: A dual-deposit escrow smart Contract-for-Difference

parties. Three things happen when this function is invoked. First, the spot price is obtained from the Oracle. Thereafter, SHARK computes the difference between the spot and strike price. The difference is then transferred to the escrow account of the appropriate party. See the following Solidity code snippet:

```
function difference_Payment(uint spotPrice) public
  onlyTrustlessParty payable {
  require (now >= lastWithdrawTime[msg.sender] +
    tradingPeriodTime);
  account_Balance [generator] -=
    trustlessPartyDifferenceFee/2;
  account_Balance [supplier] -=
    trustlessPartyDifferenceFee/2;
  msg.sender.transfer(trustlessPartyDifferenceFee);

  if (strikePrice > spotPrice) {
    uint diff = (strikePrice - spotPrice) *
      contractedCapacity;
    account_Balance [supplier] -= diff;
    account_Balance [generator] += diff;
  }
  else if (strikePrice < spotPrice) {
    uint diff = (spotPrice - strikePrice) *
      contractedCapacity;
    account_Balance [generator] -= diff;
    account_Balance [supplier] += diff;
  }
  else { revert (); }
  lastWithdrawTime[msg.sender] = now; }
```

3) *Close*: The Close function can be called by the contracting parties or trustless party. Either of the contracting parties will call this function if they intend to exit SHARK. When this function is invoked, SHARK closes but before then, it transfers the maintenance margin to the exiting party, and the balances of both escrow accounts (if any) and termination penalty to the non-exiting party. This ultimately discourages a rational party

from terminating SHARK. The trustless party, on the other hand, is financially incentivized to invoke this function for a bounty called *trustless party close fee*, when any of the escrow accounts is below the maintenance margin. Otherwise, it loses Ether due to gas costs and SHARK reverts to its initial state. If the trustless party invokes this function due to either party failing to meet their maintenance margin requirement, a bounty is paid to its account, while the balances of both escrow accounts and termination penalty are transferred to the non-defaulting party. See the following Solidity code snippet:

```
function close_Contract() public payable {
  if (msg.sender == generator){
    msg.sender.transfer(maintenanceMargin);
    selfdestruct(supplier);
  }
  else if (msg.sender == supplier){
    msg.sender.transfer(maintenanceMargin);
    selfdestruct(generator);
  }
  else if (msg.sender == trustlessParty){
    if (account_Balance [generator] < marginAndPenalty
      && account_Balance [supplier] >=
      marginAndPenalty) {
      msg.sender.transfer(trustlessPartyCloseFee);
      selfdestruct(supplier);
    }
    else if (account_Balance [supplier] <
      marginAndPenalty && account_Balance [generator
    ] >= marginAndPenalty) {
      msg.sender.transfer(trustlessPartyCloseFee);
      selfdestruct(generator);
    }
    else if (account_Balance [supplier] <
      marginAndPenalty && account_Balance [generator
    ] < marginAndPenalty) {
      selfdestruct(msg.sender);
    }
  }
}
```

```
else { revert (); } } else { revert (); }
```

D. Access restriction and Security

Access is restricted in SHARK so that only permitted parties can call certain functions. SHARK runs on Ethereum, which is a public blockchain, so in spite of the fact that we restrict access, transactions are publicly recorded. We restrict access in SHARK, through the `require` object. For recurring and function-wide access restriction requirements, the `require` object was wrapped inside a *function modifier* and re-used, such as the `onlyGenerator`, `onlySupplier` and `onlyTrustlessParty` modifiers in Subsection III-C. SHARK is also resistant to a popular vulnerability called – “*re-entrancy bug*” – which results when a malicious smart contract tricks an exploitable smart contract by recursively invoking functions of the smart contract [24]. SHARK’s resistance to re-entrancy is because access control per function only allows interaction with the contracting parties and trustless party. Moreover, there is no function that sends Ether to the public; thus, making it impossible for a malicious contract to use a *fallback* function to re-enter SHARK. Overall, we ensured that the state of SHARK was fully updated on the blockchain before interacting with any party’s public address. Although there are many other security risks and vulnerabilities that are plausible in Ethereum’s smart contract ecosystem which we have already taken cognizance of (see [24] for details), the re-entrancy bug is the most popular.

IV. RESULTS AND DISCUSSION

A. Case study

Consider a wind generator on the island of Ireland with an installed capacity of 110 MW. On a trading day, the wind generator offers 100 MWh of energy into the day-ahead physical market at a price of 80€/MWh. It also holds a 100 MWh SHARK contract with an electricity supplier at a strike price of 80€/MWh to hedge against the spot price volatility on a trading day as shown in Fig. 2. The incentive mechanism built into SHARK includes: a maintenance margin of €8,000, a termination penalty premium of €4,000, a trustless party difference fee of €4 and a trustless party close fee of €1,000.

The wind generator starts the trading day with an escrow balance of €20,000. On trading period 1 (00:00), the spot price per MWh is €76. In this case, SHARK automatically pays the wind generator the product of the contracted capacity and the difference between the spot and strike price, less the trustless party difference fee of €2. The balance of the trustless party difference fee (i.e. €2) is simultaneously borne by the supplier. The total amount transferred to the generator’s escrow account by SHARK on trading period 1 is equals to €398. Its escrow account balance after trading period 1 (00:00) becomes €20,398.

The net difference payment in the first seven trading periods as shown in Fig. 3 is significantly in favor of the wind generator. This led its account to grow much higher than SHARK’s required maintenance margin. Given its healthy escrow account, the generator decides to withdraw €10,000 from its account on trading period 7 (6:00). Later on the same trading day, the net difference payment began to favor its counterparty – the

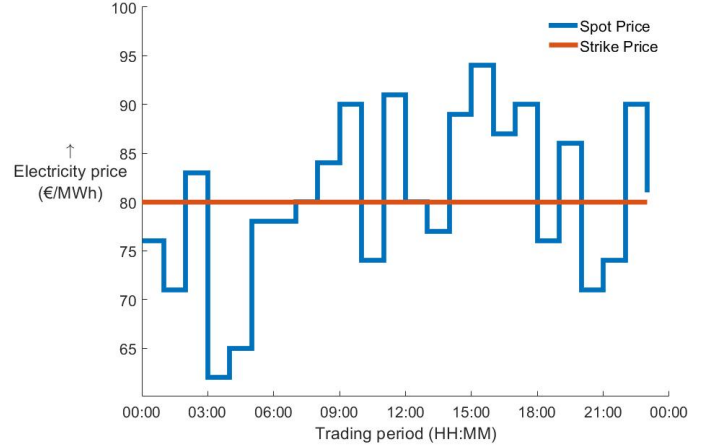


Fig. 2. Spot prices on a trading day (00:00 to 23:00)

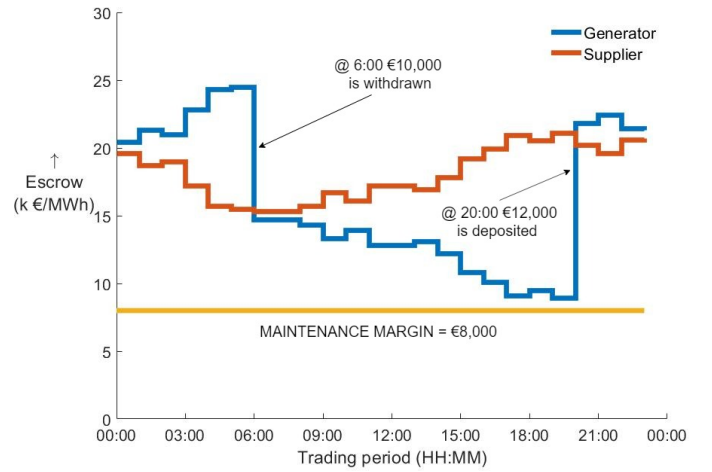


Fig. 3. SHARK’s escrow accounts on a trading day (00:00 to 23:00)

electricity supplier. To avoid liquidation of its escrow account, which will automatically occur when its escrow account balance goes below the maintenance margin (i.e. €8,000), it deposits €12,000 on trading period 21 (20:00). It ends the trading day with a healthy escrow account balance of €21,300. SHARK continues to pay out to the appropriate party this way, until any of the contracting parties exits the contract or default on the maintenance margin.

B. Discussion

1) *Collateral requirements and Margining risk*: Counterparty credit risk is the greatest concern for parties entering a CfD and is usually difficult to hedge since it is inherently part of a CfD agreement. Collateral (or margin) requirements are therefore imposed to limit this risk exposure [2], [3], [6], [25]. In traditional CfDs, collateral requirements are very high since settlement typically occurs monthly or longer [25]. However, since SHARK settles every hour, the possible counter-party exposure per time is limited and thus, the collateral requirements

are significantly lower. This collateral requirement reduction is significant and directly affects margining risk – the risk that future cash flows are lower because of maintenance margin payments [6]. This implies that with SHARK, investors would not only hedge counterparty credit risk, but will increase their liquid assets per time.

2) *Hedging cost*: SHARK presents significant cost reduction and increased efficiency over the whole CfD transaction cycle. In fact, the only cost that is incurred is a meagre fee called gas cost, which is the infrastructure fee of the Ethereum network and is used to create incentives for miners to process transactions in the Ethereum blockchain. Unlike traditional CfDs, SHARK eliminates the costs incurred by counterparties due to clearing and settlement, and other overhead costs such as, audits, enforcement & compliance, and confirmations. Another non-recurring but possible cost that SHARK removes, is the cost of potentially long and arduous arbitration processes that could arise due to disputed payments or payment defaults [25]. Arbitration processes are non-existent in SHARK because it is not susceptible to fraud or human errors, like traditional CfDs where paper contracts and backend processes are the norm. Transactions are frictionless, and pre-agreed terms are written in stone and are made available through SHARK’s open-source code on the blockchain.

3) *Consistency, transparency and security*: SHARK’s operation remains consistent forever until it is exited by any of the contracting parties. It also does not have a single point of failure as its transactions and data are decentralised. Transactions that emanate via SHARK are immutable forever on the blockchain and can be referred to by any party in the future. As far as possible, SHARK is designed to be secure and cheat-proof against any malicious actor who intends to gain access to the contract’s funds.

V. CONCLUSION

This paper has demonstrated the feasibility of using blockchain for electricity derivatives and has shown that a well-designed business logic for a blockchain-based smart contract financial instrument can reduce hedging related risks that arise due to traditional Contract-for-Difference agreements. Altogether, the proposed SHARK prototype will result in lower cost of obtaining finance for renewable electricity developers. It will also increase the value of renewable electricity generators’ shares, since they will become more creditworthy ventures. There are some aspects of SHARK that require further research and development, such as the incentive mechanisms, Oracle and stable coin features. However, it is hoped that future iteration of SHARK will be sufficiently robust to become the standard CfD financial instrument for renewable electricity generators in day-ahead electricity markets.

REFERENCES

- [1] The International Energy Agency, “Renewables 2018: Market analysis and forecast from 2018 to 2023,” Tech. Rep., 2019.
- [2] Clifford Chance, “Contracts for Difference: an EMR CfD Primer,” Tech. Rep., 2015, pp. 1–7.
- [3] Baringa Partners LLP, “Forward hedging under I-SEM,” Tech. Rep., 2016, pp. 1–29.
- [4] Eirgrid, “Industry Guide to the Integrated Single Electricity Market The I-SEM Project,” Tech. Rep., 2017, pp. 1–97.
- [5] S. J. Deng and S. S. Oren, “Electricity derivatives and risk management,” *Energy*, vol. 31, no. 6-7, pp. 940–953, 2006, issn: 03605442. doi: 10.1016/j.energy.2005.02.015.
- [6] J. Hull, *Options, Futures and Other Derivatives*. 2009, ISBN: 0135009944. doi: 10.1007/978-1-4419-9230-7_2.
- [7] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. 2014, ISBN: 9781449374044.
- [8] M. Andoni et al., “Blockchain technology in the energy sector: A systematic review of challenges and opportunities,” *Renewable and Sustainable Energy Reviews*, vol. 100, pp. 143–174, 2019, issn: 18790690. doi: 10.1016/j.rser.2018.10.014.
- [9] K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016, issn: 21693536. doi: 10.1109/ACCESS.2016.2566339.
- [10] Maker Team, “The Dai Stablecoin System,” Tech. Rep., 2017.
- [11] A. S. Musleh, G. Yao, and S. M. Mueen, “Blockchain Applications in Smart Grid–Review and Frameworks,” *IEEE Access*, vol. 7, pp. 86 746–86 757, 2019. doi: 10.1109/access.2019.2920682.
- [12] M. T. Devine and P. Cuffe, “Blockchain electricity trading under demurrage,” *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 2323–2325, 2019, issn: 1949-3053. doi: 10.1109/TSG.2019.2892554.
- [13] M. Pipattanasomporn, M. Kuzlu, and S. Rahman, “A blockchain-based platform for exchange of solar energy: Laboratory-scale implementation,” pp. 1–9, 2018. doi: 10.23919/ICUE-GESD.2018.8635679.
- [14] E. Mengelkamp, J. Garttner, K. Rock, S. Kessler, L. Orsini, and C. Weinhardt, “Designing microgrid energy markets: A case study: The Brooklyn Microgrid,” *Applied Energy*, vol. 210, pp. 870–880, 2018, issn: 03062619. doi: 10.1016/j.apenergy.2017.06.054.
- [15] Sun Exchange, *The Sun Exchange*. [Online]. Available: <https://thesunexchange.com/>.
- [16] C. Hahn and A. F. Wons, “Initial Coin Offering,” *Finanzierung von Start-up-Unternehmen*, pp. 237–251, 2018. doi: 10.1007/978-3-658-20642-0_9.
- [17] Ryan Surujnath, “Off The Chain! A Guide to Blockchain Derivatives Markets and the Implications on Systemic Risk,” *Journal of Corporate & Financial Law*, vol. 22, no. 2, pp. 257–304, 2017.
- [18] A. Asgaonkar and B. Krishnamachari, “Solving the Buyer and Seller’s Dilemma : A Dual-Deposit Escrow Smart Contract for Provably Cheat-Proof Delivery and Payment for a Digital Good without a Trusted Mediator,” pp. 1–7, 2018. arXiv: arXiv:1806.08379v1.
- [19] C. P. Fries and P. Kohl-landgraf, “Smart Derivative Contracts: Detaching Transactions from Counterparty Credit Risk,” pp. 1–22, 2018.
- [20] ISDA, “Legal Guidelines for Smart Derivatives Contracts,” 2019.
- [21] V. Buterin, “Next generation smart contract & decentralized application platform,” no. January, pp. 1–36, 2013.
- [22] BTL, “Powered by Blockchain: Reinventing Information Management in the Energy Enterprise,” Tech. Rep., 2018.
- [23] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
- [24] A. M. Antonopoulos and G. Wood, *Mastering Ethereum*. 2018, ISBN: 9783540773405.
- [25] ESB, “Master Contract for Difference Agreement Between Electricity Supply Board and [Buyer] Being a PSO-Supported Contract Issued on [Date],” Tech. Rep., 2013.