Ethereum: State of Knowledge and Research Perspectives

Sergei Tikhomirov(✉)

SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
sergey.s.tikhomirov@gmail.com

Abstract. Ethereum is a major blockchain-based platform for smart contracts – Turing complete programs that are executed in a decentralized network and usually manipulate digital units of value. A peer-to-peer network of mutually distrusting nodes maintains a common view of the global state and executes code upon request. The stated is stored in a blockchain secured by a proof-of-work consensus mechanism similar to that in Bitcoin. The core value proposition of Ethereum is a full-featured programming language suitable for implementing complex business logic.

Decentralized applications without a trusted third party are appealing in areas like crowdfunding, financial services, identity management, and gambling. Smart contracts are a challenging research topic that spans over areas ranging from cryptography, consensus algorithms, and programming languages to governance, finance, and law.

This paper summarizes the state of knowledge in this field. We provide a technical overview of Ethereum, outline open challenges, and review proposed solutions. We also mention alternative smart contract blockchains.

Keywords: Blockchain · Ethereum · Smart contracts
State of knowledge

1 Introduction

Bitcoin [Nak08] is the first fully decentralized digital currency introduced in 2008 and launched in 2009. It innovatively combines cryptographic techniques with economic incentives to make rational participants likely to play by the rules. Bitcoin gained significant traction, reaching $80 billion market capitalization in September 2017. Hundreds of alternative cryptocurrencies based on similar general design have appeared since Bitcoin's launch. Programming languages in early blockchains, e.g., the Bitcoin scripting language, were deliberately limited to reduce complexity for the sake of security.

Ethereum [VB+14, Woo14], announced in 2014 and launched in 2015, aims at creating a universal blockchain-based application platform. It incorporates a Turing complete language, making it theoretically possible to express all practical computations in *smart contracts* – pieces of code permanently stored on the

A. Imine et al. (Eds.): FPS 2017, LNCS 10723, pp. 206–221, 2018.
https://doi.org/10.1007/978-3-319-75650-9_14

blockchain and capable of responding to users' requests. This enhanced functionality introduces new security challenges related to language design and secure programming practices.

Ethereum is not the only smart contract blockchain system [BP17]. Ethereum Classic [Eth17c] is an alternative blockchain originating from a controversial Ethereum update. Rootstock [Roo17] and Qtum [Qtu17] aim at implementing smart contracts in combination with the Bitcoin blockchain. Chain [Cha17a], Corda [Cor17], and Hyperledger [Hyp17] propose permissioned (i.e., with a fixed set of approved participants) smart contract blockchains, designed to simplify transactions between corporate entities.

This paper focuses on Ethereum as the most mature open blockchain with Turing complete programming capabilities. We summarize the state of knowledge and outline the research perspectives in this rapidly developing field. We assume familiarity with the basic blockchain concepts; [BMC+15, TS15] provide the necessary background.

2 Technical Overview

2.1 State and Accounts

Ethereum can be thought of as a state machine. Nodes of the Ethereum peer-to-peer network maintain a shared view of the global state. A user interacts with the network by issuing a transaction representing a valid state transition. Nodes pick transactions from the *mempool* (the set of unconfirmed transactions), verify their validity, perform the corresponding computation (possibly changing ownership of units of the Ethereum native cryptocurrency *ether*), and update the state. There are two types of accounts in Ethereum: *externally owned accounts* and *contract accounts* controlled by a private key or by a smart contract – a piece of code deployed on the blockchain – respectively.

The account state consists of the following fields:

- *nonce* – the number of transactions sent by this account (for externally controlled accounts) or the number of contract creations made by this account (for contract accounts);
- *balance* – the number of *wei*[1] owned by this account;
- *storageRoot* – Merkle Patricia tree root of this account's storage;
- *codeHash* – hash of this account's contract bytecode.

Accounts' 160-bit addresses[2] are derived from its public key or, in case of contract accounts, from the address of the contract's creator and its nonce [eth16]. The global state maps addresses to account states. The primary data structure in Ethereum is the Merkle Patricia tree – a radix tree optimized for key-value mappings with 256 bit keys [VBR+17, Buc14]. The root hash authenticates the whole data structure. Values pairs are editable in logarithmic time.

[1] Smallest denomination of ether: 1 ether = 10^{18} wei.
[2] Addresses are usually written in hex with a `0x` prefix.

The Ethereum state model (accounts and states) differs from than in Bitcoin. The Bitcoin blockchain stores unspent transaction output (UTXO); balances of addresses are calculated off-chain by wallet software.

2.2 Transactions and Gas

The halting problem – determining if a given program will ever halt – is unsolvable in the general case [Chu36]. This poses a challenge: nodes running the Ethereum virtual machine (EVM) cannot foresee the amount of resources required for validating a transaction, which enables denial-of-service attacks.

To overcome the issue, the Ethereum protocol incorporates a pricing mechanism. It makes resource-intensive computations in smart contracts economically infeasible. Every computational step in EVM is priced in units of *gas*. EVM opcodes and their gas costs are defined in the Yellow paper [Woo14]. The price of a gas unit in ether is determined by the market. For every transaction, the sender specifies the maximum amount of gas that the intended computation is expected to consume (the *gas limit*) and the price the user wishes to pay per unit of gas (the *gas price*). The transaction fee equals the gas limit multiplied by the gas price. If the execution is successful, the remaining ether is refunded. If an error occurs, the transaction has no effect on the state, but all provided gas is consumed. Miners can vote to gradually change the limit on the total amount of gas consumed in a block [jnn15].

A transaction is a signed data structure comprising a set of instructions to be atomically executed by the EVM. It consists of the following fields:

- *nonce* – the number of transactions sent by the sender;
- *gasPrice* – the number of wei per gas unit that the sender is paying;
- *gasLimit* – the maximum amount of gas to be spent during execution;
- *to* – the destination address (`0x0` for contract creation transactions);
- *value* – the number of wei transferred along with the transaction;
- v, r, s – signature data.

There are two types of transactions in Ethereum. A *contract creation transaction* is used to deploy a new contract. It contains an additional *init* field that specifies the EVM code to be run on contract creation, as well as the EVM code of the new contract. A *message call transaction* is used to execute a function of an existing contract (with arguments specified by the an optional *data* field) or to transfer ether.

2.3 Block Structure and Mining

Ethereum uses proof-of-work (PoW): nodes compete to find a partial collision of a cryptographic hash function and produce the next block[3]. Both Bitcoin [Wui17] and Ethereum [Joh17] chose the *heaviest* chain as a valid one in case of forks, where a chain's *weight* is defined as the sum of its blocks' difficulties.

[3] See [ato16] for a visual interpretation of the block structure in Ethereum.

Good connectivity is crucial for Bitcoin mining operation: the resources spent mining on a block other than the latest one are essentially wasted. Good connectivity puts big pools at an advantage, while blocks from worse connected miners propagate slowly and increase the orphan rate. Thus Bitcoin mining is prone to centralization. To be able to operate with block times much shorter than Bitcoin's 10 min (about 30 s in September 2017), Ethereum uses a mining protocol [doc17] similar to GHOST [SZ13]. Ethereum considers *uncles* – valid orphan blocks that are ancestors of the current block (no more than 6 generations deep). For each block, the miner receives a static reward of 5 ether, payments for the gas consumed by transactions in the block, and 1/32 of the static reward (0.15625 ether) per uncle, whose hash is included in the block header (no more than 2 uncles per block). Miners of uncles whose headers get included in the main chain receive 7/8 of the static reward (4.375 ether). Due to uncles, the energy spent on orphan blocks contributes to security, increasing the amount of work required for a double-spend.

Contrary to Bitcoin, where coins are issued on a diminishing rate with a total cap of 21 million, Ethereum issues ethers at a constant rate with no total cap. Ethereum's issuance parameters may change after switching to proof-of-stake (see Sect. 3.1).

Bitcoin PoW uses a general purpose cryptographic hash function SHA-256, which can be efficiently implemented in hardware. Specialized mining equipment (application-specific integrated circuits, ASIC) is orders of magnitude more efficient than commodity hardware, which puts small miners at a disadvantage. Ethereum uses a memory hard hash function Ethash and targets GPUs as the primary mining equipment. It helps prevent mining centralization akin to Bitcoin's and throttles CPU mining (botnets or cloud VM instances can be rented for a short time to perform an attack).

Table 1 compares some properties of Bitcoin and Ethereum. Note that the practical requirements regarding the disk space for an Ethereum node can be greatly reduced due to the explicit storage of account balances and data as opposed to Bitcoin's UTXO [Dom17].

Table 1. Bitcoin and Ethereum, September 2017 [Eth17d, Bit17c, Eth17e, Bit17b, Coi17a]

Metric	Bitcoin	Ethereum
Number of nodes	9428	22007
Blockchain size	158 GB	52 GB
Transactions per hour	8509	12406
Market capitalization ($ million)	62812	27200
Daily trading volume ($ million)	997	420

2.4 Smart Contract Programming

EVM bytecode is a low-level Turing complete stack-based language operating on 256-bit words designed to be simple compared to general purpose VMs

like JVM, execute deterministically, and natively support cryptographic primitives [But17b]. Developers usually write contracts in high-level languages targeting EVM, the most popular one being Solidity [Sol17] – a statically typed language with a Javascript-like syntax. Others include Serpent [Ser17] (deprecated in 2017 [Cas17]) and LLL [Ell17] (Python- and Lisp-like syntax respectively).

```
pragma solidity 0.4.17;
contract StringStorageContract {
    string private str = "Hello, world!";
    function getString() public constant returns (string) {
        return str;
    }
    function setString(string _str) public {
        str = _str;
    }
}
```

Listing 1.1. A simple contract in Solidity

2.5 Applications

Among many potential applications of smart contracts [McA17], crowdfunding is arguably the first widely successful one. The first wide-scale Ethereum-based crowdfunding project was a decentralized investment fund called The DAO, launched on 30 April 2016[4]. In 2017, the amount of money collected during so-called initial coin offerings (ICO) skyrocketed, reaching $1.8 bn [Coi17b] and surpassing early stage venture capital funding [Sun17]. ICO is usually based around a token – a smart contract that maintains a list of users' balances and allows them to transfer tokens or buy and sell them for ether. Tokens are usually implemented with respect to the API defined in the ERC20 standard [Vog17]. The ICO organizers often promise that the tokens will be required to use the to-be developed product or service. Prominent Ethereum applications include decentralized file storage [Fil17,Sia17,Sto17] and computation [Gol17,Son17], name systems [ENS17], and prediction markets [Aug17,Gno17].

3 Open Problems

3.1 Core Protocol

Cryptographic Primitives. Ethereum uses ECDSA for signatures[5], Keccack256 for generating unique identifiers[6], and Ethash [Eth17a] for proof-of-work.

[4] In June 2016, an unknown hacker exploited a vulnerability in the DAO code and withdrew around $50 million, leading to a controversial [ETC16] hard fork.

[5] See [May16] for a study of ECDSA security in Bitcoin and Ethereum.

[6] Though Keccak256 is the winning proposal in the SHA3 competition, it differs from the officially standardized SHA3. SHA3 in the Ethereum documentation and source code refers to Keccak256.

Based on Dagger [But13] and Hashimoto [Dry14], Ethash is a memory intensive, GPU-friendly and ASIC-resistant hash function[7].

The algorithm is composed of four steps. In the first step, a seed is created from the blockchain by hashing the headers of each block together with the current epoch using Keccak. An epoch consists of 30 thousand blocks. In the second step, a 16 MB pseudorandom cache is generated from the seed using a memory-hard hash function. In the third step, done once per epoch, a linearly growing dataset (approximately 2 GB in 2017 [DAG17]) consisting of 64 byte elements is generated from the cache using a non-cryptographic hash function Fowler-Noll-Vo [Nol17]. In the fourth step, the dataset, a header, and a nonce are repeatedly hashed until the result satisfies the difficulty target.

Both Dagger and Hashimoto, in contrast to standardization attempts like the SHA-3 competition [SHA17] or the Password hashing competition [PHC15], were announced shortly before the Ethereum launch and did not undergo significant cryptanalysis in the academic community. The Ethash design rationale [Eth17b] lacks details on why established and well-tested memory-hard hash functions do not serve the purpose. [Ler14] claims that an earlier version of Dagger (as of 2014) was flawed. Rigorous cryptanalysis of Ethereum's underlying cryptographic primitives is required to guarantee its long-term security.

Consensus Mechanism. Though some argue that PoW is the only viable blockchain consensus mechanism [And14, Szt15], Ethereum is planning to switch from proof-of-work to proof-of-stake (PoS) [Her17]. As of September 2017, the first step of a two-stage process is due October 2017, transitioning Ethereum to a hybrid PoW-PoS consensus mechanism. The second step will make Ethereum fully PoS. PoS aims to address the drawbacks of PoW:

- energy consumption comparable to a mid-sized country as of 2017 [Dig17];
- centralization risks: miners are incentivized to invest in specialized hardware, which pushes up the entry cost of participating and puts big miners at an advantage due to economies of scale;
- game-theoretic attacks like selfish mining [ES13].

PoS can be described as "virtual mining": a miner purchases coins instead of hardware and electricity. The consensus mechanism distributes power proportionally to the amount of coins miners hold (*stake*), not computing power (see [BGM16] for a review of cryptocurrencies without PoW). Known issues with naive PoS implementations include:

- *Nothing-at-stake.* As producing new blocks incurs only a negligible cost, a rational PoS validator extends all known chains to get a reward regardless of which one wins. This opens the door to attacks that require far less than 51%

[7] Ethash is also referred to as Dagger-Hashimoto. Official documentation [Eth17a] states that Ethash "is the latest version of Dagger-Hashimoto, although it can no longer appropriately be called that since many of the original features of both algorithms have been drastically changed".

of the stake[8]: the attacker's chain wins if the attacker supports it exclusively, whereas other validators behave rationally and support all chains.
- *Randomly choosing validators.* Using randomness from the blockchain itself (i.e., previous block hash) to determine the next validator is insecure, as it is determined by validators in previous rounds. A possible solution is to use verifiable secret sharing for randomness generation.
- *Transaction finality.* In PoW, a block header which has a hash less than the target simultaneously represents the choice of the next validator and the very act of validating the block. PoS separates choosing the next validators and producing the block. A PoS validator may create its own chain, plug in a constant instead of a pseudo-random number generator (PRNG) output, and produce blocks despite owning an arbitrarily small stake.
  A rule of thumb in Bitcoin considers transactions older than six blocks final, as the chance of a minority attacker overtaking the main chain becomes negligible. By contrast, as PoS blocks cost nearly nothing to produce, an attacker can secretly create an alternative chain starting from the genesis block. To prevent this, a PoS blockchain must provide finality – i.e., guarantee that after a fixed number of blocks old transaction can not be reversed[9].

The central concept of the proposed Ethereum PoS algorithm Casper [But16a] is "consensus by bet": validators bet on the future blockchain state [PoS16, But17c]. Casper addresses the nothing-at-stake problem by introducing validator punishments for incorrect behavior, e.g., extending multiple chains, in addition to rewards, which makes the game-theoretic analysis of the protocol more complex. Long range attacks are addressed with the concept of *finality* [But17a].

Recent PoS designs also include 2-hop blockchain [DFZ16], Algorand [Mic16], Ouroboros [KRDO16], SnowWhite [DPS16], Proof of luck [MHWK17]. Blockchain networks Ripple [SYB14] and Stellar [Maz14] use consensus mechanisms inspired by Byzantine fault tolerant consensus protocols like PBFT [CL02]. Developing an efficient, secure and incentive compatible PoS algorithm is an important task in blockchain research.

Scalability. Open blockchains deliberately sacrifice performance for what a smart contracts pioneer Nick Szabo describes as *social scalability* [Sza17] – "the ability of an institution [...] to overcome shortcomings in human minds [...] that limit who or how many can successfully participate". Both Bitcoin and Ethereum have been facing scalability problems [Sil16, Bit17a]. Improving blockchain scalability while minimally sacrificing security is an important research direction. Blockchain scalability can be defined as two goals: increasing

[8] A commonly used term "51% attack" is not precisely correct: "51%" here means "strictly greater than 50%".

[9] Interestingly, the reference Bitcoin implementation uses checkpoints to skip validation of very old blocks for efficiency, effectively providing finality for transactions older than the latest checkpoint [Bit16].

transaction throughput and decreasing the requirements on bandwidth, storage, and processing power for nodes (thus preserving decentralization).

The first goal can be addressed by payment channel networks and sharding. A bidirectional payment channel is a protocol that lets users exchange signed transactions before publishing of them on-chain as settlement. A network of payment channels is a protocol that finds a sequence of payment channels across the network, a mechanism similar to the IP packet routing [McC15]. Payment channel networks for Bitcoin [Lig16] and Ethereum [Rai17] are in development.

In open blockchains, every node is usually required to process every transaction. This provides strong security, but severely limits scalability. Sharding [GvRS16,LNZ+16] might alleviate this problem by spreading transactions across groups of nodes (*shards*), which should be large enough to provide a sufficient level of security and a significantly better throughput [Sha16].

The second goal can be addressed by skipping the validation of old blocks [Jun17] or by additionally providing new nodes with full snapshots of a previous state [Par17].

Privacy. Most open blockchains[10], including Ethereum, guarantee integrity and availability, but provide little to no privacy. All transactions are broadcast in plaintext and can be intercepted (or later obtained from the blockchain) and analyzed. Deanonymization of blockchain transactions is an active business area with start-ups (e.g., [Cha17b]) offering blockchain analysis tools, which is in line with government demands of KYC/AML compliance for financial services.

A common but only partially efficient privacy preserving practice in Bitcoin, which takes advantage of the UTXO structure of its state, is to use a new address for every transaction. This technique is not applicable in Ethereum, because it uses addresses for authentication and explicitly maps them to accounts states. For instance, if a user purchases tokens using a particular address, they have to use the same address to redeem them.

An additional privacy challenge comes from the requirement to hide business logic behind smart contract code. Though Ethereum only stores bytecode, users are reluctant to trust contracts without published source code. Moreover, bytecode analysis[11] tools are already available [NPS+17,Sui17]. Possible research directions in the privacy domain include privacy preserving smart contracts with zero-knowledge proofs [KMS+15] (support for zero-knowledge proofs in Ethereum was first tested in September 2017 [O'L17]), mixing, computations on encrypted data, and code obfuscation.

[10] Except those using dedicated privacy-preserving cryptographic techniques, e.g., Dash, Monero, Zcash.

[11] Decompiling bytecode to source code is hardly possible as the information about function and variable names is lost during compilation; nevertheless it is possible to display bytecode as a sequence of mnemonics or convert it into an intermediate higher-level representation suitable for analysis.

3.2 Smart Contract Programming

Programming Languages. Security is of paramount importance in smart contract programming [ABC17,DAK+15]. Contrary to traditional software, smart contracts can not be patched, which brings new challenges to blockchain programming [PPMT17]. Multiple approaches exist to contract programming [STM16]. Areas of research in this domain include systematizing good and bad programming practices [Con16,CLLZ17], designing general-purpose [Hir17a, But17d,PE16] as well as domain-specific [BKT17,EMEHR17] smart contract programming languages, and developing tools for automated security analysis [LCO+16,Sec17] and formal verification [BDLF+16] of smart contract source code, EVM bytecode, and the EVM itself [Hir17b].

Secure Contract Programming. An important challenge is to describe smart contracts' execution model (possibly drawing parallels from concurrent programming on a multi-threaded processor [SH17]) and to develop a usable and formally verifiable high-level language reflecting this model. Some argue that Solidity inclines programmers towards unsafe development practices [ydt16]. Typical vulnerabilities and issues in Solidity might include:

1. **Re-entrancy.** Contracts can call each other. Malicious external contracts can call the caller back. If the victim contract does its internal bookkeeping after returning from an external call, its integrity can be compromised[12].
2. **Miner's influence.** Miners can to some extend influence execution (front-running, censorship, or altering environmental variables, e.g., timestamp).
3. **Out-of-gas exceptions.** Computation in Ethereum is many orders of magnitude more expensive than with centrally managed cloud computing services. Developers who do not take it into account may implement functions that require too much gas to fit in the block gas limit and thus always fail.

Trusted Data Sources. Many smart contract applications (financial derivatives, insurance, prediction markets) depend on real-world data. Ethereum is isolated from the broader Internet to guarantee consistent execution across nodes. A popular approach to providing data to contracts in a trust-minimizing way is an oracle – a specialized data provider, possibly with a dedicated cryptographic protocol to guarantee integrity [Ora17]. A recent development is TownCrier – an oracle built with trusted hardware [ZCC+16].

3.3 Higher Level Issues

Governance. In June 2016, a massive Ethereum-based crowdfunding project – The DAO – ended in a disaster: an unknown hacker exploited a bug in the smart contract and obtained around $50 million out of $150 million collected [Sir16].

[12] This bug led to the DAO hack of 2016.

Despite the fact that the Ethereum protocol correctly executed the smart contract code, the Ethereum developers implemented a hard fork that allowed stakeholders to withdraw their deposits. This event raised concerns about Ethereum's governance, as the fork violated the premise of decentralized applications running "exactly as programmed" and lead to the creation of Ethereum Classic [Eth17c]. Governance mechanisms should provide certainty over how updates (potentially breaking compatibility) are introduced.

Though the gas price in ether is determined by the market, the relative gas costs of EVM bytecodes are constant. In September 2016, an attacker exploited a weakness in gas pricing and organized a DoS attack on the network, taking advantage of the fact that certain operations were under-priced [But16b]. The problem was ultimately fixed with a hard fork. Research is needed to propose more flexible mechanisms for determining relative prices of EVM operations.

Incentives. Open blockchains rely on the participants' rationality [CXS+17] and must maintain incentive compatibility, so that rational behavior leads to the overall benefit for the network [LTKS15]. This introduces a new field of study dubbed *cryptoeconomics* – the study of incentives in cryptographic systems. The trustless nature of smart contracts might be used for benign (managing mining pools [LVTS17]) as well as for malicious (providing automatic rewards for attacking mining pools [VTL17]) purposes. Rigorous research should guarantee the proper functioning of the blockchain networks and applications based on a definition of rational behavior.

Usability. Considering the influx of new people into the blockchain space, usable yet secure lightweight blockchain software is needed. From the human-computer interaction (HCI) perspective, a challenging task would be to help users grasp the smart contracts fundamentals without going into technicalities. Research shows that cryptographically sound systems may fail to gain traction due to usability issues [RAZS15]. HCI research is needed to make blockchains and smart contracts usable by general public.

Ethical and Legal Issues. Information security researchers usually adhere to the "responsible disclosure" policy: they report a bug privately to the vendor and give developers time to fix it before publishing the information in the open. Though some oppose this practice [Sch07], it is assumed to decrease the probability of an attack on the live system (unless the attackers discover the same bug independently before a patch is applied). Ethereum introduces a new dimension to the responsible disclosure debate, as smart contracts can not be patched. It is unclear whether it is ethical to fully disclose a vulnerability discovered in a smart contract, if developers can not fix it anyway[13].

[13] A technical response to this issue could be updateable contracts: users communicate with a proxy contract, which redirects their transactions to the latest version of the main contract. Such scheme assumes that the developers are honest and competent so that the latest update does not run away with everyone's money.

A whole separate range of topics, which is outside the scope of this paper, is how (and if at all) smart contracts fit into existing legal frameworks. For instance, BitLicense [ofs15] – a controversial [Act15] piece of regulation that came into force in New York in 2015 – prompted many cryptocurrency businesses to withdraw their services from the residents of this US state [Rob15]. In July 2017, the US Securities and Exchange Commission stated that issuers of digital assets may be subject to requirements of the US law [SC17].

4 Conclusion

Ethereum is a fascinating research area at the intersection of multiple fields: cryptography and distributed systems, programming languages and formal verification, economics and game theory, human-computer interaction, finance and law. The promise of smart contracts is not limited to making existing processes more efficient by putting parts of their logic onto a very inefficient, yet very secure decentralized network. This new way of handling value without a trusted third party opens up whole new classes of previously impossible use cases. Thorough research is needed to realize this vision.

References

[ABC17] Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8

[Act15] EFF Action. Stop the BitLicense (2015). https://act.eff.org/action/stop-the-bitlicense

[And14] Andreev, O.: Proof that proof-of-work is the only solution to the Byzantine generals' problem (2014). http://nakamotoinstitute.org/mempool/proof-that-proof-of-work-is-the-only-solution-to-the-byzantine-generals-problem/

[ato16] atomh33ls. Ethereum block architecture (2016). https://ethereum.stackexchange.com/a/6413/5113

[Aug17] Augur (2017). https://augur.net/

[BDLF+16] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., Zanella-Béguelin, S.: Formal verification of smart contracts: short paper. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS 2016, pp. 91–96. ACM, New York (2016)

[BGM16] Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 142–157. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_10

[Bit16] bitcoin/src/chainparams.cpp (2016). https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp#L146

[Bit17a] Block size limit controversy (2017). https://en.bitcoin.it/wiki/Block_size_limit_controversy
[Bit17b] Cryptocurrency statistics (2017). https://bitinfocharts.com/
[Bit17c] Bitnodes.21.co. Global Bitcoin nodes distrubution (2017). https://bitnodes.21.co/
[BKT17] Biryukov, A., Khovratovich, D., Tikhomirov, S.: Findel: secure derivative contracts for Ethereum (2017). https://hdl.handle.net/10993/30975
[BMC+15] Bonneau, J., Miler, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Research perspectives and challenges for Bitcoin and cryptocurrencies. Cryptology ePrint Archive, Report 2015/261 (2015). http://eprint.iacr.org/2015/261
[BP17] Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. CoRR, abs/1703.06322 (2017)
[Buc14] Buchman, E.: Understanding the Ethereum trie (2014). https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/
[But13] Buterin, V.: Dagger: a memory-hard to compute, memory-easy to verify Scrypt alternative (2013). http://www.hashcash.org/papers/dagger.html
[But16a] Buterin, V.: Casper the friendly finality gadget (2016). https://github.com/ethereum/research/blob/master/casper4/papers/casper_paper.md
[But16b] Buterin, V.: Long-term gas cost changes for IO-heavy operations to mitigate transaction spam attacks (2016). https://github.com/ethereum/eips/issues/150
[But17a] Buterin, V.: Casper the friendly finality gadget (2017). http://vitalik.ca/files/casper_note.html
[But17b] Buterin, V.: Design rationale (2017). https://github.com/ethereum/wiki/wiki/Design-Rationale
[But17c] Buterin, V.: Minimal slashing conditions (2017). https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c
[But17d] Buterin, V.: New experimental programming language (2017). https://github.com/ethereum/viper
[Cas17] Castor, A.: One of Ethereum's earliest smart contract languages is headed for retirement (2017). https://www.coindesk.com/one-of-ethereums-earliest-smart-contract-languages-is-headed-for-retirement/
[Cha17a] Chain (2017). https://chain.com/
[Cha17b] Protecting the integrity of digital assets (2017). https://www.chainalysis.com/
[Chu36] Church, A.: A note on the Entscheidungs problem. J. Symb. Logic **1**(1), 40–41 (1936)
[CL02] Castro, M., Liskov, B.: Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)
[CLLZ17] Chen, T., Li, X., Luo, X., Zhang, X.: Under-optimized smart contracts devour your money. In: SANER, pp. 442–446. IEEE Computer Society (2017)
[Coi17a] Cryptocurrency market capitalizations (2017). https://coinmarketcap.com/
[Coi17b] Coindesk. ICO tracker (2017). https://www.coindesk.com/ico-tracker/
[Con16] Ethereum contract security techniques and tips (2016). https://github.com/ConsenSys/smart-contract-best-practices

[Cor17] Corda (2017). https://www.corda.net/
[CXS+17] Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., Shi, W.: Decentralized execution of smart contracts: agent model perspective and its implications (2017). http://fc17.ifca.ai/wtsc/Decentralized%20Execution%20of%20Smart%20Contracts%20-%20Agent%20Model%20Perspective%20and%20Its%20Implications.pdf
[DAG17] Dag file size calculator (2017). https://investoon.com/tools/dag_size
[DAK+15] Delmolino, K. Arnett, M., Kosba, A., Miller, A., Shi, E.: Step by step towards creating a safe smart contract: lessons and insights from a cryptocurrency lab. Cryptology ePrint Archive, Report 2015/460 (2015). http://eprint.iacr.org/2015/460
[DFZ16] Duong, T., Fan, L., Zhou, H.-S.: 2-hop blockchain: combining proof-of-work and proof-of-stake securely. Cryptology ePrint Archive, Report 2016/716 (2016). http://eprint.iacr.org/2016/716
[Dig17] Digiconomist. Bitcoin energy consumption index (2017). http://digiconomist.net/bitcoin-energy-consumption
[doc17] Ethereum documentation. Mining (2017). http://ethdocs.org/en/latest/mining.html
[Dom17] Domchi. What are the ethereum disk space needs? (2017). https://ethereum.stackexchange.com/q/143/5113
[DPS16] Daian, P., Pass, R., Shi, E.: Snow white: provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919 (2016). http://eprint.iacr.org/2016/919
[Dry14] Dryja, T.: Hashimoto: I/O bound proof of work (2014). https://pdfs.semanticscholar.org/3b23/7cc60c1b9650e260318d33bec471b8202d5e.pdf
[Ell17] Ellison, D.: An introduction to LLL for Ethereum smart contract development (2017). https://media.consensys.net/an-introduction-to-lll-for-ethereum-smart-contract-development-e26e38ea6c23
[EMEHR17] Egelund-Müller, B., Elsman, M., Henglein, F., Ross, O.: Automated execution of financial contracts on blockchains (2017). https://ssrn.com/abstract=2898670
[ENS17] ENS (2017). https://ens.domains/
[ES13] Eyal, I., Gün Sirer, E.: Majority is not enough: Bitcoin mining is vulnerable. CoRR, abs/1311.0243 (2013)
[ETC16] The Ethereum Classic declaration of independence (2016). https://ethereumclassic.github.io/assets/ETC_Declaration_of_Independence.pdf
[eth16] eth. How is the address of an ethereum contract computed? (2016). https://ethereum.stackexchange.com/q/760/5113
[Eth17a] Ethash (2017). https://github.com/ethereum/wiki/wiki/Ethash
[Eth17b] Ethash design rationale (2017). https://github.com/ethereum/wiki/wiki/Ethash-Design-Rationale
[Eth17c] Ethereum Classic (2017). https://ethereumclassic.github.io/
[Eth17d] Etherchain.org. Mining statistics (last 24h) (2017). https://etherchain.org/statistics/miners
[Eth17e] Ethernodes.org (2017). https://www.ethernodes.org/network/1
[Fil17] Filecoin (2017). https://filecoin.io/
[Gno17] Gnosis (2017). https://gnosis.pm/
[Gol17] Golem (2017). https://golem.network/
[GvRS16] Gencer, A.E., van Renesse, R., Sirer, E.G.: Service-oriented sharding with Aspen. arXiv preprint arXiv:1611.06816 (2016)

[Her17] Hertig, A.: Ethereum's big switch: the new roadmap to proof-of-stake (2017). https://www.coindesk.com/ethereums-big-switch-the-new-roadmap-to-proof-of-stake/

[Hir17a] Hirai, Y.: Bamboo: a morphing smart contract language (2017). https://github.com/pirapira/bamboo

[Hir17b] Hirai, Y.: Formal verification of Ethereum contracts (2017). https://github.com/pirapira/ethereum-formal-verification-overview

[Hyp17] Hyperledger (2017). https://www.hyperledger.org/

[jnn15] jnnk. What is gas limit in Ethereum? (2015). https://bitcoin.stackexchange.com/a/39197

[Joh17] Johnson, N.: What is the exact "longest chain" rule implemented in the ethereum "homestead" protocol? (2017). https://ethereum.stackexchange.com/a/13750/5113

[Jun17] Junge, H.: What is Geth's "light" sync, and why is it so fast? (2017). https://ethereum.stackexchange.com/a/11300

[KMS+15] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. Cryptology ePrint Archive, Report 2015/675 (2015). http://eprint.iacr.org/2015/675

[KRDO16] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889 (2016). http://eprint.iacr.org/2016/889

[LCO+16] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. Cryptology ePrint Archive, Report 2016/633 (2016). http://eprint.iacr.org/2016/633

[Ler14] Lerner, S.D.: Ethereum "Dagger" PoW function is flawed (2014). https://bitslog.wordpress.com/2014/01/17/ethereum-dagger-pow-is-flawed/

[Lig16] Lightning network (2016). https://lightning.network/

[LNZ+16] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 17–30. ACM (2016)

[LTKS15] Luu, L., Teutsch, J., Kulkarni, R., Saxena, P.: Demystifying incentives in the consensus computer. Cryptology ePrint Archive, Report 2015/702 (2015). http://eprint.iacr.org/2015/702

[LVTS17] Luu, L., Velner, Y., Teutsch, J., Saxena, P.: SmartPool: practical decentralized pooled mining. Cryptology ePrint Archive, Report 2017/019 (2017). http://eprint.iacr.org/2017/019

[May16] Mayer, H.: ECDSA security in Bitcoin and Ethereum: a research survey (2016)

[Maz14] Mazières, D.: The Stellar consensus protocol: a federated model for internet-level consensus (2014). https://www.stellar.org/papers/stellar-consensus-protocol.pdf

[McA17] McAdams, D.: An ontology for smart contracts (2017). https://iohk.io/research/papers/#QCNR6SCZ

[McC15] McCone, R.: Ethereum Lightning network and beyond (2015). http://www.arcturnus.com/ethereum-lightning-network-and-beyond/

[MHWK17] Milutinovic, M., He, W., Wu, H., Kanwal, M.: Proof of luck: an efficient blockchain consensus protocol. Cryptology ePrint Archive, Report 2017/249 (2017). http://eprint.iacr.org/2017/249

[Mic16] Micali, S.: ALGORAND: the efficient and democratic ledger. CoRR, abs/1607.01341 (2016)
[Nak08] Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
[Nol17] Landon Curt Noll. FNV hash (2017). http://www.isthe.com/chongo/tech/comp/fnv/index.html
[NPS+17] Norvill, R., Pontiveros, B.B.F., State, R., Awan, I., Cullen, A.: Automated labeling of unknown contracts in Ethereum (2017). https://bradscholars.brad.ac.uk/handle/10454/12220
[ofs15] Department of financial services. Bitlicense regulatory framework (2015). http://www.dfs.ny.gov/legal/regulations/bitlicense_reg_framework.htm
[O'L17] O'Leary, R.R.: Ethereum's Byzantium testnet just verified a private transaction (2017). https://www.coindesk.com/ethereums-byzantium-testnet-just-verified-private-transaction/
[Ora17] Oraclize (2017). http://www.oraclize.it/
[Par17] Warp sync snapshot format (2017). https://github.com/paritytech/parity/wiki/Warp-Sync-Snapshot-Format
[PE16] Pettersson, J., Edström, R.: Safer smart contracts through type-driven development (2016). https://publications.lib.chalmers.se/records/fulltext/234939/234939.pdf
[PHC15] Password hashing competition (2015). https://password-hashing.net/
[PoS16] Proof of stake FAQ (2016). https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
[PPMT17] Porru, S., Pinna, A., Marchesi, M., Tonelli, R.: Blockchain-oriented software engineering: challenges and new directions. CoRR, abs/1702.05146 (2017)
[Qtu17] Qtum (2017). https://qtum.org/en/
[Rai17] Raiden network: high speed asset transfers for Ethereum (2017). http://raiden.network/
[RAZS15] Ruoti, S., Andersen, J., Zappala, D., Seamons, K.E.: Why Johnny still, still can't encrypt: evaluating the usability of a modern PGP client. CoRR, abs/1510.08555 (2015)
[Rob15] Roberts, D.: Behind the "exodus" of Bitcoin startups from New York (2015). http://fortune.com/2015/08/14/bitcoin-startups-leave-new-york-bitlicense/
[Roo17] Rootstock (2017). http://www.rsk.co/
[SC17] U.S. Securities and Exchange Commission. SEC issues investigative report concluding DAO tokens, a digital asset, were securities (2017). https://www.sec.gov/news/press-release/2017-131
[Sch07] Schneier, B.: Debating full disclosure (2007). https://www.schneier.com/blog/archives/2007/01/debating_full_d.html
[Sec17] Securify. Formal verification of Ethereum smart contracts (2017). http://securify.ch/
[Ser17] Serpent (2017). https://github.com/ethereum/wiki/wiki/Serpent
[SH17] Sergey, I., Hobor, A.: A concurrent perspective on smart contracts. CoRR, abs/1702.05511 (2017)
[Sha16] Sharding FAQ (2016). https://github.com/ethereum/wiki/wiki/Sharding-FAQ
[SHA17] SHA-3 competition (2007–2012) (2017). http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[Sia17] Sia (2017). https://sia.tech/
[Sil16] Siludin. Let's talk about how poor this network is at handling any type of major transaction traffic (2016). https://redd.it/6ifl5f
[Sir16] Gün Sirer, E.: Thoughts on The DAO hack (2016). http://hackingdistributed.com/2016/06/17/thoughts-on-the-dao-hack/
[Sol17] Solidity official documentation (2017). https://solidity.readthedocs.io/
[Son17] Sonm (2017). https://sonm.io/
[STM16] Seijas, P.L., Thompson, S., McAdams, D.: Scripting smart contracts for distributed ledger technology. Cryptology ePrint Archive, Report 2016/1156 (2016). http://eprint.iacr.org/2016/1156
[Sto17] Storj (2017). https://storj.io/
[Sui17] Suiche, M.: Porosity. Decompiling Ethereum smart-contracts (2017). https://blog.comae.io/porosity-18790ee42827
[Sun17] Sunnarborg, A.: ICO investments pass VC funding in blockchain market first (2017). https://www.coindesk.com/ico-investments-pass-vc-funding-in-blockchain-market-first/
[SYB14] Schwartz, D., Youngs, N., Britto, A.: The Ripple protocol consensus algorithm. Ripple Labs Inc White Paper (2014). https://ripple.com/files/ripple_consensus_whitepaper.pdf
[SZ13] Sompolinsky, Y., Zohar, A.: Accelerating Bitcoin's transaction processing. Fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881 (2013). http://eprint.iacr.org/2013/881
[Sza17] Szabo, N.: Money, blockchains, and social scalability (2017). https://unenumerated.blogspot.lu/2017/02/money-blockchains-and-social-scalability.html
[Szt15] Sztorc, P.: Nothing is cheaper than proof of work (2015). http://www.truthcoin.info/blog/pow-cheapest/
[TS15] Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: a technical survey on decentralized digital currencies. Cryptology ePrint Archive, Report 2015/464 (2015). http://eprint.iacr.org/2015/464
[VB+14] Vogelsteller, F., Buterin, V., et al.: Ethereum whitepaper (2014). https://github.com/ethereum/wiki/wiki/White-Paper
[VBR+17] Vogelsteller, F., Buterin, V., Reitwiessner, C., Kotewicz, M., et al.: Merkle Patricia trie specification (2017). https://github.com/ethereum/wiki/wiki/Patricia-Tree
[Vog17] Vogelsteller, F.: ERC: token standard (2017). https://github.com/ethereum/eips/issues/20
[VTL17] Velner, Y., Teutsch, J., Luu, L.: Smart contracts make Bitcoin mining pools vulnerable. Cryptology ePrint Archive, Report 2017/230 (2017). http://eprint.iacr.org/2017/230
[Woo14] Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 151 (2014). http://yellowpaper.io/
[Wui17] Wuille, P.: What does the term "longest chain" mean? (2017). https://bitcoin.stackexchange.com/a/5542/31712
[ydt16] ydtm. The bug which the DAO hacker exploited was not merely in the DAO itself (2016). https://redd.it/4opjov
[ZCC+16] Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town Crier: an authenticated data feed for smart contracts. Cryptology ePrint Archive, Report 2016/168 (2016). http://eprint.iacr.org/2016/168