DOI:10.1145/3274276

Although smart contracts are Turing complete, it is a misconception that they can fulfill all routine contracts.

BY YONGGE WANG AND QUTAIBAH M. MALLUHI

The Limit of Blockchains: Infeasibility of a Smart Obama-Trump Contract

BLOCKCHAINS HAVE BECOME a buzzword, and many blockchain proponents believe a smart contract is a panacea for redefining the digital economy. But the community has a misconception that any kind of contract could be implemented as a blockchain smart contract. There is no doubt that Turing-complete scripting languages in blockchain techniques, such as Ethereum, can be used to draft many important smart contracts, but the digital economy is much more than Turing-complete smart contracts. Many protocols/contracts in our daily lives could not be implemented using Turing-complete smart contracts. As an example, we have formulated an Obama-Trump contract and show such a contract cannot be implemented using blockchain smart-contract techniques.

As the Internet increasingly becomes part of our daily lives, it will be convenient to have a digital payment system or design digital currency for society. It is generally easy to design an electronic cash system using public key infrastructure (PKI) systems. But PKI-based electronic cash is also easy to trace. Theoretically, banknotes could be traced using sequence numbers, though there is no convenient infrastructure to trace banknote sequence numbers back to users. Banknotes thus maintain sufficient anonymity.

An electronic cash system must avoid double spending and is preferred to be nontraceable and convenient for carrying out small transactions of, say, even a few cents. Such electronic cash systems could be designed using Chaum's blind signatures for untraceable payments.[4] Assume the bank has an RSA public key (e, N) and a private key d. In order for Alice to withdraw $10 from her bank account and convert it to a digital coin m of $10, the system carries out the following protocol:

- Alice chooses a random number r and computes $m' = m \cdot r^e (\bmod N)$.
- The bank generates a signature $s' = (m')^d$ on m'.
- Alice calculates a signature s on m as $s = s' \cdot r^{-1} = (m \cdot r^e)^d \cdot r^{-1} = m^d$.
- Alice spends (m, s) as $10, although the bank cannot link this coin m to Alice's account.

» key insights

- **Smart contracts are self-executing Turing-complete programs stored permanently on the blockchain, triggered by blockchain transactions and able to read/write data from/to the blockchain database.**
- **Expectations for what smart contracts can do are inflated; Turing completeness does not imply they are a comprehensive tool for implementing contracts, as some contracts in our daily lives cannot be realized through smart contracts.**
- **The limit of smart contracts is theoretically proven by using impossibility results in secure multiparty computation to show the infeasibility of implementing an Obama-Trump contract as a smart contract.**

IMAGE BY BY ANDRIJ BORYS ASSOCIATES, USING DOD PHOTO OF TRUMP & OBAMA BY U.S. AIR FORCE STAFF SGT. MARIANIQUE SANTOS

There are various challenges to such a blindsignature-based electronic cash system. The first is what happens if Alice asks the bank to sign m' = $100 \cdot r^e (\text{mod } N)$ instead of m' = $10 \cdot r^e$ (mod N)? It could be resolved by requiring that all coins have the same value or by using the following probabilistic approach:

▸ Alice generates 100 blind coins: $m_i' = m_i \cdot r_i^e \left(\text{mod}\, N\right)$ for $i = 1, \ldots, 100$.

▸ The bank randomly selects $1 \leq j_1 < \ldots < m_{j99}' \leq 100$.

▸ Alice reveals the values m_{j_i}, r_{j_i} to the bank for $i = 1, \ldots, 99$.

▸ The bank issues a signature on the remaining m' only if the $m_{j_i} = 10$ and $m_{j_i}' = m_{j_i} \cdot r_{j_i}^e \left(\text{mod}\, N\right)$ for $i = 1, \ldots, 99$.

The second challenge for Chaum's blind-signature-based electronic cash system is a seller must contact the bank to make sure the coin m has not been spent yet before accepting the coin m from Alice. This requires that the bank remains online at all times. Chaum et al.[5] constructed an electronic cash that does not need the bank to be online. Let H_1, H_2 be hash functions and k be a fixed even integer. Assume Alice has an account u with a bank, and the bank keeps a counter number v for Alice. In order for Alice to get a digital coin from the bank, the following steps are carried out:

▸ Alice chooses random a_i, c_i, d_i, and r_i for $1 \leq i \leq k$.

▸ Alice sends k blind candidates $B_i = r_i^e \cdot H_1(x_i,\ y_i)$ to the bank where

▸ $x_i = H_2(a_i,\ c_i)$

$y_i = H_2(a_i \oplus (u \| (\upsilon + i)),\ d_i)$

▸ The bank chooses a random subset $R \subset \{1, \ldots k\}$ of size $k/2$ and sends R to Alice.

▸ Alice reveals a_i, c_i, d_i, and r_i for all $i \in R$.

- Bank signs $S = \prod_{i \notin R} B_i^d$, deducts the dollar from Alice's account, and increases ν by k.
- Alice extracts coin $C = S \cdot (\prod_{i \notin R} r_i)^{-1} = \prod_{i \notin R} (H_1(x_i,\ y_i))^d$.

When Alice wants to make a payment to Bob, Alice sends C to Bob. Assume

$$\{i_1,\ \cdots,\ i_{k/2}\} = \{1,\ 2,\ \cdots,\ k\} \setminus R.$$

Bob sends random bits $z_{i_1}, \cdots, z_{i_{k/2}}$ to Alice. For $j = 1, \ldots, k/2$, Alice responds as follows:

1. If $z_{i_j} = 0$, then Alice sends $a_{i_j},\ c_{i_j}$ and y_{i_j} to Bob. In this case, Bob is able to compute $H_1(x_{i_j},\ y_{i_j}) = H_1(H_2(a_{i_j},\ c_{i_j}),\ y_{i_j})$.
2. If $z_{i_j} = 1$, then Alice sends x_{i_j}, $a_{i_j} \oplus (u \| (\upsilon + i_j))$, and d_{i_j} to Bob. In this case, Bob is able to compute

$$H_1(x_{i_j}, y_{i_j}) = H_1(x_{i_j}, H_2(a_{i_j} \oplus (u \| (\upsilon + i_j)), d_{i_j})).$$

After receiving these values, Bob is able to verify that C is a signature on the message $\prod_{j=1}^{k/2} H_1(x_{i_j}, y_{i_j})$.

In this transaction process, Alice's bank does not need to be online. In order for Bob to cash the coin C at Alice's bank, Bob sends the coin C together with Alice's response to Alice's bank. One may wonder: If Alice's bank is not online, how can we avoid double spending? If Alice spends the same coin both at Bob's shop and at Charlie's shop, then the challenge sequences $z_{i_1}, \cdots, z_{i_{k/2}}$ from Bob and Charlie are different with high probability. Assume the challenge bit $z_{i_1} = 0$ for Bob and $z_{i_1} = 1$ for Charlie. Then Alice has revealed a_{i_1}, c_{i_1}, y_{i_1}, x_{i_1}, $a_{i_1} \oplus (u \| (\upsilon + i_1))$, and d_{i_1}. That is, Alice's account number u could be recovered from these revealed values. Or if Alice double spends, her identity will be revealed.

Many other non-PKI-based digital cash systems have been proposed in the literature. For example, Rivest and Shamir[12] proposed the PayWord and MicroMint payment schemes. In PawWord, Alice computes a sequence of binary strings w_0, w_1, w_2, …, w_n such that $w_i = H(w_{i+1})$, where H is a secure cryptographic hash function. Alice then commits w_0 to the bank that cannot be spent. Assume each payment is one cent, then the i-th cent is spent as (i, w_i). In MicroMint, there is a central broker to mint the coins. For example, in order for the broker to mint 2^{30} coins, it will use an array of 2^{30}. The broker will repeatedly hash randomly selected binary strings r and put the pair $(r, H(r))$ in the bin labeled $H(r)$. The mint process is finished when each of these bins contains four entries. Each bin is considered as one coin. That is, each coin is a tuple (x_1, x_2, x_3, x_4) such that $H(x_1) = H(x_2) = H(x_3) = H(x_4)$.

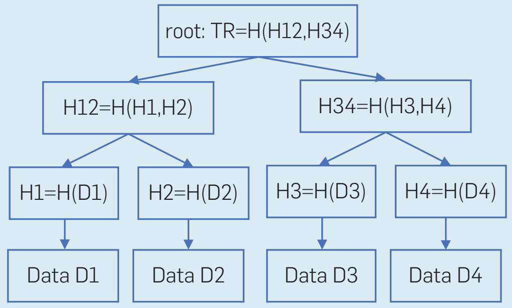
Bitcoin

The cryptographic currencies in the preceding section have never been adopted in practice. The situation has changed as the cryptographic currency Bitcoin was introduced in a paper by pseudonym "Satoshi Nakamoto."[10] Since 2009, Bitcoin has been in operation and widely adopted as one of the major cryptographic currencies in the market. The cryptography behind Bitcoin is quite simple. The start coinbase by Satoshi Nakamoto is a binary string w_0. In order to mine the first Bitcoin BTC, one needs to find a random number r_0 such that the first 32 bits of $w_1 = H(w_0, r_0)$ are 0…0 (that is, $w_1 < 2^{|w_0|-32}$). Anyone who finds this r_0 is rewarded with a few BTCs. The next person who finds another r_1 such that the first two bits of $w_2 = H(w_1, r_1)$ are 0…0 will also be rewarded with a few BTCs. This process continues, and new blocks w_{i+1} keep adding to the existing block chain w_0, …, w_i. If the frequency of finding a BTC block is less than 10 minutes, the community initiates a voting process to increase the number of 0s in the required prefix of the hash outputs. The Bitcoin is a chain w_0, w_1, …, w_n, where w_n is the current Bitcoin head everyone works on. The Bitcoin network is a peer-to-peer (P2P) network with all participants working on the longest chain. There is no benefit for one to work on a shorter chain, as it is a waste of time and the transaction included in these chains will not be valid. The transactions of Bitcoins are included in the hash inputs so they can be verified later. Specifically, we have

$$w_{i+1} = H(w_i,\ \mathrm{TR},\ r_i)$$

where TR is the Merkle hash of the transactions one wants to include and r_i is a random number one finds to make w_{i+1} have a certain number of 0s; the Merkle hash tree is outlined in the figure here.

Merkle hash tree.



In the Bitcoin system, a user is identified by a public key, and a transaction is in the format of "Alice pays x BTC to Bob." Alice achieves a transaction by signing the message "reference number, Bob's pub key, BTC amount x," where the reference number refers to a block w_i in the current BTC chain w_0, w_1, …, w_n where Alice received at least x BTCs in a transaction with the given reference number included in w_i. For example, the block w_i includes a transaction with this given reference number showing Alice received certain amount of BTCs. Bitcoin transactions are described using Forth-like scripts. The scripts enable smart contracts, such as "the transaction will be valid two days after all three persons have signed the contract." The Forth-like scripts are a stack-based script language and was used mainly in calculators. For example, in order to compute 25 × 10 + 50, one needs to initialize the stack as "[top] 25, 10, *, 50, + [bottom]."

Though it is argued that if the majority of users are honest, then the Bitcoin protocol should be reliable,[10] Eyal and Sirer[7] showed this may not be true. In Eyal's and Sirer's attack, the adversary controls 1/3 computing power of the entire Bitcoin community and does not reveal the block it mined if it leads. The other 2/3 of users will waste their time on a chain that will be abandoned at some point when the adversary reveals its own leading chain. As users could choose arbitrary public keys for Bitcoins, it is claimed that user privacy is preserved in Bitcoins to some degree.[10] There have been significant efforts to analyze the privacy issues in Bitcoin systems, and the conclusion is that a significant amount of private information could be recovered from Bitcoin chains. There have been many proposals concerning privacy-preserving solutions in Bitcoin networks. Androulaki et al.[2] tried to give a privacy definition in Bitcoin networks based on the traditional definition of privacy in computer networks. Following these definitions, Androulaki et al.[2] implemented a simulated Bitcoin network and observed a 40% user profile could be identified in the simulated environments. Ober et al.[11] analyzed some global properties of Bitcoin networks and their impact on user privacy. Möser[9] analyzed three mixing services for Bitcoin networks: BTC Fog, BitLaundry, and Shared Wallet from Blockchain.info. Möser[9] observed that among these three services, BTC Fog and Shared Wallet have good privacy protection, and tainted analysis could be used to trace Bitcoins in BitLaundry due to its lower volume per day. Moore and Christin[8] analyzed 40 Bitcoin exchange centers, observing the smaller the volume, the shorter the lifetime of the exchange center. On the other hand, more recent work by Ahmed et al.[1] showed serious attacks against public cryptocurrency-mining pools, such as Minergate and Slush Pool. In them, an attacker needs only a small fraction (such as one millionth) of the resources of a victim-mining pool to render the victim-mining pool nonfunctional.

If the contract language is Turing-complete, then the required validation systems are equivalent to the problem of deciding whether a universal Turing machine halts on a specific input.

Ethereum

Though Forth-like scripts in Bitcoin are sufficient for designing various kinds of smart contracts, it has a limited capability. One underlying philosophy in Ethereum is to include a Turing-complete programming language within the blockchain system so any kind of smart contract can be supported in the blockchain. Ethereum was designed as an Internet Service Platform with the goal that anybody can upload programs to the Ethereum World Computer, and anybody can request an uploaded program be executed. There are mainly two new functions in Ethereum compared with Bitcoin:

- Ethereum is a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats, and state transition functions.
- Bitcoin supports only "proof of work," whereas Ethereum supports both "proof of stake" *and* "proof of work," where "proof of stake" calculates the weight of a node as proportional to its currency holdings, not its computational resources.

The runtime environment for smart contracts in Ethereum is based on the Ethereum virtual machine (EVM). The EVM can run any operations that are created by the user using the Turing-complete Ethereum scripting language called Solidity. An Ethereum account is a 20-byte string with four fields: nonce, ether balance, contract code (optional), and storage (empty by default). There are two kinds of Ethereum accounts: Ethereum externally owned accounts (EOAs) and contract accounts. An EOA is linked to a private key, and a contract account can be "activated" only by an EOA. A contract account is governed by its internal smart-contract code programmed to be controlled by an EOA with a certain address. A smart-contract program within a contract account executes when a transaction is sent to that account. Senders of a transaction must pay for each step of the "program" they activate. This includes both computation and memory storage costs. Users can create new contracts by deploying code to the blockchain.

Infeasibility of a Smart Obama-Trump Contract

As blockchains use Turing-complete script languages to draft smart contracts, many people might have the misconception that any kind of contract can be implemented in blockchains. Though most financial contracts can be implemented using Turing-complete script languages, there are challenges in implementing smart contracts with private inputs. In this section, we analyze the limit of smart contracts that can be implemented in blockchains. In particular, we show it is theoretically impossible to implement the so-called Obama-Trump contract.

In the legal system, there are four types of classifications of contracts with various bases: formation, nature of consideration, execution, and validity.

1. On the basis of formation, there are three types of contracts: express, implied, and quasi contracts. For an express contract, there is an expression or conversation. For an implied contract, there is no expression. For example, sitting in an airplane incurs an implied contract between the passenger and the airline. For a quasi contract, there are no contractual relations between the partners. This kind of contract is created by virtue of law.

2. On the basis of the nature of the consideration, there are two types of contracts: bilateral and unilateral. A bilateral contract requires considerations in both directions to be moved after the contract, whereas a unilateral contract requires considerations to be moved in only one direction after the contract. An example of a bilateral contract is "Alice delivers goods to Bob on January 1st and Bob pays Alice on January 15th."

3. On the basis of execution, there are two types of contracts: executed and executory. In an executed contract, the performance is completed. In an executory contract, the contractual obligations are to be performed in the future.

4. On the basis of validity, there are five types of contracts: valid, void, voidable, illegal, and unenforceable. A contract that is enforceable in a court of law is called a valid contract, and a contract that is not enforceable in a court of law is called a void contract, as in, say, a contract between Alice and Bob where Bob is a minor who has no capacity to contract is a void contract. A voidable contract is deficient only in terms of free consent. For example, the contract between Alice and Bob where Bob has forcibly made Alice involved in the contract is a voidable contract at the option of Alice. An illegal contract contains an unlawful object. An unenforceable contract has not properly fulfilled legal formalities.

With the classification of these contract types, it is important to design validation systems to check the validity of smart contracts. We would like to see the following validation systems:

- Check whether one transaction is an implied contract;
- Check whether one transaction follows a quasi contract; and
- Check whether a contract is valid, void, voidable, illegal, or unenforceable.

If the contract language is Turing-complete, then the required validation systems are equivalent to the problem of deciding whether a universal Turing machine halts on a specific input. It is thus infeasible to design efficient validation systems to carry out these tasks due to the nondecidability of the universal Turing machine halting problem. In the Ethereum EVM, gas is needed to evaluate a contract. If the gas runs out before the contract is validated, the contract will not be honored.

Furthermore, not all such contracts could be enforced in blockchain as smart contracts. In particular, when privacy does not have a reasonable price tag, it is generally difficult to formulate a smart contract with private inputs. As an example, we show that a bilateral contract is difficult to implement if the second consideration in the bilateral contract is not a digital cash (such as not an Ethereum ETH). In April 2011, Donald Trump made the comment[13] in an interview with ABC's George Stephanopoulos: "Maybe I'm going to do the tax returns when Obama does his birth certificate … I'd love to give my tax returns. I may tie my tax returns into Obama's birth certificate." Based on this comment, we formulate the following Obama-Trump contract and show this kind of bilateral contract is impossible to implement as a blockchain smart contract.

Obama-Trump Contract: *Donald Trump releases his tax return forms as soon as Barack Obama releases his birth certificate.*

The infeasibility of implementing the Obama-Trump contract as a blockchain smart contract can be mathematically proved using the infeasibility results in secure multiparty computations. We first review Cleve's result[6] on the limits of coin flips when half of the participants are faulty.

Theorem 4.1 *(Cleve[6]) If at least half of the participants are faulty, then there is no protocol to allow an asynchronous network of participants to agree on random (unbiased) bits.*

Cleve[6] defines a two-processor bit-selection scheme as a sequence of pairs of processors $\{(A^n, B^n)\}_{n=1}^{\infty}$ with the following properties. For each n, A^n and B^n each has access to a private supply of random bits and they can communicate with each other. If the system is executed, then A^n and B^n will output bits a and b, respectively, within a polynomial time. Assume the system consists of $r(n)$ rounds where each round consists of the following events: A^n performs some computations and sends a message to B^n, and then B^n performs some computations and sends a message to A^n. The two-processor bit-selection scheme is said to be "correct" if after the scheme is run, we have $a \neq b$ with a negligible probability. The two-processor bit-selection scheme is said to be "random" if the scheme is correct and if after the scheme is run, the value $|\text{Prob}[a=0]-\frac{1}{2}|$ is negligible. If one of the two processors is faulty, then it is unrealistic to expect the correctness of the scheme as the faulty processor could output a bit that is independent of the scheme that was run. However, it is desirable that the output of the honest processor is still random. Cleve[6] defines a two-processor bit-selection scheme to be *secure* if the following holds: For each n, if one of A^n, B^n is faulty, then $|\text{Prob}[c=0]-\frac{1}{2}|$ is negligible where c is the output of the honest processor. Cleve[6] shows that no secure two-processor bit-selection scheme exists when one of the processors is faulty. A similar construction as in Cleve's proof[6] could be used to show the following theorem, as outlined here.

Theorem 4.2 *Obama-Trump smart contract cannot be enforced on blockchain.*

Theorem 4.2 shows it is infeasible to implement an Obama-Trump smart contract on blockchains. On the other hand, if a trapdoor function exists, then coin-flipping protocols (see Blum[3] and Cleve[6]) can be used to design weakly secure Obama-Trump smart contracts over blockchain.

Smart Contract Scenarios

The results in the preceding section show that not all contracts can be implemented as a blockchain smart contract. However, blockchain smart contracts could do better than other technologies in many practical contract scenarios where the contract execution process takes a significant amount of time. For example, insurance-claim processing involves many manual operations and much human action. Blockchain smart contracts could help reduce these manual steps by including some measurable parameters, such as earthquake magnitude, within the contracts. When an insured event occurs, the event information is converted to smart contract input parameters, and the claim process is triggered immediately.

Smart contracts can also be used in many other scenarios where a lot of paperwork and coordination are required. For example, in trade finance, the process of letter-of-credit issuance requires numerous physical documents. As another example, in the rental-property application process, the applicant needs to submit numerous documents, including income certificates, rental credit reports, eviction history, and other related documents to the landlord. Note the user may need to submit identical documents to both the trade-finance vendor and the landlord at different times if the user is involved in both processes. It is thus preferred for a user to keep all these documents in a central blockchain account and submit only appropriate reference numbers to the documents for each application. The system should be designed in such a way that the user needs only to disclose minimal mandatory information to each vendor for a specific application. For example, for a user to apply for a rental property, the system should disclose only user income, rental credit reports, and eviction reports to the landlord. The system should not disclose user eviction reports to the trade finance organization.

As information stored in the blockchain is publicly accessible, it is necessary to encrypt user documents in the user account. We may assume each document in a user profile has been certified by a related agency that is also a user account in the blockchain. As an example, the user Alice's master profile may look like this:

$$\text{Alice Profile: } \mathrm{DOC}_1, \mathrm{DOC}_2, \ldots$$

where each document DOC_i is in the following format:

$$\mathrm{DOC}_i = \texttt{S.Enc}_{\texttt{K}}(F, \texttt{Sign}_{\texttt{Agency.pk}}(F)),\\ \texttt{P.Enc}_{\texttt{Alice.pk}}(\mathrm{K}), \texttt{Agency.pk}$$

where the document F is certified by the agency with a digital signature $\texttt{Sign}_{\texttt{Agency.pk}}(F)$ using the agency public key $\texttt{Agency.pk}$. The certified document $(F, \texttt{Sign}_{\texttt{Agency.pk}}(F))$ is then encrypted using a symmetric encryption scheme $\texttt{S.Enc}_{\texttt{K}}(\cdot)$ with a key $\texttt{K}$. The symmetric key $\texttt{K}$ is encrypted using a public encryption scheme $\texttt{P.Enc}_{\texttt{Alice.pk}}(\cdot)$ with Alice's public key $\texttt{Alice.pk}$. In order for Alice to disclose the certified document $(F, \texttt{Sign}_{\texttt{Agency.pk}}(F))$ to the landlord, Alice needs to provide the document reference number DOC_i and the symmetric key $\texttt{K}$ to the landlord.

Other Sophisticated Smart Contracts

A blockchain smart contract is generally written using a blockchain scripting language, such as Solidity. The algorithms within the smart contract are thus available for public review. In some applications, such as the insurance industry, the vendor may not want the public to learn the claim-processing algorithms used in the smart contract. Software obfuscation techniques may be used by smart contracts to hide these algorithms. Indeed, using reusable garble circuit techniques or fully homomorphic encryption (FHE) techniques are preferred for writing smart contracts in these scenarios. However, there are challenges in employing garbled circuits or FHE techniques in these scenarios, as it is difficult to convert plaintext inputs into garbled inputs for garbled circuits or into encrypted inputs for FHE schemes.

PKI is the core component of the secure Internet infrastructure. Note, a PKI system based on blockchain smart-contract systems may be established to replace the current certificate authority (CA)-based PKI systems for Internet infrastructure. It depends on the corresponding cost and security characteristics for one to consider whether to use the current CA-based PKI system or blockchain-based PKI system for Internet infrastructure.

Acknowledgment

The work reported in this article is supported by Qatar Foundation Grant NPRP X-063-1-014.

References

1. Ahmed, M., Wei, J., Wang, Y., and Al-Shaer, E. A poisoning attack against cryptocurrency mining pools. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, 140–154.
2. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., and Capkun, S. Evaluating user privacy in Bitcoin. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 2013, 34–51.
3. Blum, M. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News, 15*, 1 (1983), 23–27.
4. Chaum, D. Blind signatures for untraceable payments. In *Proceedings of Crypto*. Springer, 1983, 199–203.
5. Chaum, D., Fiat, A., Naor, M. Untraceable electronic cash. In *Proceedings of Crypto*. Springer-Verlag, New York, 1990, 319–327.
6. Cleve, R. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of 18th ACM Symposium on Theory of Computing*. ACM, 1986, 364–369.
7. Eyal, I., Sirer, E.G. Majority is not enough: Bitcoin mining is vulnerable. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 2014, 436–454.
8. Moore, T. and Christin, N. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 2013, 25–33.
9. Moser, M., Bohme, R., and Breuker, D. An inquiry into money-laundering tools in the Bitcoin ecosystem. In *eCrime Researchers Summit*. IEEE, 2013, 1–14.
10. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, 2008; https://bitcoin.org/bitcoin.pdf
11. Ober, M., Katzenbeisser, S., and Hamacher, K. Structure and anonymity of the Bitcoin transaction graph. *Future Internet 5*, 2 (2013), 237–250.
12. Rivest, R.L. and Shamir, A. PayWord and MicroMint: Two simple micropayment schemes. In *Proceedings of the International Workshop on Security Protocols*. Springer, 1996, 69–87.
13. Trump, D. I will release my tax returns when Obama releases his birth certificate. 2011; http://www.businessinsider.com/donald-trump-tax-returns-obama-birth-certificate-2011-4

Yongge Wang (Yongge.wang@gmail.com) is in the Department of Software and Information Systems, University of North Carolina, Charlotte, NC, USA.

Qutaibah M. Malluhi (qmalluhi@qu.edu.qa) is in the Department of Computer Science and Engineering, Qatar University, Qatar.