Overcoming Limits of Blockchain for IoT Applications

Francesco Buccafurri
University of Reggio Calabria
Via Graziella, Loc. Feo di Vito
Reggio Calabria, Italy 89122
bucca@unirc.it

Gianluca Lax
University of Reggio Calabria
Via Graziella, Loc. Feo di Vito
Reggio Calabria, Italy 89122
lax@unirc.it

Serena Nicolazzo
University of Reggio Calabria
Via Graziella, Loc. Feo di Vito
Reggio Calabria, Italy 89122
s.nicolazzo@unirc.it

Antonino Nocera
University of Reggio Calabria
Via Graziella, Loc. Feo di Vito
Reggio Calabria, Italy 89122
a.nocera@unirc.it

ABSTRACT

Blockchain technology allows the implementation of a public ledger securely recording transactions among peers without the need of trusted third parties. For both researchers and industry, IoT appears a domain in which there would be extraordinary benefits if the features of Blockchain can be exploited. Indeed, the possibility that IoT devices participate in public shared transactions enables a lot of challenging applications. However, there are some aspects that may limit the use of Blockchain in IoT. These are mainly related to the low computational power and storage capabilities of IoT devices. In this paper, we propose an alternative way to implement a public ledger overcoming the above drawbacks, thus appearing more suitable to IoT applications. The proposed protocol leverages the popular social network Twitter and works by building a meshed chain of tweets to ensure transaction security. Importantly, Twitter does not play neither the role of trusted third party nor the role of ledger provider.

KEYWORDS

Blockchain; Twitter; Public Ledger; IoT applications

ACM Reference format:
Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. 2017. Overcoming Limits of Blockchain for IoT Applications. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 6 pages.
DOI: 10.1145/3098954.3098983

1 INTRODUCTION

Record keeping, coordination, sharing, and irrevocability of transactions are features that make the Blockchain technology exploitable not only for cryptocurrencies. As a matter of fact, a lot of applications can be designed on top of Blockchain, every time a way to record and share anything of value arises [24]. From the point of view of security, there is some debate. Indeed, a number of possible attacks are documented [5, 20, 22, 25, 26], even though, for the advocates of Blockchain, they are mostly not relevant in practice. Anyway, security of Blockchain is not the focus of this paper.

Our interest is on the actual suitability of Blockchain for the application domains nowadays considered as good candidates. Among others, the Internet of Things (IoT) is viewed as a challenge domain in which the features offered by the Blockchain protocol can give relevant benefits. Indeed, enabling IoT devices to participate in transactions by making it possible that information coming from devices such as RFID-based locations, barcode-scan events, or device-reported data, can be securely and irrevocably shared by multiple parties appears extremely advantageous. However, it is also easy to realize that some features of Blockchain are little suitable for the IoT domain. The first and major obstacle to the adoption and the growth of Blockchain technologies in the IoT domain is the concentration and consolidation of *mining* power. Indeed, in Blockchain, the consensus on the ledger is reached thanks to the computation of the *proof of work* done by *miners* [25], requiring a huge computational effort. This means that most IoT endpoints are not suitable for mining because of their limited computational power. The second impediment is that any IoT device needs the full suite of cryptographic capabilities. This excludes most of the cheap IoT solutions and increases the minimum hardware cost. The third factor is the storage capacity. The Blockchain protocol is designed in such a way that each node has to maintain a copy of the Blockchain, that is every transaction from the beginning of time. Therefore, any new device, in order to become a node in the Blockchain community, should download all the transactions right from the first block. Unfortunately, a normal IoT device might not provide this storage capacity. Finally, any entity participating in the Blockchain system must join a P2P network and must exchange data in upload and download continuously. This has a significant impact on power consumption and battery life of wireless devices.

In this paper, we propose an alternative to Blockchain aimed to overcome the above drawbacks and thus designed to make public-ledger-based solutions suitable for the IoT domain. Our solution leverages the extraordinary power of online social networks in sharing information among people. As a matter of fact, so far social networks have been mostly used as communication media. Our vision is that they can become part of workflows, with reciprocal advantage for both people (high usability, low cost, high availability)

ARES '17, Reggio Calabria, Italy

DOI: 10.1145/3098954.3098983

and for social network providers (a more central role in the society). Social networks may thus help us to overcome the limits of Blockchain in the IoT domain. To be effective, we focus our attention on a real-life social network, namely Twitter but, in principle, we could use any social network implementing a posting mechanism and a key-based search for posts. The idea is not to implement Blockchain over Twitter, but to reinvent a consensus protocol using tweets to encode transactions and meshed replications to substitute the proof of work and thus the need of fees for miners. The proposed protocol is called `Tweetchain`. The consensus on transactions is based on the presence of a sufficient number of valid *confirmation* tweets from the community, which form a meshed chain making it impossible for anyone, including Twitter itself, to alter, delete or forge valid transactions. All nodes participate in the same way, so that from a conceptual point of view, this protocol is truly decentralized. No P2P feature must be enabled on the client machine, but only an application working on the Twitter profile of the corresponding user. The role of Twitter is not that of a trusted third party, not even a provider of some part of the protocol. It just continues to publish people tweets, as before. Twitter cannot succeed as an adversary: indeed, thanks to the meshed network of confirmation tweets, it cannot selectively omit the collaboration on some transaction, some user, some period, etc. It could only stop the entire service, i.e., the whole Tweetchain community. This situation is very little plausible, similarly to the event that Blockchain is denied by bodies providing the Internet. Anyway, to prevent this (even improbable) possibility, we could extend our protocol by involving multiple social networks also by using existing models and technologies supporting a similar approach [6, 8].

We remark that our idea is not that of substituting Blockchain, which obviously remains preferable in all the cases in which any single part should not have the chance to stop the service, as for the case of cryptocurrencies. Tweetchain can be viewed as a lightweight public ledger for non-critical services, oriented at least to the domain of IoT, but also, for example, to the domain of crowdsourcing (as preliminarily sketched in [7]) or small-value transactions (indeed, at moment, only transactions with a consistent fee have a good expectation to be validated soon, so that small-value transactions could be hardly implemented over Blockchain).

The paper is organized as follows. The comparison with the related literature is provided in Section 2. Then, in Section 3, we describe our approach. We present the security analysis of our approach in Section 4. In Section 5, we evaluate the performance of our protocol. Finally, we draw our conclusions in Section 6.

2 RELATED WORK

Since its creation in 2008 [18], Blockchain technology has sparked a growing interest within the scientific community. Indeed, many works try to solve problems related to Blockchain in order to improve its functionalities. In particular, in [4], the authors propose a new technology, called pegged sidechains, which enables Bitcoins and other ledger assets to be transferred between multiple Blockchains. This allows the interoperation among different cryptocurrency systems and gives users access to these systems reusing Bitcoin currency and maintaining the assets they already own. In [14], an approach to guarantee privacy over smart contract systems based on decentralized cryptocurrencies is presented. In the classical Blockchain protocol, the entire sequence of transactions taken in a smart contract are propagated across the network and saved in the Blockchain, and therefore are publicly visible. This system does not store financial transactions in plain text on the Blockchain, solving also deanonymization attacks [17, 20]. If Blockchain can be seen as a simple application on a decentralized, but singleton, compute resource, Ethereum implements this paradigm in a generalized manner [27]. Indeed, Ethereum is a decentralized platform used to run *smart contracts*, which are scripts run securely.

Starting from the consideration that Bitcoin and Ethereum use blocks to synchronize a global log of events between nodes in their network and that such a mechanism of block propagation is constrained (i.e., if blocks are created at a high rate compared to their propagation time in the network, many conflicting blocks are created and performance suffers greatly), the authors of [16] propose an alternative structure to the chain. This new structure consists of a directed acyclic graph of blocks that can refer multiple predecessors. This allows for higher transaction acceptance rate so that transaction volumes can be increased. Always in this direction, the works presented in [23] describe alternate structures, based on a tree of blocks, in order to reach higher rates. Our work maintains a meshed chain structure, thus forming a sort of grid. Moreover, the block entity is assimilated with a single transaction. Also performing transactions off the chain can be seen as an alternative to improving the bandwidth and latency [10, 19].

Our consensus model can be framed within the family of state-machine replication (SMR) protocols that consists essentially in replicating deterministic state machines in different hosts [21]. Moreover, it guarantees consensus despite participation of malicious (Byzantine) nodes because all the transaction exchanged are published on Twitter [2]. The Blockchain mechanism is based on incentive to incentive transaction verification, deterring the formation of large open mining pools [3, 15]. Within our protocol we do not consider incentives. Indeed, it is made up by peer nodes, assuming in turn the role of verifiers and transaction generators. If a node wants to benefit from `Tweetchain` services, it has to agree with the protocol and serve as verifier when needed.

3 THE TWEETCHAIN MODEL

In this section, we describe our model and the structure of the messages exchanged. We have three actors: The Twitter social network, providing the possibility to post, search for, and notify *tweets* for registered users; A welcome profile W used to implement a sort of yellow page support; The `Tweetchain` community, namely C, of users who join the `Tweetchain` protocol.

To participate in the `Tweetchain` community, a user builds a SHA-256 hash chain [11] of length k by starting from a secret. This chain will be used to keep all his timeline activities linked together and unforgeable. In other words, no change can be done on previous messages without compromising the remaining part of the timeline. The value k is a parameter of the system which also limits the maximum size of the chunk of the user timeline (intended as number of tweets) that must be considered to verify the validity of a given message. We will consider our system in a steady-state, meaning that there are always at least $s = \frac{2t}{1-m}$

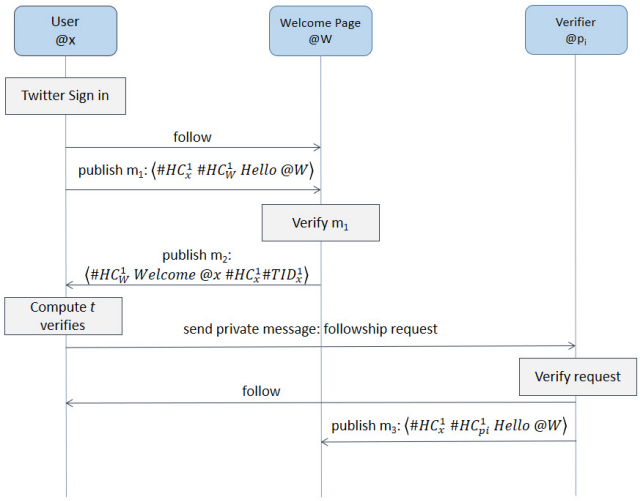

Figure 1: A scheme representing the registration procedure.

members in the Tweetchain community, where t and m are system parameters discussed in Section 4.

Now, we are ready to describe our proposal. The Tweetchain model is composed of the following protocols.

Registration. This step is run by a user x who wants to join the Tweetchain community C and is illustrated in Figure 1. Obviously, a prerequisite is the standard sign up to Twitter in order to create a profile on it.

The first step x performs is to follow a special profile W, called *welcome profile* and to publish a hello tweet with the structure $\langle\ \#HC_x^1\ \#HC_W^1$ Hello $@W\ \rangle$ where $\#HC_x^1$ and $\#HC_W^1$ are Twitter hashtags including the base64 encoding of the first element of the hash chain of x and W as text, respectively. $@W$ is a reference to the welcome page W. As second step of the protocol, W verifies the tweet of x and sends a confirmation tweet as a welcome message with the reference to this user and a link to his hello tweet. Suppose now that W has already posted $i-1$ tweets. Then, the structure of the welcome message for x is $\langle\ \#HC_W^i$ Welcome $@x\ \#HC_x^1\ \#TID_x^1\ \rangle$ where $\#HC_W^i$ is the hashtagged base64 encoding of the i-th element of the hash chain of W and $\#HC_x^1$ is the hashtagged first element of the hash chain of x. $@x$ is a Twitter reference to the user x, and $\#TID_x^1$ is a Twitter hashtag with the ID of the first tweet (hello) of x as text. Note that the ID of a tweet is unique inside Twitter.

Therefore, W contains at least one tweet per member in a join-first chronological order. At this point, x generates at random the set F_x of followings who are selected to validate his transactions in the future. The set F_x is obtained as follow:

- x finds his Twitter identifier. (Each profile in Twitter has a 64-bit identifier.)
- The identifier is used as seed of a public PRNG to generate s random numbers. For each generated number, say n, n *mod* w is computed, where w is the total number of tweets posted by W (corresponding to the size of the Tweetchain community).

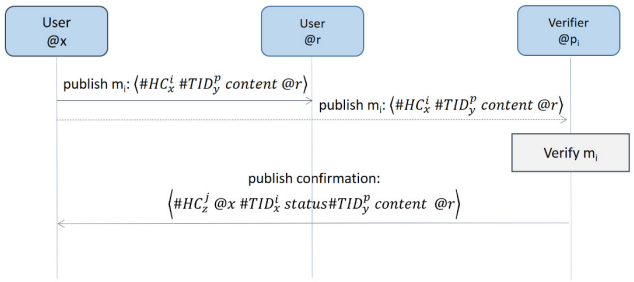

Figure 2: A scheme representing the transaction generation.

- The numbers so obtained are used as indexes to select s distinct profiles from W.
- At this point, x sends a *private* message to each of the s profiles asking them to follow him.
- After verifying the legitimacy of the message coming from x by using the community PRNG, each of the contacted profiles adds a follow link to x and duplicates the welcome tweet of W by replacing $\#HC_W^i$ with its current hash chain element.

Transaction generation. This is the protocol performed to generate a new transaction. Similarly to Blockchain, each transaction includes multiple information:

- The timestamp of the transaction,
- A payload, i.e., the transaction content,
- An input transaction.
- A transaction recipient, which is a target profile.

The generation of a new transaction in our protocol corresponds to the posting of a new *tweet* by the user. We call this tweet *t-tweet*. According to the requirements above, the i-th *t-tweet* of the user x has the structure $\langle\ \#HC_x^i\ \#TID_y^p$ content $@r\ \rangle$ where $\#HC_x^i$ is the hashtagged base64 representation of the i-th element of the hash chain of x, $\#TID_y^p$ is the hashtagged ID of the p-th tweet posted by the user y as input for the transaction, and $@r$ is a Twitter reference to the recipient r.

As soon as x posts a new *t-tweet*, the s follower users of the set F_x will be notified by Twitter automatically. They will verify the legitimacy of the new transaction by using the verification protocol below. After running the verification procedure with success, they will publish a confirmation tweet on their timeline. Now, let v be a user in F_x and let $j-1$ be the number of tweets generated by v until now. Then, the confirmation tweet for the transaction of x is:

$$\langle\ \#HC_v^j\ @x\ \#TD_x^i\ \texttt{status}\ \#TID_y^p\ \texttt{content}\ @r\ \rangle$$

where $\#HC_v^j$ is the hashtagged j-th element of the hash chain of the verifier v, $@x$ is the Twitter reference to the user x, $\#TD_x^i$ is the hashtag of the ID of the *t-tweet* generated by x, and *status* is either 1 or 0, meaning *success* or *failure*, respectively. The remaining of this tweet is the essential part of the body of the *t-tweet* of x necessary to reconstruct the original tweet in case of deletion done by x. Figure 2 shows a scheme representing the generation of a new transaction.

Verification. This protocol is used to verify the validity of a transaction. Let v be a verifier who wants to check the i-th transaction of the user x. Suppose that this transaction has as input the p-th

transaction of a user y and the profile r as target. Therefore, the transaction will be $\langle\ \#HC_x^i\ \ \#TID_y^p$ `content` $@r\ \rangle$.

The protocol works by verifying each part of this transaction tweet. Observe that the protocol does not consider the verification of the transaction content. Indeed, it is strictly related to the objective of the transaction, which is fully application dependent and orthogonal to our proposal.

Concerning the verification of $\#HC_x^i$, the protocol proceeds as follows. First, v checks whether $\#HC_x^i$ has been already used. If this is the case, the verification fails. Otherwise, the verifier computes the SHA-256 hash of $\#HC_x^i$. By construction, the result of this computation should be $\#HC_x^{i-1}$. Therefore, a search on Twitter for $\#HC_x^{i-1}$ should return the previous tweet posted by x. The goal of the verifier is to find either the previous *t-tweet* of x or the initial `hello` message. Now, since x also posts confirmation tweets for other users' transactions, $\#HC_x^{i-1}$ could refer to a confirmation tweet. If so, the SHA-256 hash of $\#HC_x^{i-1}$ is computed and the procedure is iterated until the goal is reached (i.e., either a *t-tweet* or the `hello` tweet is found). Let $\#TID_x^{i-1}$ be the ID of such a tweet. The verifier must now check whether he has confirmed this tweet in the past (i.e., he has posted a confirmation tweet with status 1). If this is not the case, then the verifier will not confirm (status 0) the new transaction. Otherwise, he will proceed with the check of the second part of the *t-tweet* of x. The verification of $\#TID_y^p$ implies the verification of the validity of the input transaction. We are supposing that the input is the p-th *t-tweet* of the user y. The validity of this tweet is induced by the presence of not less than t valid confirmation tweets generated by the s verifiers associated with y. Because each confirmation tweet contains the ID of the corresponding *t-tweet*, a search in Twitter for $\#TID_y^p$ will return all the confirmation tweets for the p-th *t-tweet* of p. Now, the verifier has to check both the presence of at least t confirmations with status 1 and that they have been posted by the legitimate verifiers for y (everyone in the community can verify this circumstance). Moreover, v verifies whether the target of $\#TID_y^p$ is really x and that it has not been already used as input in any other transaction. The last verification is the check of the existence of r (i.e., the target user of this new transaction) in the `Tweetchain` community. This is obtained by searching for $@r$ in Twitter and by verifying the presence of the `welcome` message in W along with t confirmations of the legitimate verifiers of r. At the end, if all the verification steps succeeded, then the verifier will post a confirmation tweet with status 1 for the new *t-tweet* of x, otherwise the status of this confirmation will be set to 0.

4 SECURITY ANALYSIS

In this section, we provide the security analysis of our approach showing that it accomplishes its objectives also in presence of attacks. We start by describing the security model and then we analyze the security properties of our protocols.

Our threat model includes the following assumptions:

- **A1** The attacker cannot add or compromise information shown on the social network accounts of any of the users of the `Tweetchain` community.
- **A2** Collision, preimage and second preimage attacks on the cryptographic hash function are infeasible.
- **A3** The attacker cannot know the secrets used by users to generate their hash chains.
- **A4** Given t verifiers the maximum number of those who do not respond according to the protocol to the verification request is $m \cdot t$ where $0 \leq m < 1$.
- **A5** At most t users can collude to break security properties of the protocol.

The last assumption merits a brief comment. We recall that, in our protocol, confirmations are produced collaboratively by several users playing as verifier. Some users might be corrupted by an adversary, but we assume an honest majority of users at all times. This is a common assumption borrowed by the field of e-voting [9, 12, 28]. As a consequence, our technique is parametric with respect to the value t. It is chosen in such a way that the likelihood that t randomly selected users misbehave is negligible. Similar considerations can be done for Assumption **A4**. Now, we are ready to identify the security properties (hereafter, **SP**) that our system has to assure and discuss the attacks and the countermeasures to contrast them.

SP1- Transaction Authenticity. A transaction and the user generating it can always be verified by the `Tweetchain` community.

Attack AA1: *An adversary tries to impersonate the welcome profile W to tamper the list of verifiers for a user.*

This attack is contrasted by the fact that the screen name of each account is unique in Twitter and that associated with W is known to all the `Tweetchain` community. Therefore, due to Assumption **A1** this attack cannot happen.

Attack AA2: *An adversary creates multiple accounts and tries to use them as verifiers of his own transactions.*

This attack is contrasted by two security mechanisms introduced in our approach: (i) the welcome profile yellow page service; (ii) the PRNG-based mechanism for verifiers assignment. The former forces each user who wants to join the `Tweetchain` community to perform a preliminary registration to W. After this, the latter security mechanism ensures that the election of verifiers for a given user is publicly verifiable as the PRNG-base strategy is entirely reproducible by each user in the `Tweetchain` community. Therefore, also thanks to Assumption **A1**, this attack cannot happen.

SP2 - Transaction Integrity. The whole message (*t-tweet*) representing a transaction cannot be tampered once posted on the system.

Attack AI1: *Some of the verifiers do not execute the verification protocol invalidating a transaction.*

Thanks to Assumption **A4** we know that, among t verifiers, the maximum number of those who may not collaborate properly by posting the confirmation tweet is $m \cdot t$, where $0 \leq m < 1$. To resist such attack we have to set the number of verifiers for any user of the community to $\frac{t}{1-m}$. Indeed, our aim is to obtain at least t confirmations to validate a new transaction. We know that if we consider an initial set of t verifiers, to obtain t confirmations we have to add $(m \cdot t)$ verifiers to this set, to compensate those who may not respond among the first t. Now, in the set of $(m \cdot t)$ added users, once again, we have that $m \cdot (m \cdot t)$ may not respond. Therefore to obtain at

least t confirmations we need to set the number of verifiers to $t + (m \cdot t) + (m^2 \cdot t) + \cdots + (m^i \cdot t) = t \cdot \sum_{i=0}^{\infty} m^i = \frac{t}{(1-m)}$. With this setting, such attack is contrasted.

Attack AI2: *Some of the verifiers collude to compromise the integrity of a transaction.*

This attack is contrasted by considering that, according to Assumption **A5**, at most t users can collude to perform an attack. Clearly, if we want to obtain t confirmations to validate a new transaction, to contrast the collusion attack, we should set the number of verifiers for each user to $2 \cdot t$. By combining this intuition with the countermeasure adopted to make the approach resistant to Attack **AI1** we have that the number of verifiers for each user has to be set to $\frac{2 \cdot t}{(1-m)}$.

Attack AI3: *Twitter, playing as an adversary, tries to compromise the integrity of a transaction by deleting a piece of the chain (even the whole) this transaction depends on.*

Suppose the adversary has as target a transaction q. The logic underlying this attack is that if the adversary is able to delete a partial or the whole set of transactions and corresponding confirmations related to q, the integrity of the latter is compromised as the verification of its input would fail. However, to prevent such an attack in our approach not only each transaction is linked in a chain of transactions posted by other users (horizontal chaining) but it is also linked, by means of the user hash chain, to messages generated by its owner before and after it (vertical chaining). Moreover, this reasoning applies also for the confirmations of each involved transaction, thus creating a mesh of chained transactions and confirmations around the attacker target, i.e. q. The deletion of any piece of this scheme will produce a domino effect causing all the transactions in this mesh (possibly extending to the entire community due to the small world property [13]) to be compromised. Clearly, the reasoning above holds if there are transactions in the network which are more recent than the piece of chain the adversary is planning to delete. Consider the improbable situation in which all the transactions involved in the piece of the horizontal chain under attack are the last posted by the corresponding users. In this case, to contrast such an attack, we enforce that a transaction, already validated through the verification protocol, can be considered in a *safe* state only if there are at least h transactions generated after any element of the horizontal chain linking it (a similar measure is adopted also in Blockchain). This reasoning is sketched in Figure 3, in which the horizontal chaining, the vertical chaining, and the depth h (necessary to consider the *safe* state for a transaction) are highlighted.

SP3 - No repudiation of Transactions. The user generating a transaction cannot repudiate it.

Attack AR1: *An adversary tries to make ambiguous a transaction by forging another one with the same input transaction.*

In this case, the attacker tries to create a fork in the chain using the same input transaction but with different recipient. However, this attack cannot occur due to the verification mechanism according to which each verifier has to check for the presence of the input in a previous transaction (see Section 3). Therefore, also thanks to Assumption **A1**, the attempt of creating a fork is demised because the confirmations will have *status* 0.

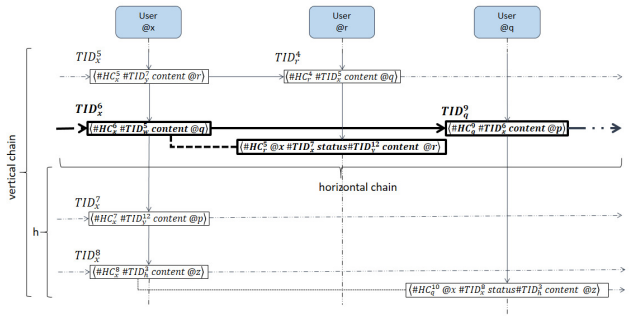

Figure 3: A fragment of the grid structure created by our protocol.

Attack AR2: *The user, acting as an adversary, tries to repudiate a transaction by deleting the corresponding t-tweet.*

This attack can be performed in two ways: *(i)* the adversary deletes the *t-tweet* before all the confirmations are produced; *(ii)* the adversary deletes the *t-tweet* after the confirmations. In the first case, the transaction will be considered not valid by the verification protocol and hence, it will be not usable as input for future transactions. Therefore, no real advantage is produced with this attack. In the second case, the adversary tries to make the transaction valid by waiting for all the confirmations to be produced, and then proceeds by deleting the original *t-tweet* with the objective of repudiating the transaction itself. However, the attack fails as, according to the verification protocol and due to Assumptions **A4** and **A5**, the transaction will be considered valid because at least t confirmations will be found.

5 PERFORMANCE ANALYSIS

Hereafter, we describe an experimental campaign to evaluate the performance of our system when it comes to time execution. We built a Java prototype implementing our protocol and all the experiments were executed on a personal computer with a 4.0 GHz Intel I7 CPU, 8 GB of RAM. As for the adopted software, the operative system was Ubuntu 16.10 equipped with JDK, Apache and Tomcat servers.

We measured that the authentication procedure lasts a time interval t_c which is less than 50 milliseconds on average. Another common aspect for the two functionalities is the computation of SHA-256 hashes used in both *t-tweets* and confirmation tweets. From our testing, we found that this time is negligible w.r.t. network times, therefore we do not consider it in our measurements.

With that said, we proceed with our analysis by focusing on the two functionalities independently. We start by considering the Transaction generation procedure. As a first step, the user computes a new element of his hash chain and then he builds the new tweet. We measured that the time t_b needed to build a *t-tweet* and send it to Twitter is about 1.8 seconds, moreover once a tweet is ready for publication on the Twitter platform, the time t_p for the actual posting is about 1.5 seconds. Still in the Transaction

generation procedure we also consider the time needed to notify the verifiers of the presence of a new *t-tweet*. This time depends on the implementation of the notification task. However, we refer to a scenario in which verifiers have some application obtaining notifications about new tweets via the Twitter Streaming APIs. Therefore, we measured the streaming time t_s and obtained that it takes 2.1 seconds on average. As a consequence, the overall time for the Transaction generation procedure is $t_g = t_c + t_b + t_p + t_s \leq 5.90$ seconds.

Concerning the Verification step, a preliminary task is to measure the time, say t_r, to perform a search for a given hashtag. From our tests we obtained that t_r is about 1.8 seconds. Moreover, we measured that the time t_a for the search of an account in Twitter is about 1.9 seconds. Now, the verification of the hash chain of the user who posted a *t-tweet* requires a time t_h ranging from 1.8 to 6.0 seconds. This variation in the value of t_h is inherent to the number of attempts required to find the previous element of the hash chain associated with a *t-tweet*. The verification time of the input transaction for a *t-tweet* is bounded by the time required to search for at least t confirmation tweets with status 1. Because this search is done via hashtag, this time is equal to t_r. As a last step, a verifier has to check the existence of the target profile of the *t-tweet* and this time is equal to t_a. Finally, the overall time for the post of the confirmation tweet is $t_b + t_p$.

In summary, the time required for the verification of a *t-tweet* is $t_v = t_c + t_b + t_p + t_h + t_r + t_a \leq 13.5$. From this result, it emerges that the performance of our technique is better than the state of the art. Indeed, the equivalent time in Blockchain is about 10 minutes (it depends on the payed fee), whereas in Ethereum it is about 15 seconds in the best case (i.e., only when the proof of stake is considered critical).

6 CONCLUSION

A great attention towards Blockchain has been recently devoted by both researchers and companies, due to its high applicative power, mainly relying on the idea that Blockchain can implement a public, shared ledger without dedicating any trusted entity.

Starting from the observation that Blockchain might be not suitable for IoT applications, we designed an alternative public ledger based on Twitter overcoming such limitations.

We highlight that `Tweet- chain` ensures, if required, the same anonymity degree as Blockchain. Instead of different public keys, the user may use different Twitter profiles and may access the network by TOR to obtain privacy. On the contrary, if we want to use `Tweetchain` for those applications in which the identity of actors is a required feature, we can enforce the use of only verified Twitter profiles with double factor strong authentication [1] and interfacing our protocol to some public digital identity system.

REFERENCES

[1] 2016. Twitter authentication. https://support.twitter.com/articles/20171580. (2016).

[2] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. 2005. Fault-scalable Byzantine fault-tolerant services. In *ACM SIGOPS Operating Systems Review*, Vol. 39. ACM, 59–74.

[3] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. 2012. On bitcoin and red balloons. In *Proceedings of the 13th ACM conference on electronic commerce*. ACM, 56–73.

[4] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. 2014. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* (2014).

[5] Lear Bahack. 2013. Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft). *arXiv preprint arXiv:1312.7013* (2013).

[6] Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. 2016. A model to support design and development of multiple-social-network applications. *Information Sciences* 331 (2016), 99–119.

[7] Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. 2017. Tweetchain: An Alternative to Blockchain for Crowd-Based Applications. In *International Conference on Web Engineering*. Springer, 386–393.

[8] Francesco Buccafurri, Gianluca Lax, Antonino Nocera, and Domenico Ursino. 2015. A system for extracting structural information from Social Network accounts. *Software: Practice and Experience* 45, 9 (2015), 1251–1275.

[9] Ronald Cramery, Rosario Gennaroz, and Berry Schoenmakersx. 1997. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications* 8, 5 (1997), 481–490.

[10] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.

[11] Youssou Faye, Ibrahima Niang, and Thomas Noel. 2011. A survey of access control schemes in wireless sensor networks. *Proc. World Acad. Sci. Eng. Tech* 59 (2011), 814–823.

[12] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. 2001. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography*. Springer, 90–104.

[13] Jon M Kleinberg. 2000. Navigation in a small world. *Nature* 406, 6798 (2000), 845–845.

[14] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2015. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *University of Maryland and Cornell University* (2015).

[15] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S Rosenschein. 2015. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 919–927.

[16] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 528–547.

[17] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 127–140.

[18] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[19] Joseph Poon and Thaddeus Dryja. 2015. The bitcoin lightning network. (2015).

[20] Dorit Ron and Adi Shamir. 2013. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*. Springer, 6–24.

[21] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[22] Brian L Shultz and Dave Bayer. 2015. Certification of Witness: Mitigating Blockchain Fork Attacks. (2015). http://bshultz.com/paper/Shultz_Thesis.pdf.

[23] Yonatan Sompolinsky and Aviv Zohar. 2013. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. *IACR Cryptology ePrint Archive* 2013 (2013), 881.

[24] Melanie Swan. 2015. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.".

[25] Tim Swanson. 2015. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. (2015). R3, Tech. Rep., 6 Apr. 2015. http://www.ofnumbers.com/2015/04/06/.

[26] Marie Vasek, Micah Thornton, and Tyler Moore. 2014. Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In *International Conference on Financial Cryptography and Data Security*. Springer, 57–71.

[27] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014).

[28] Aneta Zwierko and Zbigniew Kotulski. 2007. A light-weight e-voting system with distributed trust. *Electronic Notes in Theoretical Computer Science* 168 (2007), 109–126.