

# On Formalising and Analysing the Tweetchain Protocol

Mariapia Raimondo<sup>1</sup><sup>a</sup>, Simona Bernardi<sup>2</sup><sup>b</sup> and Stefano Marrone<sup>1</sup><sup>c</sup>

<sup>1</sup>*Dip. di Matematica e Fisica, Università della Campania “Luigi Vanvitelli”, Caserta, Italy*

<sup>2</sup>*Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain*

**Keywords:** Distributed Ledger Technology, Formal Modelling, Tweetchain, Tamarin Prover, Model Checking.

**Abstract:** Distributed Ledger Technology is demonstrating its capability to provide flexible frameworks for information assurance capable of resisting to byzantine failures and multiple target attacks. The availability of development frameworks allows the definition of many applications using such a technology. On the contrary, the verification of such applications are far from being easy since testing is not enough to guarantee the absence of security problems. The paper describes an experience in the modelling and security analysis of one of these applications by means of formal methods: in particular, we consider the Tweetchain protocol as a case study and we use the Tamarin Prover tool, which supports the modelling of a protocol as a multiset rewriting system and its analysis with respect to temporal first-order properties. With the aim of making the modeling and verification process reproducible and independent of the specific protocol, we present a general structure of the Tamarin Prover model and of the properties to be verified. Finally, we discuss the strengths and limitations of the Tamarin Prover approach considering three aspects: modelling, analysis and the verification process.

## 1 INTRODUCTION

The digitization of the economy and, more in general, of the society is an unstoppable process that recent times are just accelerating. Being able to track events and being sure of the identity of the parties involved in a transaction are cornerstones of both digital economies and smart societies and their enabler, the Internet of Things (IoT) technology.

This non-repudiation property is guaranteed by a new family of technologies: the Distributed Ledger Technology (DLT). Even if the concept of DLT is not new, its mass diffusion started with Bitcoin (Nakamoto, 2009) and other blockchain applications: a comprehensive explanation of the differences between blockchain and DLTs can be found in (Panwar and Bhatnagar, 2020). From a security perspective, the literature (Lin and Liao, 2017; Yu et al., 2018; Lee and Jang, 2020; Dai et al., 2017; Kumar and Mallick, 2018) analyses a number of possible attacks on blockchain, even if their practical relevance has not been demonstrated.

Blockchain received a lot of interest not only from the academic community but also from the indus-

trial one: the presence of development frameworks is spreading the usage of this technology by helping software designers in integrating blockchain in their applications and in getting its benefits. Such frameworks differ from each other by many features: kind of proof of stake, support to smart contracts, being permissioned/permissionless, etc.

Some examples of such frameworks are:

- Exonum: a blockchain framework developed for the enterprise panorama in 2017. It is based on a Rust-based open source code and Application Programming Interface (API)<sup>1</sup>;
- Hyperledger Fabric: oriented to the development of high-scaling enterprise applications in a flexible-permissioned environment<sup>2</sup>;
- Openchain: designed for issuing and managing digital assets, it is more oriented to financial settings. It relies on smart contract modules<sup>3</sup>;
- Ripple: a permissioned open source framework, based on a Python API, it is based on probabilistic voting. Features like “issuance” allow users to lock a particular asset and transfer it and, ul-

<sup>a</sup> <https://orcid.org/0000-0001-7483-7987>

<sup>b</sup> <https://orcid.org/0000-0002-2605-6243>

<sup>c</sup> <https://orcid.org/0000-0003-1927-6173>

<sup>1</sup><https://github.com/exonum>

<sup>2</sup><https://github.com/hyperledger/fabric>

<sup>3</sup><https://github.com/openchain>

timately, lowers the transaction cost, speeding up the transaction while maintaining transparency<sup>4</sup>.

One of the most challenging contexts where blockchain promises a greater impact is the field of the IoT; such challenges are inherent to the nature of the devices used in IoT. These devices present the following features that are hard to match with the resources needed by DLTs and, more in general, by cryptographic applications:

- limited computational power, that does not allow very cheap devices to participate to the blockchain;
- absence of specific cryptographic hardware capable of accelerating such computations;
- low capacity of storing large transaction footprints;
- limited battery that contrasts with the high energy demand due to the continuous interaction with the Peer-to-peer (P2P) network.

These applications are becoming more and more pervasive in both the economic and societal texture so that they can be considered business critical applications. The capability to sell by software companies operating in this field, also relies on their capabilities to certify blockchain-based software and to guarantee that the theoretical properties of DLTs are matched by concrete protocols and software. Formal validation techniques would constitute a cornerstone in the diffusion of such approaches.

The main objective of this work is to provide a concrete experience in the modelling and analysis of a block-chain based protocol with formal methods. Rather than providing an overview and a generic comparison of such methods, this paper focuses on one among the most promising methods, to clearly highlight its strengths and weaknesses. In particular, the formalism and the underlying solution tool used in this paper is the well-known Tamarin Prover (Meier et al., 2013), which is used since 2013 in the modelling of cryptographic mechanisms.

Even if many of the concerns dealt with in this paper can be generalized to a generic blockchain application, the objective of this paper is to present an application of Tamarin Prover modelling approach to a case study: the Tweetchain protocol defined in (Buccafurri et al., 2017a; Buccafurri et al., 2017b). This protocol perfectly fits with the scope of this work since it is meant to provide a light-weight public ledger for IoT and crowd-sourcing applications.

The paper is structured as follows: Section 2 provides the background on Tweetchain protocol and

Tamarin Prover. Section 3 shows the structure and the details of the Tweetchain modelling and analysis. Section 4 reports some considerations on the presented experience. Section 5 presents related works; Section 6 summarises the paper and draws future research directions.

## 2 BACKGROUND

In this section, basic concepts of Tweetchain and Tamarin Prover are presented in Subsections 2.1 and 2.2, respectively.

### 2.1 The Tweetchain Protocol

The Tweetchain protocol has been developed and introduced in (Buccafurri et al., 2017a; Buccafurri et al., 2017b) to overcome the problems in applying full public permissionless ledger based mechanisms in the IoT domain. Tweetchain exploits the pervasiveness of Online Social Networks (OSN) in human-to-machine and machine-to-machine communication; the usage of OSN compensates the limitations of pure blockchain-based approaches.

In fact, the Tweetchain community ( $C$ ) is mediated by a special user, the welcome profile ( $W$ ), whose scope is to provide a sort of yellow pages for all the “regular users” ( $x$ ). Let us suppose that: (1) each party has a Twitter account and signs in; (2) a generic party  $u$  owns a chain of hash values  $\#HC_u^1, \#HC_u^2, \dots, \#HC_u^{(i-1)}$  where the hash values are the digests of the  $i-1$  messages already sent by  $u$ . The Tweetchain registration scenario is specified by the Unified Modeling Language (UML) sequence diagram of Figure 1.

Before interacting in any way with other Tweetchain actors, each party has to sign in Twitter. As a first step to join  $C$ ,  $x$  publishes a *Hello tweet*, containing the first elements  $HC_x^1$  and  $HC_W^1$  of the hash-chains of  $x$  and  $W$ , respectively, to register itself to  $W$ .  $W$ , after verifying the tweet, sends a *Welcome tweet* as a response. The latter contains the hash value  $\#HC_W^i$ , computed on the base of its hash-chain, and the Twitter id ( $\#TID_x^1$ ) of the *Hello* message sent by  $x$ <sup>5</sup>.

The final step of the registration procedure prescribes that  $x$  chooses among the members of  $C$  its set of verifiers  $V_x = \{v_{x,s}\} \subset C$ , where  $s$  is described in (Buccafurri et al., 2017a). To this aim,  $x$  generates this subset according to a publicly known pseudo-

<sup>4</sup><https://github.com/ripple>

<sup>5</sup>In Twitter, each tweet has a unique id.

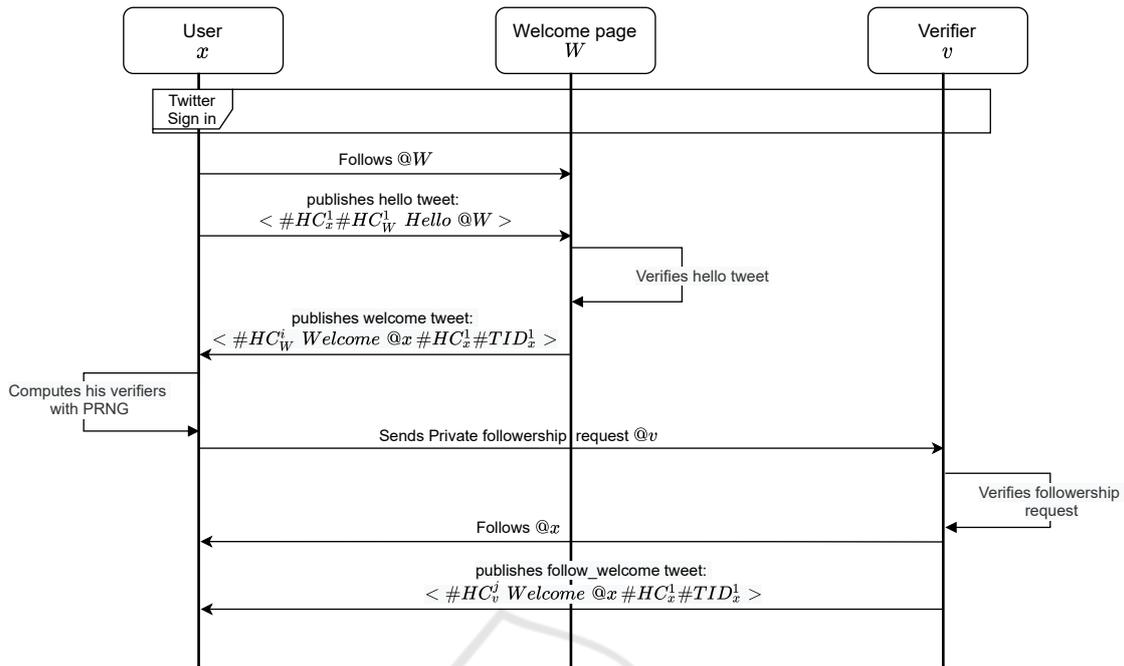


Figure 1: The Tweetchain registration scenario.

random algorithm and a public seed<sup>6</sup>. The use of publicly verifiable information, in this phase, is motivated by the necessity of assuring that each party in  $C$  does not choose its set of verifiers in a malicious way. After this check, each  $v_{x,s}$  confirms and follows the user  $x$  and publishes a *Follow\_welcome tweet*.

Once the registration has been carried out, the user  $x$  belongs to the community, thus he/she is able to generate a transaction intended for another user  $r$  of  $C$ . A transaction, in this context, corresponds to the posting of a *t-tweet*, a particular tweet reporting the last element  $i$  of the hash-chain of the sender  $HC_x^i$ , the Twitter id  $TID_y^p$  of latest approved transaction (i.e., the  $TID$  of the  $p$ -th *t-tweet* posted by user  $y$ ) and, clearly, the reference to the intended recipient. In particular,  $TID_y^p$  plays the role that the cryptographic hash of the previous block (included in the header of another block) plays in a blockchain, i.e., it allows to link them.

As soon as  $x$  posts a *t-tweet*, the users of  $V_x$  have to validate the transaction using a particular protocol shortly explained below. After the check, a verifier  $v$  posts a *confirmation tweet* containing: the last element  $j$  of its hash-chain  $HC_v^j$ , the content of the transaction, the Twitter id  $TID_x^i$ , of the transaction to validate, and  $TID_y^p$ , of the previous one, to link them, and the boolean variable *status* that represents the result

<sup>6</sup>As a seed, the protocol may use the Twitter user id that is unique and publicly available, as proposed by (Buccafurri et al., 2017a); this choice is not mandatory.

of the validation process (1 - success, 0 - failure).

The verification protocol for a verifier  $v$  consists of the following tasks:

1. checking that the sender has not already used the hash-chain element  $HC_x^i$  involved in the *t-tweet*;
2. using  $HC_x^i$  (or a previous elements in the hash-chain) to search on Twitter for the related tweet until a *Hello tweet* or another *t-tweet* is found;
3. checking that  $v$  has validated the tweet found in the previous step;
4. verifying the validity of the transaction related to  $TID_y^p$ , i.e., looking for a "sufficient number" of confirmation tweets on Twitter, posted by the verifiers of  $y$ <sup>7</sup>;
5. verifying that the recipient of the transaction belongs to the community.

To summarize, the underlying idea of Tweetchain is to define 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 consensus on transactions is based on the presence of a sufficient number of valid confirmation tweets from the community.

<sup>7</sup>The meaning of *sufficient* is clarified in (Buccafurri et al., 2017a)

## 2.2 The Tamarin Prover

The Tamarin Prover (Meier et al., 2013) is a symbolic model checker largely used for the modeling and analysis of cryptographic protocols. This tool has proved its suitability in analysing cryptographic protocols used in different application domains, which span from 5G (Basin et al., 2018) to vehicular networks (Whitefield et al., 2017) and trusted computing, using Elliptic Curve Cryptography (ECC) (Whitefield et al., 2019). The current version of the tool is 1.6.0 and can be downloaded under GNU's GPL v3.0<sup>8</sup>.

According to the classical model checking approach, a security protocol model and a specification of the desired properties have to be fed to the Tamarin Prover to begin the analysis process; furthermore, the tool also allows the user to add rules to check the behaviour of the adversary.

Protocols and properties can be specified, respectively, with labeled multiset rewriting *rules* and first order logic formulas over symbols (i.e., the *lemmas*). The adversary model, set by default, is the Dolev-Yao model (Dolev and Yao, 1983): such an adversary controls the network and can delete, inject, modify and intercept messages on the network.

There are two ways of constructing proofs (i.e., analysing the model): the *automated mode*, that combines deduction and equation reasoning with heuristic to guide the proof search; and the *interactive mode*, which allows the user to manually guide the proof search while still exploiting the automated proof search efficiency. In both the cases, the Tamarin Prover generates a Labeled Transition System (LTS) from the protocol and the adversary models, where: (1) the states are multisets of *facts*, and the initial state is the empty set; and (2) each transition transforms a state (i.e., a multiset of facts) into another state, according to the used rewrite rule.

Security properties to be verified are specified using *lemmas*, which are identified by a name and a guarded first-order formula over the action facts (i.e., the transition labels). There are two types of lemmas: *exists-trace* lemmas, to check the existence of a trace holding the property, and *all-traces*, to verify whether the property holds for all the possible traces.

Moreover, it is possible to constrain the state-space exploration by means of special lemmas named *restriction*, which allow the analyst to define properties that each trace must satisfy.

Finally, when modelling a non-trivial protocol, it might happen to run into the *partial deconstruction*, which can lead to non-termination when verifying lemmas. A way to overcome this problem is to use

<sup>8</sup><https://github.com/tamarin-prover>

the keyword *sources* in some lemmas, which will help in the pre-computation phase. Recently, the Tamarin Prover developers have improved the tool providing support to the users by automating the generation of sources lemmas (Cortier et al., 2020).

## 3 MODELLING AND ANALYSING TWEETCHAIN

This section describes the modelling of the Tweetchain protocol and a preliminary security analysis of the protocol with the Tamarin Prover tool.

In particular, Figure 2 depicts the overall structure of the Tamarin Prover model, that includes both the model of the protocol and the security properties to be verified on the model itself during the analysis. The model of the protocol is detailed in Subsection 3.1, whereas the specification of the security properties and the results of the analysis are described in Subsection 3.2.

### 3.1 The Model of the Protocol

The model of the Tweetchain protocol (top part of Figure 2) has been developed trying to preserve the notation and the structure used in (Buccafurri et al., 2017a) as far as possible. In particular, the model is composed of six sections: *Entering Twitter*, *Joining Community*, *Utility rules*, *Followership*, *Transaction*, and *Restriction*. The first five sections correspond to Tamarin rules while the last one contains Tamarin axioms.

The *Entering Twitter* section, corresponds to the initial fragment *Twitter Sign in* of the sequence diagram in Figure 1, and the action of signing into Twitter is modelled with two Tamarin rules, `create_entity_W` and `create_entity`, which create the Welcome page *W* and a generic user *x*, respectively. The need to differentiate *W* and all the users that want to join the community arises from the necessity of avoiding modelling (and analysing) issues. In fact, with the rule `create_entity_W`, it is generated a persistent fact `!Community_ID_W(...)`, which contains the necessary information that any user *x* needs to check the authenticity of a tweet posted from *W*, and the hashchain of *W*, that has to be available to a generic user *x* that aims at joining the community, whilst the hashchain of *x* bears only when he/she sends a *Hello* tweet.

The *Joining Community* section includes three Tamarin rules: `hello_tweet`, `welcome_tweet` and `community_joined`. There are a few highlights that are worth mentioning: the first rule models the action

```

1 rule hello_tweet:
2   let
3     HC_1_x = h(~root_HC_x)
4   in
5     [ St_0(x, id_x, pubk_x, ltk_x), !Community_ID_W($W, id_W, pubk_W, HC_1_W),
6       Fr(~TID_1_x), Fr(~root_HC_x) ]
7   --[ Neg(x,$W), HELLO_sent(x, $W), UniqueHello(x) ]->
8     [ St_1(x, id_x, pubk_x, ltk_x, HC_1_x, 'HELLO_sent', ~TID_1_x), HC(x, <'empty', HC.1.x>),
9       Out(<<x, HC_1_x, HC_1_W, 'Hello', $W>, sign(<x, HC_1_x, HC_1_W, 'Hello', $W>, ltk_x)>),
10      Out(<<x, ~TID_1_x>, sign(<x, ~TID_1_x>, ltk_x)>) ]

```

Listing 1: Specification of the hello\_tweet rule.

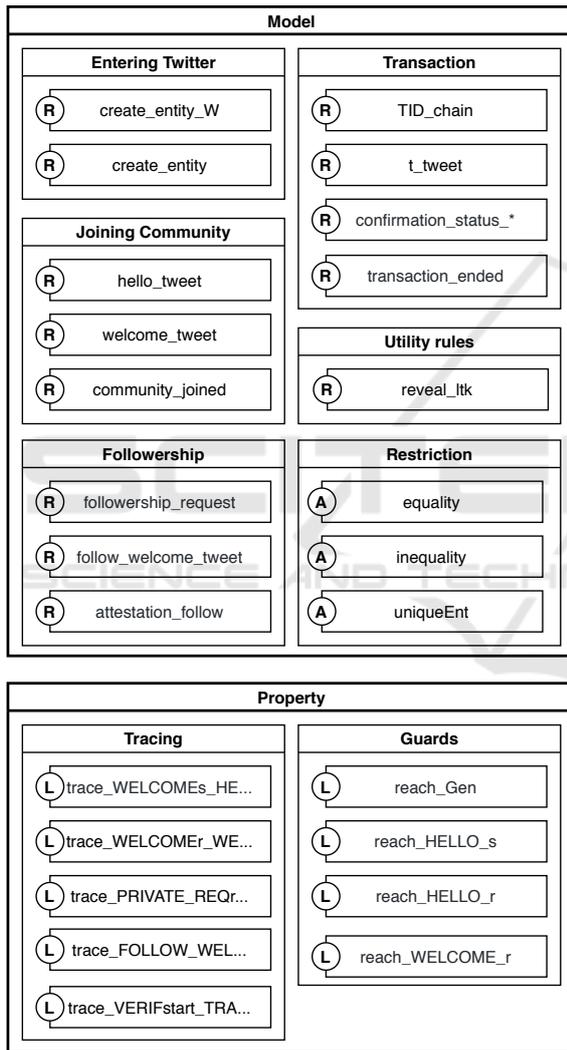


Figure 2: The structure of the Tamarin Prover model.

of posting a *Hello* tweet by a user  $x$ , who wishes to join the community, and contextually creates his/her hash-chain ( $HC(\dots)$  fact, highlighted in Listing 1).

The second rule models the reception of the *Hello* and the sending of the *Welcome* messages by  $W$ ; the latter represents the confirmation that  $x$  has joined the

community. The state reached by  $x$  is modelled by a persistent fact  $!Community\_ID(\dots)$ .

The last part of the registration (i.e., *Followership* phase) concerns the building of the set  $V_x$  of the followers of  $x$ , which are designated as verifiers of the transactions generated by  $x$ . We propose a simplified model for this phase, as the entire sequence of interactions would be tricky to model and to manage: concretely, we model the verification step that  $x$  sends the follow request to a user  $y$  who belongs to the community. The Followership phase is modelled by means of three rules: *followership\_request*, *follow\_welcome\_tweet* and *attestation\_follow*.

The *Transaction* section deals with the generation and confirmation of a new transaction. Let us assume a user  $x$ , belonging to the community, wants to generate a new transaction addressed to a recipient  $r$ . The first task  $x$  has to do is to retrieve the chain containing the IDs of the approved transactions; such a retrieved chain is generated by means of another Tamarin rule (i.e., *TID\_chain*).

The transaction is modelled with the rule *t\_tweet* (shown in Listing 2). The application of this rule generates a new fact  $TID\_temp(\dots)$ , where the ID of the transaction is stored, whilst the  $TID\_chain(\dots)$  is carried unchanged. This two-step mechanism is mandatory as a new transaction ID can be added to the chain only when it is approved by the community. Moreover, user  $x$  ensures that user  $r$  belongs to the community, too. At last, the transaction verification is modelled by two rules *confirmation\_status\_1* (shown in Listing 3) and *confirmation\_status\_0*. Both rules recover the  $TID\_chain$  and  $TID\_temp$  and check that: 1) the user  $x$  has used its last hash-chain element in the transaction ( $Eq(\dots)$  fact), and 2) that the verifier  $v$  and the recipient  $r$  are two distinct users ( $Neq(\dots)$  fact). Then, depending on the success or failure of the checks,  $v$  posts a new tweet containing a boolean variable *status* set to 1, or 0, and updates, or leaves unchanged, the  $TID\_chain$ , respectively. The last rule of the protocol is *transaction\_ended* and models the reception of the *confirmation tweet*.

```

1 rule t_tweet:
2   let
3     HC_new_x = h(HC_b_x)
4   in
5     [ St_x_2(x, id_x, pubk_x, ltk_x, HC_1_x, 'FOLLOW_WELCOME_received', TID_1_x),
6       HC(x, <HC_a_x, HC_b_x>), !Community_ID(r, id_r, pubk_r, HC_1_r),
7       TID_chain(<issuer_prec, TID_prec>), Fr(~TID_x) ]
8   -- [ Neq(x,r), TRANSACTION_sent(x,r,~TID_x), OUT_TRANS(TID_prec) ]->
9     [ St_x_3(x, id_x, pubk_x, ltk_x, HC_1_x, 'TRANSACTION_sent', TID_1_x),
10      HC(x, <HC_b_x, HC_new_x>), TID_chain(<issuer_prec, TID_prec>), TID_temp(<x, ~TID_x>),
11      Out(<x, ~TID_x>), Out(<<x, HC_new_x, TID_prec, 'content', r>,
12      sign(<x, HC_new_x, TID_prec, 'content', r>, ltk_x) >) ]

```

Listing 2: Specification of the t\_tweet rule.

```

1 rule confirmation_status_1:
2   [ St_2(v, id_v, pubk_v, ltk_v, HC_1_v, 'WELCOME_received', TID_1_v),
3     !Community_ID(x, id_x, pubk_x, HC_1_x), TID_chain(<issuer_succ, TID_succ>), TID_temp(<x, TID_x>),
4     HC(x, <HC_old_x, HC_new_x>), HC(v, <HC_old_v, HC_new_v>),
5     In(<<x, HC_new_x, TID_succ, 'content', r>, auth_msg >) ]
6   -- [ Neq(v,x), Neq(v,r),
7     Eq( verify(auth_msg, <x, HC_new_x, TID_succ, 'content', r>, pubk_x), true),
8     Eq( h(HC_old_x), HC_new_x), VERIF_starts(v,x) ]->
9     [ St_v_3(v, id_v, pubk_v, ltk_v, HC_1_v, 'CONFPOS_sent', TID_1_v), TID_chain(<x, TID_x>),
10      Out(<<v, HC_new_v, x, TID_x, '1', TID_succ, 'content', r>,
11      sign(<v, HC_new_v, x, TID_x, '1', TID_succ, 'content', r>, ltk_v) >) ]

```

Listing 3: Specification of the confirmation\_status\_1 rule.

To facilitate the analysis process with the Tamarin Prover, the model includes some additional features in the Utility Rules and Restriction sections.

The *Utility Rules* section contains the `reveal_ltk` rule that can be used when the attacker should not be able to retrieve some information. In particular, during the analysis, this rule checks if the adversary has stolen some private information of a party (e.g., a private key).

For the same sake of analysis complexity reduction, we defined a set of restrictions in the Restriction section: equality and inequality, and `uniqueEnt`, which ensures that each entity involved in the protocol is created only once.

### 3.2 Security Analysis of the Protocol

The security analysis of the protocol is carried out using the model, described in Subsection 3.1, and the security properties to be verified (bottom part of Figure 2), which are herein described.

Before specifying the security properties of interest, we performed a preliminary analysis aimed at checking the correctness of the model. Concretely, we defined a set of lemmas that enable to check the states reachability of the LTS. The structure of a reachability lemma is simple: it checks the existence of the necessary entities and a timepoint for which a specific

rule is used, i.e., a specific state of the LTS is reached from the initial state. These lemmas are marked with a `reach...` name.

An example of reachability lemma is shown in Listing 4 that is used to verify the execution of the `attestation_follow` rule, defined in the protocol model.

```

1 lemma reach_FOLLOW_WELCOME_s: exists--trace
2   " Ex y x #i. FOLLOW_WELCOME_sent(y,x) @i"

```

Listing 4: Specification of a reachability lemma.

This rule models the sending action by a verifier  $y$ , after the reception of a followership request sent by  $x$ . In particular, the lemma checks the reachability of the state `FOLLOW_WELCOME_sent` at timepoint  $i$ , where the verifier  $y$  sends a *Follow\_welcome* tweet to  $x$ .

The security properties of interest concern the authenticity of the messages sent over Twitter. In particular, with reference to the analysis carried out in (Buccafurri et al., 2017a), we checked whether the Transaction Authenticity property holds for the modelled protocol<sup>9</sup>. We did not analyze the other two properties, considered in the work by (Buccafurri et al.,

<sup>9</sup>This property is meant to verify that each transaction can always be verified by the Tweetchain community: see **SP1** in (Buccafurri et al., 2017b).

2017b), because of the modelling simplification of the verification step adopted in this work.

The authenticity is verified for each exchanged message, i.e., for each pair of send-receive rules, and it corresponds to check the following property in the LTS: “If user  $x$  receives a message  $m$  from user  $y$ , then  $y$  has sent  $m$  to  $x$  earlier”. In the protocol model, the sender authenticity is obtained by signing the messages with his/her private key, and any user can verify the identity of the sender using the built-in message theory of Tamarin that defines a signature scheme.

Thus, for a given message  $m$  exchanged in the registration scenario shown in Figure 1, we defined two different lemmas for the verification of its authenticity that account of the capability of the adversary to steal the private key of the parties involved in the communication. Such lemmas are labeled as `fake_trace...` and `trace...`: in the former, the adversary is able to steal confidential data (i.e., the private keys) whereas in the latter, he/she is not.

Listing 5 and Listing 6 show the two types of lemmas used to check the authenticity of the `follow_welcome` tweet message. The lemma in Listing 5 can be read as follows: “for each `follow_welcome` tweet, received by  $x$  at time instant  $i$  with  $y$  as a sender, then there exists a time instant  $j$ , subsequent to  $i$ , in which  $y$  has sent such a message to  $x$ .”

```

1 lemma fake_trace_FOLLOW_WELCr_FOLLOW_WELCs :
2   "All y x #i. FOLLOW_WELCOME_received(y,x) @i
3     ==>
4     Ex #j. FOLLOW_WELCOME_sent(y,x) @j & j<i"
```

Listing 5: Specification of a `fake_trace...` lemma.

The `trace...` lemma (Listing 6) has two more constraints (lines 3-4) in the premise than the previous lemma (Listings 5). Such constraints limit the capability of the adversary to steal the private key of any user involved in the communication, i.e., the adversary is not able to impersonate him/her.

```

1 lemma trace_FOLLOW_WELCr_FOLLOW_WELCs :
2   "All y x #i. (FOLLOW_WELCOME_received(y,x)
3     @i
4     & not (Ex #r. RevLtk(x) @r)
5     & not (Ex #r. RevLtk(y) @r))
6   ==>
7   Ex #j. FOLLOW_WELCOME_sent(y,x) @j & j<i"
```

Listing 6: Specification of a `trace...` lemma.

It is straightforward that for these messages it is crucial to check that the adversary cannot impersonate other users, and, thus, we expect that the analysis

gives different results between `fake_trace...` and `trace...` lemmas. The analysis results of the most relevant lemmas are reported in Table 1.

Table 1: Results for the Tweetchain protocol.

Lemma	Result
<code>fake_trace_WELCOMEs_HELLOs</code>	false
<code>trace_WELCOMEs_HELLOs</code>	true
<code>fake_trace_WELCOMER_WELCOMEs</code>	false
<code>trace_WELCOMER_WELCOMEs</code>	true
<code>fake_trace_PRIVATE_REQr_PRIVATE_REQs</code>	false
<code>trace_PRIVATE_REQr_PRIVATE_REQs</code>	true
<code>fake_trace_FOLLOW_WELCr_FOLLOW_WELCs</code>	false
<code>trace_FOLLOW_WELCr_FOLLOW_WELCs</code>	true

As mentioned in Subsection 2.2, when modelling large protocols in Tamarin, it might happen to run into partial deconstruction, and it happened for the Tweetchain protocol, too. In fact, most of the proofs of non-trivial lemmas of our model did not terminate, because the tool had troubles in determining the source of a variable (namely, the tweet ID `TID`).

To overcome this issue, a `source` lemma has been added. Such a lemma helps in the pre-computation phase, as it adds information useful to the model checker. In particular, in this case, the `source` lemma expresses that whenever a `t_tweet` (containing the `TID` variable) is used, then either the adversary stole and sent it over the public channel or a `TID_chain` rule has been applied before.

Table 2 shows the comparison of the model analysis with and without the source lemma.

Table 2: Comparison of verification with and without `source` lemma.

Lemma	Source	No Source
<code>reach_FOLLOW_WELCOME_s</code>	true	non-term
<code>reach_TRANSACTION_s</code>	true	non-term
<code>fake_trace_WELCs_HELLOs</code>	false	false
<code>trace_WELCs_HELLOs</code>	true	non-term
<code>fake_trace_PR_REQr_PR_REQs</code>	false	non-term
<code>trace_PR_REQr_PR_REQs</code>	true	non-term
<code>fake_trace_F_WELCr_F_WELCs</code>	false	non-term
<code>trace_F_WELCr_F_WELCs</code>	true	non-term

Finally, some data about the analysis times are reported. Far from being a formal performance analysis of Tweetchain model solution, the proposed model has been solved in batch mode using the following HW/SW configuration: a Linux Ubuntu Server 20.04.1 LTS hosting a Tamarin 1.6.0 and running on a quad-core Intel(R) Xeon(R) CPU E5-2650L v4 1.70GHz, with 8 Gb of RAM. In case of analysis termination, it took at most few minutes to analyse each lemma.

## 4 DISCUSSION AND LIMITATIONS

The discussion of the main results of the case study is reported in three main threads: modelling, analysis and process.

### 4.1 Modelling

The language of Tamarin has been proved to be a good tool for the modelling of security features in communication protocols.

This fact is because Tamarin provides primitives oriented to security modelling: elements as hashing, encryption, fresh value computation, represent a valuable aid for the modeller.

On the other hand, the difference in the scope between Tamarin and other “general-purpose” model checkers is reflected by some modelling choices that make Tamarin not perfectly fitting into the problem here tackled. One of the main limitations of the language is constituted by the absence of a way to model arrays. Such a modelling element is present in other model checkers — e.g., NuSMV and SPIN — and allows to model hash chains, a crucial passage in the modelling of the verification steps.

Another passage in the Tweetchain verification step that is hard to model is the consensus mechanism where a sufficient number of verifiers agree with the sent transaction. The difficulty relies upon that, as well as for integers and bytes, there is no possibility to model arrays of protocol participants; hence, to model a majority voting would be very hard in Tamarin.

In the proposed model, we have overcome all these difficulties by reducing the weight of the computational aspects model. In this way, it is easier to obtain a partial model of the Tweetchain protocol. At the best of our knowledge, indeed, Tamarin represents a useful modelling framework for protocols and not for code: computational aspects are tough to model in Tamarin, and hence some alternatives should be explored and, in case, integrated.

A notable alternative is constituted by the Stateful Applied Pi Calculus (SAPiC) formalism which promises to improve the modelling power of the Tamarin Prover with the possibility of dealing with global protocol state. As multiset rewriting is a “low-level” specification language with no direct support for concurrent message passing, encoding protocols correctly becomes tricky. The difficulties found in this paper are common to a part of the scientific community. SAPiC proposes a process calculus which is a variant of the applied pi-calculus with constructs for manipulation of a global state by processes run-

ning in parallel. The language can be translated into Tamarin Prover, preserving all security properties expressible in a dedicated first-order logic (Li et al., 2019). Expressing the Tweetchain model according to the SAPiC notation constitutes one of the future works of this paper.

### 4.2 Analysis

A first consideration is that the analysis phase strongly depends on the model and that a flawed model always affects its capability to be analysable. This is a fact for many formal languages, including Tamarin where the tuning activity of a model is a non-trivial task.

A great support is given by the interactive mode, which is an excellent way to analyse (and debug) a model and to shorten the model tuning task. It is far from the intentions of Tamarin development group, in fact, to create a “push-button” technology: using the interactive mode is the only way to know if there is some partial deconstruction left. Moreover, it produces graphs of the proofs of the lemmas, allowing the users to explore the proof state space and to inspect the attack graphs. In this way, the modeller has a previous feedback on the model enabling a faster tuning phase.

In particular, for what concerns the partial deconstruction, sometimes it is difficult to find the *perfect* source lemma. In fact, it is necessary to solve partial deconstruction, but it has to be proved true, as it helps the tool in the precomputation phase and thus all the proofs of the model rely on the sources lemmas. The graphical user interface helps a lot in this task.

Furthermore, the authors have used the version 1.6.0 of Tamarin, that introduced the mechanisms of automatic generation of source lemmas. This mechanism, as well as other restriction-related elements, can dramatically improve the efficiency of the analysis process; instead, a less-than-perfect choice of lemmas and restrictions can easily lead to a non-terminating analysis process. As the Tamarin developers highlighted in (Cortier et al., 2020), this feature only worked with many simple protocols, and thus it was not useful for our large Tweetchain protocol.

### 4.3 Process

Formal modelling is an error-prone task, especially when handling with complex protocols involving one protocol relying on the security features exposed by another protocol or security mechanism. As the complexity and the pervasiveness of networked systems grow, it is more and more uncommon to see “inde-

pendent” security solutions. The Tweetchain confirms this trend by relying, as an example, on the security of Twitter, as the description of the protocol prescribes (see Section 2.1<sup>10</sup>). As a consequence, there is a need for a compositional method at both modelling and analysis levels. By adopting the first level, methods and tools related to the reuse of existing models (e.g., a TLS model) into the wider model using this protocol, fosters the generation of larger models. By adopting the second level of composition, the reuse is at the analysis level. At the best of our knowledge, there are just a few attempts to support Tamarin composition at model-level (Blot et al., 2017).

Another family of mechanisms is related to automatic generation of Tamarin models from high-level formalisms (e.g., from UML Sequence Diagrams). Even if there are plenty of approaches able to generate formal models automatically, there are no scientific works specifically focusing on Tamarin. A scientific paper proposes the generation of Tamarin model from the Alice & Bob notation (Basin et al., 2015): this is undoubtedly a starting point for future research tasks. It is necessary to point out that a full automation of the modelling task can be achieved only in case of existing compositional approaches.

It seems natural that, even if this paper must limit this discussion merely on the Tweetchain protocol, many of the considerations here reported can be extended to other DLT approaches.

## 5 RELATED WORKS

In the scientific literature, many papers formalise and analyse blockchain mechanisms. These works are framed into the general research directions that the scientific community has drawn on the topic of the blockchain (Lu, 2019). It is worth to underline that many of the works dealing with formal verification focus on smart contracts; in particular, they apply both traditional and specific methods (e.g., the application of functional programming techniques as in (Bhargavan et al., 2016)) to verify Solidity programs or to define subsets of general purpose programming languages for securely writing smart contracts in a secure manner. Two surveys of these works are in (Almakhour et al., 2020; Liu and Liu, 2019a).

Among all the formal verification techniques, model checking has received a lot of interest due to the possible goal of a “push-button” technology able

<sup>10</sup>The reader should note that also the Twitter authentication mechanisms relies on external protocols as HyperText Transfer Protocol over Secure Socket Layer (HTTPS), Transport Layer Security (TLS), OAuth.

to verify security properties without the human interaction. Belonging to this category, there are the following works: (Liu and Liu, 2019b), which models smart contracts using Colored Petri Nets (CPN); (Osterland and Rose, 2020), which translates Solidity in Promela to verify them by SPIN; (Lahbib et al., 2020), where the Event-B framework is used for both modelling and analysing smart contract using both model checking and simulation; (Nehai et al., 2018), that models smart contracts with NuSMV.

Another recent and interesting survey starts with a quantitative analysis of the research works in this field (Singh et al., 2020). The two most interesting analysis dimensions are related to the addressed security concern aspects and to the used formal method. Fig. 3 reports the figure from the cited paper highlighting that less than one paper over five addresses security concerns: please note that, in the pie chart, both the absolute number of the works falling in a category and the percentage with respect to the overall considered works are provided (comma-separated).

Among the papers related to theorem proving that aim at tackling the problem of security assurance of blockchain protocols, we mention the work (Sun and Yu, 2020) where the famous Coq theorem prover has been successfully adopted.

At the best of our knowledge, none of these works explicitly addresses cryptographic issues even if, the work published in (Li et al., 2019) models and analyses a smart contract protocol using Tamarin Prover. Furthermore, authors of (Li et al., 2019) still focus on the internal smart contract mechanisms while the submitted paper is on the exchange protocol. Another missing aspect of the blockchains this work attempts to model and analyse is related to the verification mechanism of transaction chains.

## 6 CONCLUSIONS

This paper has presented a concrete experience in formal validation of a blockchain based protocol in Tamarin. This experience confirmed the value of the Tamarin Prover as a valid tool for the modelling and the analysis of security mechanisms; but, differently from other case studies, the Tweetchain protocol puts in evidence some limitations of the tool.

Section 4 reports a discussion on the limit of the approach proposed in this paper and has also highlighted which are the points where to focus future research effort. Among these, the first points where to focus research attention are: the definition of an automated methodology deriving formal models from high-level formalisms (e.g., UML, Domain Specific

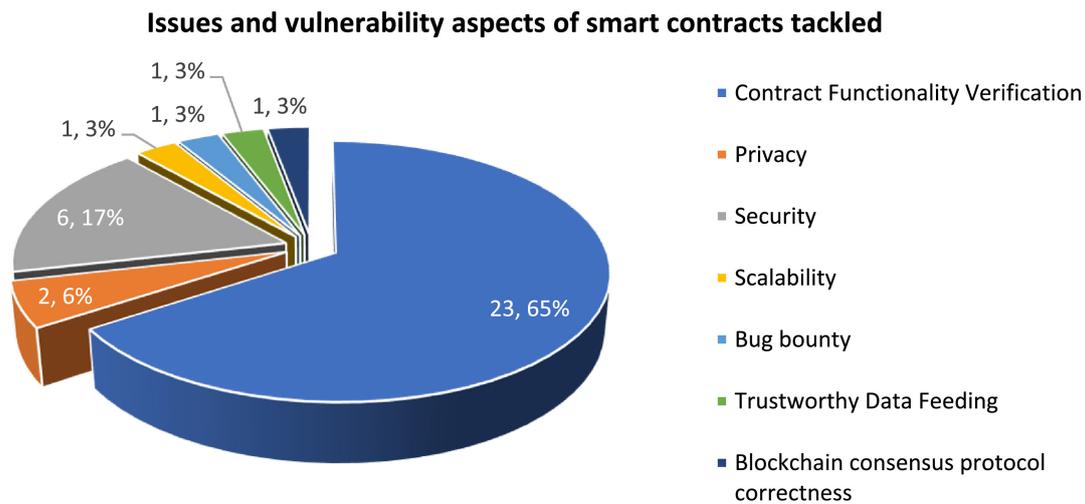


Figure 3: The issues and vulnerability aspects of smart contracts tackled by formalization approaches (Singh et al., 2020).

Modeling Language (DSML)) by using Model Driven Engineering (MDE) approaches; the usage of different model checkers that support array type naturally and where it would be simpler to model chain of hash codes (e.g. Promela and SPIN (Ben-Ari, 2008), NuSMV (Cimatti et al., 2002), UPPAAL (Behrmann et al., 2004) — where also timing issues could be considered).

## ACKNOWLEDGEMENTS

The work of Mariapia Raimondo was formerly funded by the grant “Orio Carlini” for young researchers 2019 — GARR Consortium (Italy). Currently, she is granted by INPS — Istituto Nazionale di Previdenza Sociale (Italy) — with the PhD program — XXXVI cycle.

## REFERENCES

Almakhour, M., Sliman, L., Samhat, A., and Mellouk, A. (2020). Verification of smart contracts: A survey. *Pervasive and Mobile Computing*, 67.

Basin, D., Keller, M., Radomirović, S., and Sasse, R. (2015). Alice and bob meet equational theories. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9200:160–180.

Basin, D., Radomirovic, S., Dreier, J., Sasse, R., Hirschi, L., and Stettler, V. (2018). A formal analysis of 5g authentication. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1383–1396.

Behrmann, G., David, A., and Larsen, K. (2004). A tutorial on uppaal. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3185:200–236.

Ben-Ari, M. (2008). *Principles of the SPIN model checker*. Springer-Verlag London.

Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., and Zanella-Béguelin, S. (2016). Formal verification of smart contracts: Short paper. In *PLAS 2016: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, page 91–96, New York, NY, USA. Association for Computing Machinery.

Blot, E., Dreier, J., and Lafourcade, P. (2017). Formal Analysis of Combinations of Secure Protocols. In *10th International Symposium on Foundations & Practice of Security*, page 15, Nancy, France.

Buccafurri, F., Lax, G., Nicolazzo, S., and Nocera, A. (2017a). Overcoming limits of blockchain for iot applications. In *ACM International Conference Proceeding Series*, volume Part F130521.

Buccafurri, F., Lax, G., Nicolazzo, S., and Nocera, A. (2017b). Tweetchain: An alternative to blockchain for crowd-based applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10360 LNCS:386–393.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2404:359–364.

Cortier, V., Delaune, S., and Dreier, J. (2020). Automatic generation of sources lemmas in tamarin: Towards automatic proofs of security protocols. *Lecture Notes in Computer Science (including subseries Lecture Notes*

- in *Artificial Intelligence and Lecture Notes in Bioinformatics*, 12309 LNCS:3–22.
- Dai, F., Shi, Y., Meng, N., Wei, L., and Ye, Z. (2017). From bitcoin to cybersecurity: A comparative study of blockchain application and security issues. In *2017 4th International Conference on Systems and Informatics, ICSAI 2017*, volume 2018-January, pages 975–979.
- Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.
- Kumar, N. and Mallick, P. (2018). Blockchain technology for security issues and challenges in iot. In *Procedia Computer Science*, volume 132, pages 1815–1823.
- Lahbib, A., Ait Wakrime, A., Laouiti, A., Toumi, K., and Martin, S. (2020). An event-b based approach for formal modelling and verification of smart contracts. *Advances in Intelligent Systems and Computing*, 1151 AISC:1303–1318.
- Lee, M. and Jang, D. (2020). A survey of blockchain security issues. *JP Journal of Heat and Mass Transfer*, 2020(Special Issue 1):29–35.
- Li, X., Su, C., Xiong, Y., Huang, W., and Wang, W. (2019). Formal verification of bnb smart contract. In *Proceedings - 5th International Conference on Big Data Computing and Communications, BIGCOM 2019*, pages 74–78.
- Lin, I.-C. and Liao, T.-C. (2017). A survey of blockchain security issues and challenges. *International Journal of Network Security*, 19(5):653–659.
- Liu, J. and Liu, Z. (2019a). A survey on security verification of blockchain smart contracts. *IEEE Access*, 7:77894–77904.
- Liu, Z. and Liu, J. (2019b). Formal verification of blockchain smart contract based on colored petri net models. In *Proceedings - International Computer Software and Applications Conference*, volume 2, pages 555–560.
- Lu, Y. (2019). The blockchain: State-of-the-art and research challenges. *Journal of Industrial Information Integration*, 15:80–90.
- Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The TAMARIN prover for the symbolic analysis of security protocols. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8044 LNCS:696–701.
- Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system.
- Nehai, Z., Piriou, P.-Y., and Daumas, F. (2018). Model-checking of smart contracts. In *Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, iThings/GreenCom/CP-SCom/SmartData/Blockchain/CIT 2018*, pages 980–987.
- Osterland, T. and Rose, T. (2020). Model checking smart contracts for ethereum. *Pervasive and Mobile Computing*, 63.
- Panwar, A. and Bhatnagar, V. (2020). Distributed ledger technology (dlt): The beginning of a technological revolution for blockchain. In *2nd International Conference on Data, Engineering and Applications (IDEA)*, pages 1–5.
- Singh, A., Parizi, R., Zhang, Q., Choo, K.-K., and Dehghan-tanha, A. (2020). Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. *Computers and Security*, 88.
- Sun, T. and Yu, W. (2020). A formal verification framework for security issues of blockchain smart contracts. *Electronics (Switzerland)*, 9(2).
- Whitefield, J., Chen, L., Kargl, F., Paverd, A., Schneider, S., Treharne, H., and Wesemeyer, S. (2017). Formal analysis of V2X revocation protocols. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10547 LNCS:147–163.
- Whitefield, J., Chen, L., Sasse, R., Schneider, S., Treharne, H., and Wesemeyer, S. (2019). A symbolic analysis of ECC-based direct anonymous attestation. In *Proceedings - 4th IEEE European Symposium on Security and Privacy, EURO S and P 2019*, pages 127–141.
- Yu, Y., Li, Y., Tian, J., and Liu, J. (2018). Blockchain-based solutions to security and privacy issues in the internet of things. *IEEE Wireless Communications*, 25(6):12–18.