Journal of Computer Virology and Hacking Techniques (2023) 19:17–32
https://doi.org/10.1007/s11416-022-00444-z

INVITED PAPER

An approach for the automatic verification of blockchain protocols: the Tweetchain case study

Mariapia Raimondo[1] · Simona Bernardi[2] · Stefano Marrone[1] · José Merseguer[2]

Received: 14 November 2021 / Accepted: 4 June 2022 / Published online: 25 August 2022

Abstract
This paper proposes a model-driven approach for the security modelling and analysis of blockchain based protocols. The modelling is built upon the definition of a UML profile, which is able to capture transaction-oriented information. The analysis is based on existing formal analysis tools. In particular, the paper considers the Tweetchain protocol, a recent proposal that leverages online social networks, i.e., Twitter, for extending blockchain to domains with small-value transactions, such as IoT. A specialized textual notation is added to the UML profile to capture features of this protocol. Furthermore, a model transformation is defined to generate a Tamarin model, from the UML models, via an intermediate well-known notation, i.e., the Alice&Bob notation. Finally, Tamarin Prover is used to verify the model of the protocol against some security properties. This work extends a previous one, where the Tamarin formal models were generated by hand. A comparison on the analysis results, both under the functional and non-functional aspects, is reported here too.

Keywords Distributed ledger technology · Formal modelling, Automatic model generation · Vulnerability discovery · Formal verification · UML profile

1 Introduction

Blockchain is a software layer that provides the basis for verification, validation, recording, and integrity of digital assets transfers, e.g., digital currencies [1]. Blockchain security is therefore a must, which is why it has attracted researchers since its inception. In fact, blockchain technologies are touted as being extremely secure due to the tamper resistant design [2]. However, as also explained in [2], blockchain applications are not immune to malicious actors, who can exploit vulnerabilities and attack them just like websites or applications are attacked today.

Our interest is on the verification of blockchain security properties using a mathematical standpoint. This topic has been successfully addressed in the literature. In particular, model checking, theorem proving or simulation are techniques that have offered good results, as we explore in Sect. 2. Among them, the Tamarin Prover [3] has been used in different works [4,5] to successfully model and analyze security protocols. However, we are concerned with the development of blockchain applications, and there is still a huge gap between the software development process and the formal verification of the blockchain security properties. This work aims to bridge this gap by offering an approach to reconcile both fields: software design and formal verification of blockchain security properties, i.e., to integrate the latter in the development field. Consider that the development of blockchain applications is a growing huge market that, among many others, includes wallets, smart contracts and decentralized applications. Moreover, it may affect almost any industrial sector, among them financial or logistics.

Mariapia Raimondo, Simona Bernardi, Stefano Marrone and José Merseguer were contributed equally to the realization of the paper.

✉ Stefano Marrone
stefano.marrone@unicampania.it

Mariapia Raimondo
mariapia.raimondo@unicampania.it

Simona Bernardi
simonab@unizar.es

José Merseguer
jmerse@unizar.es

[1] Dipartimento di Matematica e Fisica, Università della Campania "Luigi Vanvitelli", viale Lincoln, 5, 81100 Caserta, Italy

[2] Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, C. Maria de Luna, 1, 50018 Zaragoza, Spain

The paper contributes to the state of the art as follows. First, we present an approach for the automatic verification of blockchain protocols. The main idea is to fit the approach within the software development process. Second, the approach proposes guidelines for creating a UML profile, that helps in the modelling and analysis of the protocols, developing a part of this profile. Third, the paper proposes an automatic transformation of UML-profiled models into AnB [6], a formal language based on Alice and Bob notation. Fourth, the approach is applied to a case study: the Tweetchain protocol [7]. The modelling and analysis of the protocol have confirmed the feasibility of the approach.

The analysis of the Tweetchain confirms the validity of the generated model since the analysis of the considered lemmas gives the same results in both, generated and hand-made Tamarin model. Furthermore, a quantitative analysis drives the definition of a first optimization in the translation, which produces a version with smaller execution times and memory occupation.

The Tweetchain protocol has been chosen as a case study because, although being a *lightweight* blockchain protocol, it contains characteristics needed to perform a security analysis, such as coordination, sharing or irrevocability of transactions. Tweetchain was introduced to apply blockchain technology in domains where some of its features, such as the *mining* power, are little suitable. For example, the IoT domain.

This paper extends the work in [8] as follows. The approach for automatic verification is entirely new. Concretely, regarding[8], the approach: (1) introduces a new UML profile; (2) applies the profile to model Tweetchain; (3) models Tweetchain using the AnB language; and (4) gets a new Tamarin model, which has been validated with the hand-made model from [8] and it has been used to verify new properties.

The rest of the paper is structured as follows. Section 2 revises related works. Section 3 presents our approach for an automatic verification of blockchain protocols in the development process. Section 4 accomplishes the modelling of the Tweetchain. Section 5 accomplishes the analysis of the Tweetchain and reports a first optimization experience. Section 6 concludes the paper.

2 Related work

In this section, we revise the contributions in the literature concerning the main aspects of our proposals, that is: (a) formal modelling and analysis of blockchain-based systems, (b) blockchain standards and proposals of reference models, and (c) UML profiling approaches supporting system security analysis.

Blockchain formal modelling and analysis

In the past few years, blockchain has been one of the major focus for security research, resulting in a large number of contributions in the formalization and analysis of blockchain-based systems. The survey [9] discusses 35 papers from 2015 to 2019 just focusing on formalization of smart-contracts. Results from the survey indicate that theorem proving is the most common technique used with the purpose of verifying security properties. Like our proposal, the following revised works rely upon already existing formalisms and techniques. They target either generic systems [4,10,11] or specific protocols [12–14].

Similar to our approach, Duan et al. [10] start from a high-level specification language. In particular, they use the Specification and Description Language (SDL) to define a generic model of a blockchain system. However, the main goal in [10] is different, they aim at verifying the correctness of the specification using simulation and model-checking engines of the Telelogic Tau tool. They use simulation to check whether all the modelled functionalities are operational, whereas model-checking is used to verify the classic logical properties of state-based systems, such as, absence of deadlocks or livelocks, boundedness or reachability.

Boyd et al. [4] define a formal model of blockchain in Tamarin Prover [3] to support the security analysis of cross chain trading protocols based on hash time lock contract. They enhance the blockchain modelling capability of Tamarin by defining domain specific rules to add a transaction to a block, global time rules to specify the time instant of a block being added to the blockchain, as well as HTLC rules for the contract initiator and the responder.

Egger et al. [11] present a framework for defining and analyse the security of distributed ledgers. Thus, a general functionality is defined that aims to cover both, classic blockchains and non-blockchain distributed ledgers, in a unified way. The work considers protocol composability and the framework [15] supports modular analysis of different types of protocols under various security settings.

In the case of specific protocols, their formalization is mainly aimed at enabling analysis for security assessment. Maung et al. [12], for example, formally specify the Tendermint proof-of-stake consensus protocol with CSP# [16] language and analyse it with the Process Analysis Toolkit (PAT) [17] LTL model-checker. CSP# is an extension of CSP (Communicating Sequential Process) with embedding of data operations. The analysis is carried out on the model under normal conditions, as well as against specific attacks in Byzantine environment, e.g., censorship attacks.

Modesti et al. [14] formally model and analyse the payment protocol standard BIP70, which is built on top of the blockchain Bitcoin protocol and specifies how payment in bitcoin is performed by merchants and customers. The authors use AnB [6], a formal language based on Alice and

Bob notation that we also consider in our approach, to formalize the protocol and perform the analysis with the symbolic model-checker OFMG [18].

Tolmach et al. [13] focus instead on decentralized finance (DeFi) protocols, which are one of the most prominent applications of blockchain and smart contracts. Similarly to [12], they use the CPS# modelling language for the specification of the DeFi protocols and the PAT model-checker for the analysis. They propose a compositional approach for the formal analysis of two concrete DeFi protocols, namely Curve and Compound.

Blockchain standards and reference models

Standardization of blockchain distributed ledger technologies (DLTs) is an important step towards a common concept, interoperability and possible regulation since the software industry starts to suffer from the excessive fragmentation of the DLTs market. The survey [19] provides an overview on existing standardization efforts, where international standardization organizations include the National Institute of Standards and Technologies (NIST), ANSI Accredited Standards Committee X9, International Organization for Standardization (ISO), the European Union Agency for Cybersecurity (ENISA), International Telecommunication Union (ITU), and European Committee for Electrotechnical Standardization (CENELEC). Besides, the Object Management Group launched a Request For Information [20] to consider interoperability issues across blockchain and DLTs, however no public documents are currently available.

Concerning the definition of reference conceptual models, we mention two works [21,22] that contribute to define domain models for the blockchain ecosystem and are closely related to our approach. Ellervee et al. [21] propose a comprehensive model of the blockchain and DLT. In particular, their data model represents the basic concepts of transactions and blocks. In our proposal, we rely upon the NIST standard [2], then we refine the transaction concept with the input and output assets that were not considered in [21]. Skotnica and Pergl [22] define a domain-specific language for specifying smart-contracts to support automated code generation. Such language is defined as a meta-model using UML class diagrams. Also our proposal defines a domain-specific language, but with a different goal: the security analysis of blockchain protocols. Moreover, we apply UML profiling techniques.

UML profiling for security analysis

There is no UML standard profile—i.e., a UML profile defined by the Object Management Group—devoted to the security analysis of blockchain-based protocols and applications. However, the QoS&FT profile [23] provides general support for the specification of Quality of Service (QoS) characteristics and for risk assessment. Moreover, there are contributions that propose UML profiling as an approach for the definition of domain specific languages for the modelling and analysis of security properties of software systems, such as SecureUML [24], UMLSec [25] and SecAM [26]. Nevertheless, none of the profiles mentioned address the domain of blockchain-based systems.

SecureUML [24] focuses on the access-control mechanisms and automatic generation of access control infrastructures. Automated analysis of SecureUML models is also proposed [27], where the security properties to be verified are specified as formula with the Object Constraint Language (OCL).

UMLSec [25] provides instead a support for the specification of security requirements as logical constraints (confidentiality and integrity properties) and the requirements assessment via formal analysis. For example, in [28] a tool-support is proposed where UMLSec models are verified against the specified requirements. The automatic checking is carried out using a theorem-prover for first-order logic (e-SETHEO).

SecAM [26] enables the security specification and modelling of critical infrastructures. In particular, it consists of several packages, each one defining extensions for a specific aspect of security (resilience, cryptographic, security mechanisms and access-control): SeCAM provides support for the survivability analysis of critical infrastructure using a model-driven approach.

3 Approach for automatic verification

Our idea is to automatically verify security properties of blockchain protocols using a mathematical standpoint. However, we propose to model the very same protocol using a software design standpoint, which is important to facilitate the integration of verification activities within the development cycle. At this regard, the model-driven engineering (MDE) paradigm [29] offers solutions, for bridging different technologies, through the concept of *technical space* [30]. A technical space is a working context, with a set of concepts, a body of knowledge, tools, required skills, and possibilities [31]. Hence, as proposed by Bézivin et al. [32] we propose to bridge the gap between our technical spaces, i.e., the software design and the mathematical verification, using *model transformations*.

According to previous MDE premise, our statement is that *starting from software models, representing (part of) the behaviour of the protocol and the security properties to verify, mathematical formal models can be automatically generated. Then, such models can be used for the formal analysis of the security properties using existing tools*.

3.1 Overview of the approach

A practical realization of the previous general statement is presented in Fig. 1, which conforms to our proposal for

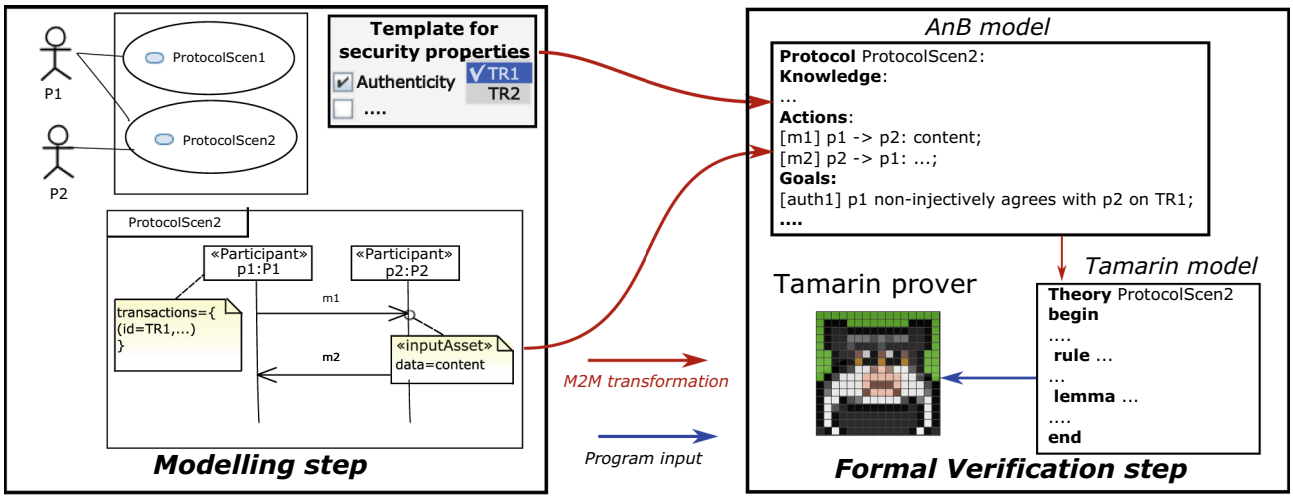

Fig. 1 Approach overview

automatic verification. The figure highlights the state of the art languages and tools, in the model-driven context, that we have selected. Initially, the engineer models a particular blockchain protocol feature using UML (Modelling step in the figure). As a second step, the UML models together with the security properties to be verified will be automatically transformed to Alice and Bob (AnB) notation (the two red arrows from Template for security properties and UML models to the AnB model, in the figure). In a third step, the AnB models are automatically transformed to the Tamarin language (the red arrow from the AnB model to the Tamarin model). Finally, the Tamarin Prover [3] will execute the Tamarin model to verify the desired properties of the protocol (the blue arrow).

Before exploring each step of the approach, we need to discuss some practical and implicit assumptions that underlie the approach. Assuming that UML is closer than Tamarin to the engineer interests, for example the very same models can be reused for code generation, a question arises: Why not transform the UML models directly in the Tamarin language? In other words, what is the practical need for having AnB as intermediate notation? Certainly, this would also be a plausible solution, that we do not belittle. However, Basin et al. [5] have already proposed a transformation of AnB models into Tamarin models, moreover they have implemented such transformation, which in practice means to have an automatic transformation at hand.

On the other side, we can find tools (e.g., [33,34]) that convert UML sequence diagrams to Alice and Bob notation, and vice versa, although such transformations are purely syntactical. Consequently, it is clear that currently, we are closer to find an automated solution by leveraging the AnB notation than by proposing a direct transformation to Tamarin. Nevertheless, as a conceptual solution, we see more elegant the direct transformation, but a considerable benefit of the AnB notation is that it can also be transformed to other models for different analyses.

The first step of the approach (Modelling step in the figure) addresses both: a) the modelling of the target blockchain protocol, e.g., Tweetchain, and b) the definition of the protocol security properties, that the engineer needs to verify. For the first, we have chosen UML, since it is usually defined as a suitable tool for designing software models. This is true considering that in a given project, the same UML models can be leveraged by the engineer, in the MDE context, for multiple purposes. For example, UML models are useful for code generation, as previously mentioned, for automatic testing, and also for performance assessment [35] and dependability assessment [36]. However, UML falls short for representing those specific concepts of the blockchain domain that will be eventually needed for proving the security properties. For example, the attributes of a block chain transaction, which will be needed to prove security properties such as transaction authenticity, integrity and no repudiation. A common solution to this problem is to extend the UML semantics with the concepts of the target domain, in this case the blockchain protocols. This process is known as the creation of a "profile" for UML. An alternative, also promoted in the MDE context, is to create a domain specific modelling language (DSML), in this case it would mean to create a "blockchain specific" modelling language, as proposed in [22] for smart-contracts. In this paper, the UML Profile case is adopted to avoid the limited reusability of DSML in different contexts. Section 3.2 offers the main ideas for developing a UML profile suitable for the verification of security of blockchain protocols. Regardless of using a DSML or a UML profile, a benefit is that the same language guides the modelling step. For example, the basic constructs of a UML profile, i.e. stereotypes, indicate the modeller the concepts that need to be addressed

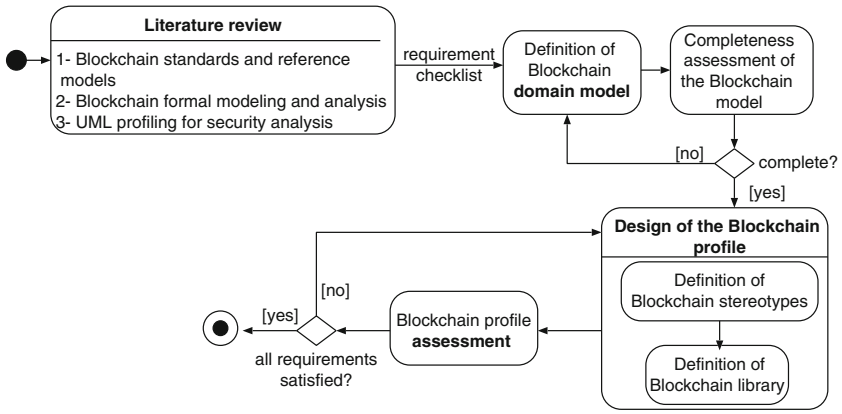

Fig. 2 Approach for UML profile definition, taken from [39]

and how they need to be used. Section 3.2 makes this guidance explicit through the concept of 'blockchain transaction'.

On the other hand, UML also falls short regarding the definition of security properties. We envision two possible strategies. The first one is to provide extensions within the UML profile for the specification of the properties to be verified as well. We think that this is a viable solution in case of "easy to specify" properties, i.e., properties related to single model elements, such as the above-mentioned properties related to a transaction. Such properties do not require a specific syntax, such as OCL. The second strategy is to define the security properties as parametric query templates, as in [37], which allows addressing properties involving various model elements. These queries will be instantiated by binding the model elements to parameters, via a proper GUI. The latter strategy enables to specify finer grained security properties.

In the case study, we have adopted a hybrid approach, by proposing some query templates but letting that their instantiation could be driven by simple specification of transaction id. For sake of simplicity, this specification is not made in the UML model, but delegated to command-line tools.

The second step of the approach addresses a transformation (the two red arrows from Template for security properties and UML models to the AnB model, in the figure). The protocol model in UML and the security properties to be verified are transformed into the AnB model. The latter is also leveraged to model blockchain-specific constructs. In the third step (the red arrow from the AnB model to the Tamarin model), we reuse and customize the transformation proposed in [5]. Hence, the AnB model is automatically transformed into a Tamarin model. Customization is indeed needed, since the original proposal considers general cryptographic protocols, i.e., not blockchain specific. The fourth step (the blue arrow) consists in the formal analysis, with the Tamarin Prover tool [3], of the Tamarin model already obtained. We remark that the tool enables to check the lemmas (e.g., first-order logic formulas over symbols, which express the security properties to be verified), via model state-space exploration, under the Dolev–Yao adversary model [38].

3.2 Towards a UML profile for blockchain protocols

The construction of a UML profile is a well-known process, which does not mean to be easy. The authors have previously proposed different profiles [39,40] following recommendations from Selic [41] and Lagarde et al. [42]. Figure 2 summarizes the main steps for producing a technically correct, high-quality UML profile.

For the *literature review*, we propose to study the main blockchain standards and reference models, the blockchain literature on formal modeling and analysis and the existing UML profiles for security modeling and analysis. An initial work on these subjects has been presented in Sect. 2. The output of this first step will be a *requirements checklist*, which will be useful for the definition of a blockchain domain model.

The *domain model* must represent main blockchain concepts that will be useful for both, the modeling of the protocol and its analysis. Figure 3 depicts, for illustrative purposes, how to model the concept of blockchain **transaction**. For this step we envision the need of creating a core domain model with the core concepts of the blockchain technology, as given for example in the NIST document [2]. Such domain model will be complemented with specific concepts of the target blockchain protocol, in case they exist. For example, specific concepts of the Tweetchain protocol. Finally, an assessment on the completeness of the domain model is proposed, this must ensure that each requirement in the checklist is properly addressed in the domain model.

The next step means the *design of the profile*. We need to map the concepts in the domain models to UML. Since a profile is made of UML extensions (stereotypes, tags, and constraints), we need to adjust each concept in the domain model to a proper UML extension. In general, we advise defining a minimum, yet powerful, set of stereotypes. In the end, each stereotype will be used to annotate concrete model elements in the UML models representing the protocols, see Fig. 5 for different examples. For this to work, we need to select the appropriate UML metaclasses for each stereotype.

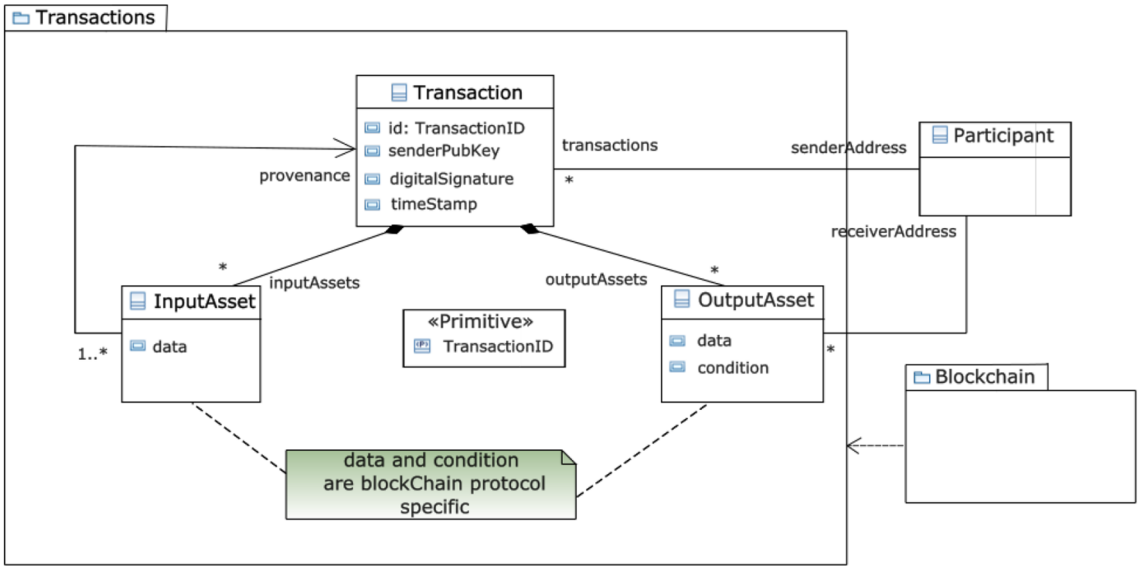

Fig. 3 Domain model of transactions

Table 1 UML extensions of the profile, mapped from the domain model in Fig. 3

Stereotype	Model element	Tag	Description
Participant	Lifeline	Transactions	List of transactions associated to a participant
InputAsset	MessageOccurrenceSpecification	Data	Message content sent (list of string values)
		Provenance	Source transaction of the asset
OutputAsset	MessageOccurrenceSpecification	Data	Message content received (list of string values)
		ReceiverAddress	Asset recipient (a Participant)
		Conditions	Conditions to be met by the recipient
New types	Kind	Tag	Description
Transaction	Datatype	id	Transaction identifier (TransactionID)
		SenderPubKey	Sender public key
		SenderAddress	Sender (a Participant)
		digitalSignature	Sender digital signature
		TimeStamp	Time stamp
		InputAssets	List of input assets
		OutputAssets	List of output assets
TransactionID	Primitive		

Table 1 shows the stereotypes corresponding to the concepts identified in the domain model of Fig. 3, and included in the profile. For this step, we will need to create the UML extensions corresponding to the core domain model and the UML extensions for the specific domain model of the target protocol. A BNF grammar can be an alternative to the protocol-specific UML extensions. The specification language of a particular blockchain protocol will be the sum of both extensions, obviously. Finally, another assessment process is advised to ensure that all the concepts proposed have been correctly mapped to UML extensions.

4 Case study: modelling

This section focuses on applying the first three steps, described in Sect. 3.1, to the Tweetchain case study. The fourth step will be carried out in Sect. 5. For the first step, Sect. 4.1 presents the UML model of the Tweetchain protocol. For the second step, Sect. 4.2 describes a model transformation to yield an AnB model. For the third step, Sect. 4.3 reports a Tweetchain Tamarin model (partially) obtained by applying the transformation in [5].

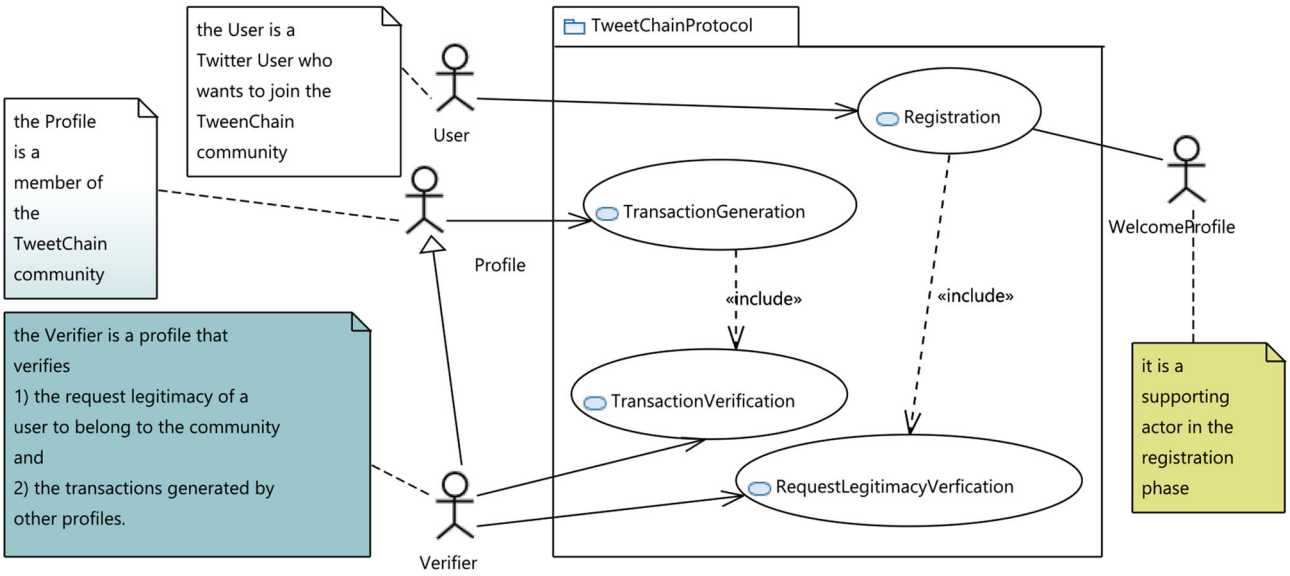

Fig. 4 Use Case diagram of the Tweetchain protocol

4.1 UML modelling

The UML model of the Tweetchain protocol is made of a use case diagram, see Fig. 4, and a sequence diagram, see Fig. 5.

The use case diagram defines the context of the system and it is made of four use cases: *Registration, RequestLegitimacyVerification, TransactionGeneration* and *TransactionVerification*. For our purposes, only the first two use cases are considered. The other two use cases are explained in [43].

The sequence diagram details the interactions, among actors, carried out in the two use cases of interest. Hence, each lifeline represents the involved actors: the User (x), the WelcomeProfile (W) and the Verifier (y). The sequence diagram starts when each party signs in Twitter. Then, a participant x publishes a *hello tweet* message to register itself to W. Such message contains the first elements, HC_X^1 and HC_W^1, of the hash-chains of x and W, respectively. So, W verifies the tweet and sends as a response a *welcome tweet*. This response contains the hash value HC_W^i, computed on the base of its hash-chain, the hash-chain of x (HC_x^1) and the Twitter id (TID_x^1), which is unique in Twitter for each tweet. After that, x chooses, among the members of the community, its set of verifiers. Then, it sends them a private *followership request*. After checking the legitimacy of the request, each verifier y confirms and follows the user x and publishes a *follow_welcome tweet*. Consequently, the sequence diagram ends and the registration of x has been completed, also considering the verification of the legitimacy of its request.

The following aspects are of interest in the sequence diagram:

- The verifiers are chosen on the base of a publicly known algorithm and by a public seed. As a seed, the protocol may use the Twitter user id, that is unique and publicly available, as proposed by [7]. The use of publicly verifiable information, in this phase, is motivated by the necessity of assuring that each party in the community does not choose its set of verifiers in a malicious way.
- Each lifeline is stereotyped as *Participant*. According to Table 1, each participant has **transactions** and each transaction is made of different fields. The **outputAsset** and **inputAsset** fields summarize the sets of events, sent and received, composing the transaction defined by the lifeline. Moreover, the fields **senderPubKey** and **digitalSignature** represent the shared knowledge.
- Each message m is characterized by a pair of send and receive events, (mSE, mRE): such events are the *MessageOccurrenceSpecification* model elements as in Table 1. However, only events stereotyped as `«InputAsset»` (`«OutputAsset»`) belong to security-sensitive messages, which are depicted as thick arrows. By "security-sensitive" message, we mean a message that is relevant for the formal analysis of the protocol, then it will be dealt with in the model transformation. These assets have a **data** tag, whose type, according to Table 1, is a list of strings. This list represents the content of the exchanged message, and its semantics is protocol-specific. Therefore, it needs to be provided by a protocol-specific UML profile or, alternatively, by a BNF grammar. For the case study, we have developed the BNF grammar described in the following.

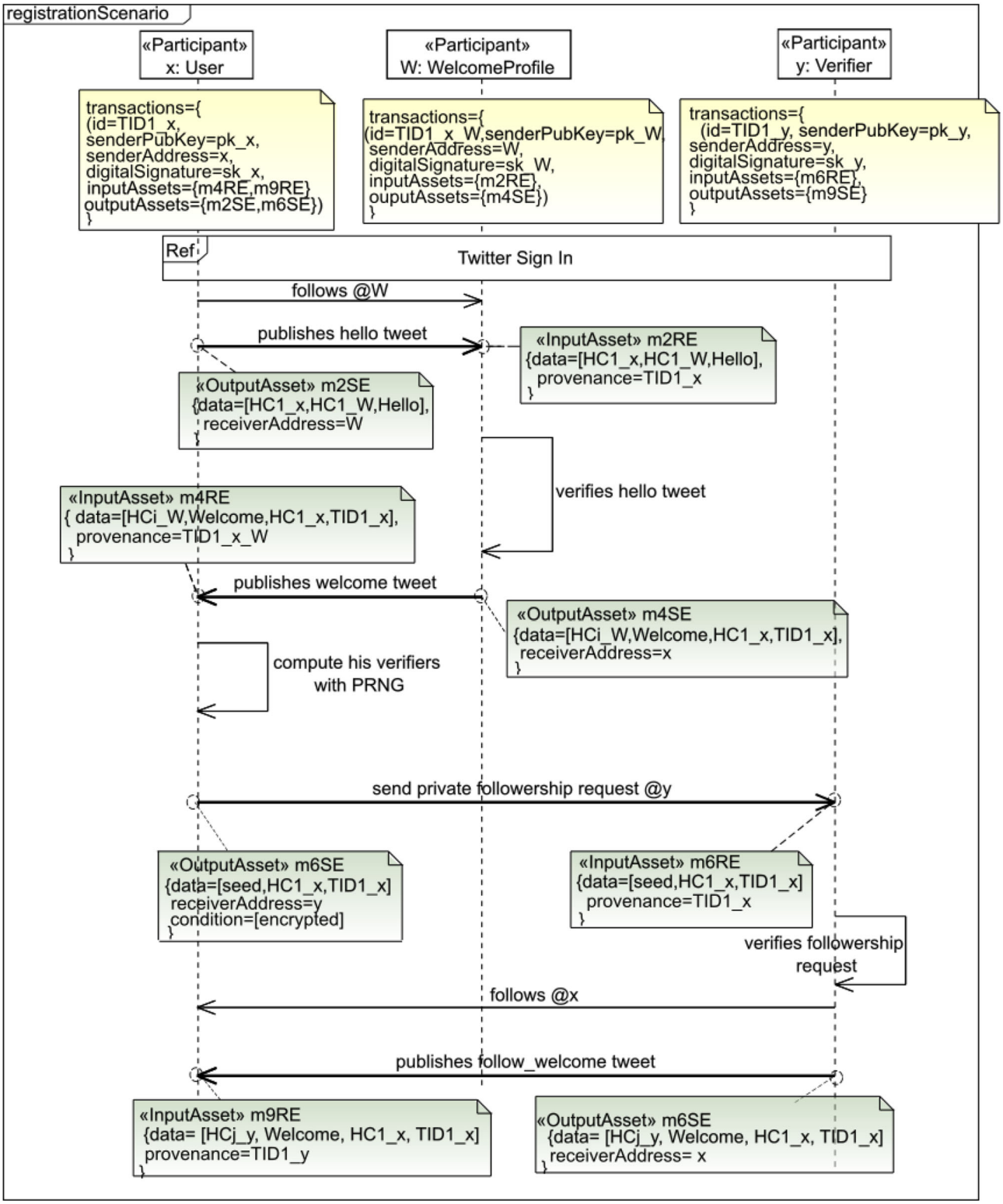

Fig. 5 Sequence diagram detailing the Registration use case

Table 2 Data specification notation for the Tweetchain protocol

```
1  <data >            ::=  <openmessage> <message> <closemessage >
2  <message>          ::=  { hello_msg }  <hello > |
3                          { welcome_msg }  <welcome>
4  <hello >           ::=  <hash_x > <sep > <hash_w> <sep > <hello_tok >
5  <welcome>          ::=  <hash_x > <sep > <welcome_tok> <sep > <hash_x > <sep > <tid_x >
6  <hash_x >          ::=  <hash >
7  <hash_w>           ::=  <hash >
8  <tid_x >           ::=  <tid >
9  <hash >            ::=  <hashsymbol > <hashstring >
10 <tid >             ::=  <digit > [ <digit >]
11 ...
12 <openMessage>      ::=  "]"
13 <closeMessage >    ::=  "["
14 <sep >             ::=  ","
15 <hello_tok >       ::=  "Hello "
16 <welcome_tok>      ::=  "Welcome"
17 <digit >           ::=  "1"  | ..  | "9"
18 ...
```

Protocol-specific textual notation

Table 2 reports part of the BNF grammar. The grammar has been implemented by means of the SableCC compiler generator tool.[1] All the artefacts, described in this work, are publicly available in their full version in a public Git Hub repository.[2]

Among all the messages considered in the Tweetchain protocol, the BNF grammar reports two of them—e.g., `hello` and `welcome`—for the sake of simplicity (see lines 3 and 4). The grammar also describes the format of the keywords (i.e., 'Hello' for the hello message and 'Welcome' for the welcome one), and of the response status of the verification (i.e., '1' for valid transactions, '0' otherwise). It is important to underline that this grammar does not refer to any concrete environment and hence some clarifications are due: curly brackets indicate the labels related the different cases of a production rule (e.g., hello_msg, welcome_msg); on the other hand, square brackets report optional information.

4.2 From UML to AnB

Algorithm 1 proposes a transformation of the UML model presented in previous section into an AnB model. The algorithm receives as inputs the very same UML *model* and the name of the Tweetchain welcome profile wp. Then, it produces the corresponding AnB model, anb. An AnB model is structured into four parts (see Appendix A.2 for details):

- A `Declaration` section—where user-defined functions and helpers are reported. This part is not relevant in this paper;
- A `Knowledge` section—devoted to declare the information that each participant knows about the other ones;
- An `Actions` section—that is related to the messages exchanged by the participants;
- A `Goals` section—which includes the security properties to verify.

Algorithm 1 UML-to-AnB

```
1: procedure UML2ANB(model,wp)
2:     anb ← None
3:     parts ← model.getParticipants()
4:     msgs ← model.getMessages()
5:     for all p ∈ parts do
6:         temp ← knowledge(p, parts, wp)
7:         anb.update(temp)
8:     end for
9:     for all m ∈ msgs do
10:        temp ← action(m)
11:        anb.update(temp)
12:    end for
13:    return anb
14: end procedure
```

Algorithm 1 starts extracting from the *model* the list of participants *parts*, which are the lifelines stereotyped as «`Participant`» in the *model*, and the list of messages *msgs*. The algorithm proceeds applying the `knowledge()` and `action()` procedures to all participants and messages, respectively. These procedures are detailed by Algorithms 2 and 3, respectively.

Algorithm 2 creates the `Knowledge` section of the AnB model for a participant *part*, Listing 1 presents an excerpt of such section produced by the Algorithm 2. The algorithm relies on the functions $rootHash(part)$, $hash(z)$ and $pk(other)$, as follows:

- $rootHash(part)$ generates the first hash of *part* (knowledge portions `root_HC_part` for the participant `part` in the Listing 1);

[1] https://sablecc.org/.

[2] *Security4Blockchain* repository URL: https://github.com/stefanomarrone/security4blockchain.

Algorithm 2 Knowledge generation

```
1: procedure KNOWLEDGE(part,list,wp)
2:     retval ← part
3:     retval.append(rootHash(part))
4:     retval.append(hash(rootHash(wp)))
5:     for all other ∈ list − {part} do
6:         retval.append(other)
7:         retval.append(pk(other))
8:     end for
9:     return retval
10: end procedure
```

- $hash(z)$ generates the hash value of z (knowledge portions `h(root_HC_part)` for the participant `part`);
- $pk(other)$ generates the public key of $other$ (knowledge portions `pk(other)`).

Listing 1 Example of the `Knowledge` section

```
1  Knowledge:
2  W  : pk(x), pk(y), root_HC_W, h(root_HC_W), x, y;
3  x  : pk(W), pk(y), root_HC_x, h(root_HC_W), W, y;
4  y  : pk(W), pk(x), root_HC_y, h(root_HC_W), W, x;
5  end
```

Algorithm 3 creates the `Actions` section of the AnB model, Listing 2 presents an excerpt of such section. The algorithm only processes those messages stereotyped as «`InputAsset`» or «`OutputAsset`», i.e., secure-sensitive messages, which are translated into AnB rules as indicated by lines 4-10. Function $msg.getST()$ extracts the information contained in these stereotypes. In particular, the **data** tagged-value contains information to decide the rules to generate, $getTV("data")$ extracts such information, which is then parsed by a compiler based on the grammar presented in Sect. 4.1, see $oasset$ function in line 6. The algorithm determines the kind of the message, $msgKind(data)$, and for each kind of message writes the specific $actions$ in the model. Lines 8-10 process messages of the kind $Hello$, then generating specific actions for hello_tweet messages. The other kind of messages are processed in the same way, so the algorithm does not repeat such lines, see line 11.

Listing 2 Example of the `Action` section

```
1  Actions:
2      [Hello_1] x -> W (TID_1_x):
3          x.h(root_HC_x).h(root_HC_W).'Hello'.W.
               TID_1_x.aenc{x.h(root_HC_x).h(
               root_HC_W).'Hello'.W.TID_1_x}sk(x);
4
5      [Hello_out_2] x -> y:
6          x.h(root_HC_x).TID_1_x.aenc{x.h(root_HC_x
               ).TID_1_x}sk(x);
```

Algorithm 3 Action generation

```
1: procedure ACTION(msg)
2:     retval ← []
3:     secureMsg ←
           msg.hasST("InputAsset") &&
           msg.hasST("OutputAsset")
4:     if secureMsg = True then
5:         oasset ← msg.getST("OutputAsset")
6:         data ← oasset.getTV("data")
7:         kind ← msgKind(data)
8:         if kind =' Hello' then
9:             retval ← genHello(msg, data)
10:        end if
11:        ...
12:    end if
13:    return retval
14: end procedure
```

Listing 2 details the actions created in the AnB model for a hello_tweet message:

- The first action (lines 2–3) corresponds to the message sent from x to W. It is built on the information contained in the model (x, W), in $data$ (root_HC_x, root_HC_W), in the message (TID_1_x) and in labels (Hello_1);
- The second action (lines 5-6) corresponds to the message sent by x to all the participants to inform about its TID_1_x, which is publicly available in Twitter. This rule is replicated for each participant in the sequence diagram.

While, in the long run, the generation of the goals to verify will be done from their specification in the UML model (see Sect. 3), this paper sketches transformation rules from UML to Tamarin query templates or by generating AnB goals. The first case is described in Sect. 4.3 while the second is discussed in Sect. 5.2. Both the cases can be implemented by command-line tools.

4.3 The Tamarin model

The security rules mentioned in previous subsection must be added to the Tamarin model automatically produced by the approach in [5]. In particular, we add a rule to consider the theft of the private keys of the participants by an adversary. Moreover, we need to add *lemmas*, expressing the expected properties, and labels to the *rules*. By doing so, we certainly could compare our analysis results with those from [8]. Listing 3 shows an an example of label, highlighted in green.

Listing 3 Hello Tweet rule

```
rule hel_twe_x:
        [ St_init_x(TID_1_x, W, r,
    root_HC_x, x, y, sk(k_x), pk(k_W), pk(
    k_r),
          pk(k_x), pk(k_y), alpha)
        ]
        --[ HELLO_sent(x, W, TID_1_x) ]->
        [ Out(<x, h(root_HC_x), alpha, '
    Hello', W, TID_1_x, aenc{<x, h(
    root_HC_x
          ), alpha, 'Hello', W, TID_1_x>}
    sk(k_x)>),
          St_hel_twe_x(TID_1_x, W, r,
    root_HC_x, x, y, sk(k_x), pk(k_W), pk(
    k_r)
          , pk(k_x), pk(k_y), alpha)
        ]
```

The *lemmas* added to the Tamarin model mean to consider the authenticity of the data transmitted, that can be verified for each exchanged message. In fact, it corresponds to check the following property "If user x receives a message m from user y, then y has sent m to x earlier". In the model, the authenticity of the sender is obtained by signing the messages with his/her private key. Then, 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, two different query templates account for the capability of the adversary to steal the private key of the parties involved in the communication: the `fake_trace` and `trace`. In the former, the adversary can steal the private keys, whereas in the latter, he/she is not. Listing 4 reports the query template of the `trace` property, the text between two semicolons needs to be instantiated.

Listing 4 Template of `trace` lemma

```
lemma ;lemma_name;:
  " All ;sndr; ;rcvr; ;tid; #i.
      (;msg_lbl;(;sndr; , ;rcvr; , ;tid;)
    @i
       & not (Ex #r. RevLtk(;rcvr;) @r)
       & not (Ex #r. RevLtk(;sndr;) @r))
    ==>
    Ex #j.
      ;prev_lbl;(;rcvr; , ;sndr; , ;tid;)
    @j & j<i "
```

For instantiating the lemma in Listing 4 let us consider the specification of a transaction (tid) in the UML model. By navigating the UML model, it is possible to extract the messages related to tid, then the variables to be instantiated in the lemma can be computed from them. Given a tid, a list of messages is considered and for each message the following information can be obtained:

a *;lemma_name;* is obtained by concatenating: `trace` (name of the template), tid and the position of the message in the transaction's list;

b *;sndr;* and *;rcvr;* are the names of the sender and receiver participants, respectively;

c *;msg_lbl;* and *;prev_lbl;* are the labels of the messages as they have been added in the Tamarin rules.

d *RevLtk* is a label added after generating the AnB.

It is worth noticing that the model transformation described above is in its design stage, and the implementation of such a tool is an ongoing work.

5 Case study: validation and verification

This section has two main goals. First, the validation of the Tamarin model proposed in this paper. Second, the verification of the transaction authenticity in the registration scenario of the case study (cfr. Fig. 5).

All the analyses have been carried out by running the Tamarin Prover tool version 1.6.0 in batch mode using the following HW/SW configuration: a Linux Ubuntu Server 20.04.1 LTS running on a quad-code Intel(R) Xeon(R) CPU E5-2650L v4 1.70GHz, with 8 Gb of RAM.

5.1 Model validation

The validation of the Tamarin model, generated using the proposed two-staged transformation, is carried out by comparing it with the hand-made model in [8]. The comparison is twofold. First, we assess the correctness of the model by checking whether the Tamarin Prover produces the same results as the ones obtained for the hand-made model. Second, we assess performance by comparing the execution times needed to analyze the two models.

Correctness assessment

Let us consider the Tamarin model describing the registration scenario of the Tweetchain protocol and the security goals. All the considered security properties hold, assuring that the adversary did not compromise the asymmetric cryptographic system. The types of security goals addressed in this analysis concern the authenticity of each exchanged message in the protocol. In particular, for each message two different lemmas have been defined that account of the capability of the adversary to steal the private key of the participants. Such lemmas are labelled 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. Table 3 reports the analysis results of the considered lemmas: `true` indicates that the property holds, whilst `false` indicates that the tool has returned a counterexample representing an attack scenario that violates the desired property.

Table 3 Authenticity of messages in the Registration scenario (Fig. 5): results of lemmas proof

Message	Lemma type	Result
Publishes hello tweet	`Fake_trace`	`False`
	`Trace`	`True`
Publishes welcome tweet	`Fake_trace`	`False`
	`Trace`	`True`
Send private followership request	`Fake_trace`	`False`
	`Trace`	`True`
Publishes follow_welcome tweet	`Fake_trace`	`False`
	`Trace`	`True`

The results of the lemmas' verification correspond to the ones obtained with the analysis of the hand-made model in [8].

Performance assessment

The objective is to compare the execution times of both models. Hence, the two models (named, respectively, *generated* and *manual*), are solved on the machine described above. Furthermore, regarding the *generated* model, we performed some simplifications at the AnB level: this simplifications are derived from the assumption that some parties (e.g., y) already belong to the Tweetchain community and hence their *TID* is already known. Practically, this makes a change in the AnB model `Knowledge` section as in the following:

$$y : pk(W), pk(x), \textbf{TID\_y}, root\_HC\_y, W, x;$$

With respect to Listing1, the previous line changes **root_HC_y** into **TID_y** inhibiting Tamarin Prover to regenerate such an information in its analysis. This version can be generated by modifying Algorithm 2. Another change is related to the `Action` section, which can be managed by a small change in the Algorithm 3. The full discussion of optimization opportunities is out of the scope of this paper and will be considered in future research. In the rest of this paper, this model is named as `generated (simpl.)` since it will be used in Sect. 5.2.

The executions of Tamarin Prover have been instrumented to capture both CPU times and the peak memory occupation. The executions refer to the verification of all the security properties considered in the previous analysis. All the measures have been computed by using the Linux program `/usr/bin/time` with the `-v` option. Table 4 reports such numeric results: these values represent the mean and the standard deviation computed on 30 repetitions of the analysis.

From the performance results, we can conclude that the manual model performs better than the generated one. This is due to the reuse of existing tool chains (i.e., the Keller's transformation) while hand-made code allows implicit optimization the modeller introduces. Moreover, the first attempt of model optimization/simplification strongly improves performance: even if the Keller's tool has been used for this as an opaque box and the optimization opportunities are at the first steps, the analysis is boosted meaningfully.

Table 4 Comparison of the Tamarin model executions

	CPU time (s)	Peak memory (MBs)
Generated	1067.68 ± 10.01	6322.32 ± 70.13
Generated (simpl.)	15.68 ± 0.37	103.53 ± 7.16
Manual	264.36 ± 1.66	2017.99 ± 34.76

A final consideration concerns the modelling effort. Hand-made models are hard to build by a non-expert and they are long to debug and tune. We estimate that, the first working version of the hand-made model has taken approximately three weeks to be done, and almost two weeks to be tuned. The model-driven process applied to the case study has drastically shortened the time to build the formal model reducing it to few hours.

5.2 Verification of transaction authenticity

Further analysis has been carried out in the Tamarin model generated from the AnB, by leveraging the feature of the automatic translation. In particular, it is possible to add in the `Goals` section of the AnB model a statement expressing the *injective agreement* between two participants. As explained in [44], an *injective agreement* goal aims at checking that whenever a participant `A` completes a run of the protocol—apparently with `B`—then `B` has been running the protocol—apparently with `A`—and the two participants agree on the message `m`. This property can be checked from the point of view of both participants.

Regarding the Tweetchain protocol, this kind of property has been checked by stating an *injective agreement* goal, on the transaction identifier, for each ordered pair of participants to the protocol. In fact, the aim was to check that, in the Registration phase of x, each of the involved participants (y and W) run the protocol with x agreeing on the transaction id used by him/her to achieve the registration to the community. Thus, four statements have been added to the `Goals` section of AnB, expressing the agreement between x and W, and x and y, from the points of view of both the participants. The analysis has been carried out assuming that the adversary was not able to steal the private keys of the participants.

The *injective agreement* properties have been checked in the hand-generated model, too. This has been accomplished by adding the necessary labels to the rules and by writing the lemmas in the proper way. The results are reported in Table 5.

Table 5 Transaction authenticity in the Registration scenario (Fig. 5): results of lemmas proof

Participants interaction	Generated	Manual
(x,W)	`True`	`True`
(W.x)	`True`	`True`
(x,y)	`True`	`True`
(y,x)	`False`	`False`

It is possible to notice that the lemma aimed at checking the *injective agreement* between x and y, from the point of view of y, resulted `false` in both the models. This means that from the point of view of y it cannot be guaranteed that the agreement has been done between x and y themselves. Moreover, by analysing the protocol with the Tamarin Prover in the interactive mode, the generated counterexample graph shows the steps of a possible replay attack.

A final remark is related to the automatic generation of the properties to check. At the contrary of the cases described in Sect. 5.1, the generation of the goals is not mediated by Tamarin's lemma templates but are injected directly into the AnB. The reason is the support for this kind of analysis by Keller's tool. Such goals are reported in Listing 5 and are easy to generate from the UML model given the id of the transaction to verify.

Listing 5 AnB goals for injective agreement analysis

```
1 [IA_x_W] x injectively agrees with W
      on TID_1_x;
2 [IA_W_x] W injectively agrees with x
      on TID_1_x;
3 [IA_x_y] x injectively agrees with y
      on TID_1_x;
4 [IA_y_x] y injectively agrees with x
      on TID_1_x;
```

6 Conclusions and future works

This paper presents a model-driven approach for the automated generation of formal models, oriented to the security analysis of blockchain-based protocols. The automated process has been applied to the specific case study of the Tweetchain protocol. At the current point of development, the paper means a first experiment in the application of model-driven principles to blockchain protocols design and verification. The experimentation leads to the definition of a first UML profile for the definition of generic transaction-based systems, and also to the definition of a specific notation for the Tweetchain data and messages, as well as on model-to-model transformations. The transformations generate analyzable Tamarin models from UML specifications.

The Tamarin models can be easily analyzed, and first optimization/simplification approaches show a meaningful performance and memory occupation improvement with respect to the hand-made model. Furthermore, MDE approaches enable a more reproducible modelling experience for the engineering since UML is a well-known language. Guidelines would complete the support to the modeller.

This experience means a first effort in the proposed toolchain. More work is needed to consolidate it by: (a) developing the profile according to the approach in Fig. 2, (b) improving it by improving security goals specification, and (c) applying the query-template approach at AnB level. Other, deeper future research threads could be considered:

- To explore other blockchain-based protocols to refine both the modelling and the generational approach;
- To explore A&B-level performance-optimized models and their generation from UML. This could be done in an open framework where optimization-oriented toolchain "bricks" may produce scalable code;
- To use and extend the UML profile to bridge the two main investigation lines of the blockchain modelling research, i.e., protocols (as in this work) and applications (e.g., smart contract generation and verification).

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 XXXVI cycle PhD program. S. Bernardi and J. Merseguer were supported by the Spanish Ministry of Science and Innovation [ref. PID2020-113969RB-I00].

Funding Open access funding provided by Università degli Studi della Campania Luigi Vanvitelli within the CRUI-CARE Agreement.

Appendix A: Background

A.1 UML and profiling

The Unified Modeling Language (UML) [45] is a general purpose, Object Management Group (OMG) standard, modeling language for software system specification. The

semantics of UML diagrams is expressed in natural language, while their abstract syntax is provided in terms of UML meta-models, which are models of a modelling language. UML encompasses different types of diagrams which enable to model both the structure and the behavior of a software system. Besides, UML has been also suggested in [1] for the design of decentralized blockchain applications.

In this work, we consider two types of behavioral diagrams: UML uses cases and sequence diagrams. In particular, the former are used to define the context of the protocol, in terms of the roles of the participants and the protocol phases. The latter is the most suitable diagram to be used for the protocol specification, that is the participants involved in the protocol and their interactions.

The UML profiling is a *lightweight* meta-modelling technique to extend UML, since the standard semantics of UML model elements can be refined in a strictly additive manner. Stereotypes and tags are the main extension mechanisms used to define a UML profile. In particular, a stereotype extends one or more UML meta-classes and can be applied to those UML model elements that are instantiations of the extended meta-classes. For example, in Table 1, the *Participant* stereotype extends the *Lifeline* meta-class, then the former can be applied to a lifeline of a sequence diagram (Fig. 5). Just like classes, a stereotype can have properties which are referred to as attributes or tags. When a stereotype is applied to a model element, the value assigned to a stereotype property is called *tagged-value*. In the previous example, *transactions* is a tag of *Participant* stereotype, with multiplicity *"*"* and type *Transaction*: thus, a tagged-value is a list of *Transaction*-typed values.

A.2 Alice and Bob notation

The Alice & Bob notation is a simple and intuitive language useful to specify communication protocols.

More in detail, the principals involved in an A&B protocol are in a finite number and usually denoted with *Alice*, *Bob*, *Charlie*, ..., and a protocol, in this notation, is a list of messages.

The structure of a message clearly depends on the communication type, but in general it is of the following type:

$$A \rightarrow B : msg$$

in which A and B are honest principals and A sends a message msg to B. As an example, a message could consist of a plaintext ciphered with the public key of B or a nonce authenticated with the secret key of A, and the A&B notation would be, respectively:

$$A \rightarrow B : \{plaintext\}_{pk(B)}$$

$$A \rightarrow B : \{nonce\}_{sk(A)}$$

The AnB language [6], used in the automatic translation from AnB to Tamarin [44], is a formal language based on the Alice and Bob notation. In particular, there are four main blocks in an AnB protocol specification that might be used. First, a *Declaration* block can be used to explicitly state user-defined functions, in order to be able to use them in the protocol. Then, for each participant there might be the need to describe its initial *Knowledge*, that describes the values that are known to a principal at the very beginning (i.e., before the sending of the first message) and at any time during protocol execution. Then, there is an *Actions* block, which is mandatory, since it specifies the protocol itself; it consists of a sequence of messages, as stated before, but in this case they have the following form

$$[label] \;\; A \rightarrow B\,(n_1, \ldots, n_y) : msg$$

where $label$ is a unique identifier given to the specific message exchange and $(n_1, \ldots, n_y)$ are fresh variables used in msg.

Finally, there is a *Goals* block that includes the specification of the security properties to be verified. The properties concern the secrecy and authenticity of messages.

An example of authenticity property is the following:

$$[auth] \;\; A \text{ injectively agrees with } B \text{ on } n$$

The meaning of this goal is that whenever A completes a run of the protocol—apparently with B—then B has been running the protocol—apparently with A—and the two participants have a consistent view on n (that is, they have the same value for it).

A.3 Tamarin prover

The Tamarin Prover [3] is a symbolic model checker largely used for the modeling and analysis of cryptographic protocols. 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 [38]: 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 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. An example of the latter type is shown in Listing 6.

Listing 6 All-traces lemma

```
lemma propertytocheck:
  " All A B m #i. Message_received(A,B,m)
    @i
    ==>
    Ex #j. Message_sent(A,B,m) @j & j<i"
```

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 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 [46].

References

1. Ramamurthy, B.: Blockchain in Action. Manning, Shelter Island (2020)
2. Yaga, D., Mell, P., Roby, N., Scarfone, K.: Blockchain Technology Overview. Technical report, National Institute of Standards and Technology (2018). https://doi.org/10.6028/NIST.IR.8202
3. Meier, S., Schmidt, B., Cremers, C., Basin, D.: 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, pp. 696–701 (2013). https://doi.org/10.1007/978-3-642-39799-8_48
4. Boyd, C., Gjøsteen, K., Wu, S.: A Blockchain Model in Tamarin and Formal Analysis of Hash Time Lock Contract. In: Bernardo, B., Marmsoler, D. (eds.) 2nd Workshop on Formal Methods for Blockchains, FMBC@CAV 2020, July 20-21, 2020, (Virtual Conference). OASIcs, vol. 84, pp. 5–1513. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Los Angeles (USA) (2020). https://doi.org/10.4230/OASIcs.FMBC.2020.5
5. Basin, D., Keller, M., Radomirović, S., Sasse, R.: Alice and Bob Meet Equational Theories. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9200, pp. 160–180 (2015). https://doi.org/10.1007/978-3-319-23165-5_7
6. Mödersheim, S.: Algebraic Properties in Alice and Bob Notation. In: Proceedings of the The Forth International Conference on Availability, Reliability and Security, ARES 2009, March 16–19, 2009, pp. 433–440. IEEE Computer Society, Fukuoka (2009). https://doi.org/10.1109/ARES.2009.95
7. Buccafurri, F., Lax, G., Nicolazzo, S., Nocera, A.: Overcoming Limits of Blockchain for IoT Applications. In: ACM International Conference Proceeding Series, vol. Part F130521 (2017). https://doi.org/10.1145/3098954.3098983
8. Raimondo, M., Bernardi, S., Marrone, S.: On formalising and analysing the Tweetchain protocol. In: ICISSP 2021—Proceedings of the 7th International Conference on Information Systems Security and Privacy, pp. 781–791 (2021)
9. Singh, A., Parizi, R.M., Zhang, Q., Choo, K.-K.R., Dehghantanha, A.: Blockchain smart contracts formalization: approaches and challenges to address vulnerabilities. Comput. Secur. **88**, 101654 (2020). https://doi.org/10.1016/j.cose.2019.101654
10. Duan, Z., Mao, H., Chen, Z., Bai, X., Hu, K., Talpin, J.P.: Formal modeling and verification of blockchain system. In: Proceedings of the 10th International Conference on Computer Modeling and Simulation. ICCMS 2018, pp. 231–235. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3177457.3177485
11. Egger, C., Graf, M., Küsters, R., Rausch, D., Ronge, V., Schröder, D.: A Security Framework for Distributed Ledgers. IACR Cryptol. ePrint Arch., vol. 145 (2021)
12. Thin, W.Y.M., Dong, N., Bai, G., Dong, J.S.: Formal analysis of a proof-of-stake blockchain. In: 2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 197–200 (2018). https://doi.org/10.1109/ICECCS2018.2018.00031
13. Tolmach, P., Li, Y., Lin, S.-W., Liu, Y.: Formal Analysis of Composable DeFi Protocols. In: Bernhard, M., Bracciali, A., Gudgeon, L., Haines, T., Klages-Mundt, A., Matsuo, S., Perez, D., Sala, M., Werner, S. (eds.) Financial Cryptography and Data Security. FC 2021 International Workshops, pp. 149–161. Springer, Berlin (2021)
14. Modesti, P., Shahandashti, S.F., McCorry, P., Hao, F.: Formal modelling and security analysis of Bitcoin's payment protocol. Comput. Secur. **107**, 102279 (2021). https://doi.org/10.1016/j.cose.2021.102279
15. Camenisch, J., Krenn, S., Küsters, R., Rausch, D.: iUC: flexible universal composability made simple. In: Advances in Cryptology—ASIACRYPT 2019—25th International Conference on the Theory and Application of Cryptology and Information Security, December 8–12, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 191–221. Springer, Kobe (2019)
16. Sun, J., Liu, Y., Dong, J.S., Chen, C.: Integrating Specification and Programs for System Modeling and Verification. In: 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering, pp. 127–135 (2009). https://doi.org/10.1109/TASE.2009.32
17. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, pp. 709–714. Springer, Berlin (2009)
18. Basin, D.A., Mödersheim, S., Viganò, L.: OFMC: a symbolic model checker for security protocols. Int. J. Inf. Secur. **4**(3), 181–208 (2005). https://doi.org/10.1007/s10207-004-0055-7
19. König, L., Korobeinikova, Y., Tjoa, S., Kieseberg, P.: Comparing blockchain standards and recommendations. Future Internet (2020). https://doi.org/10.3390/fi12120222
20. Blockchain Ecosystem Interoperability. Technical report, Object Management Group (2019). RFI: mars/19-08-03

21. Ellervee, A., Matulevičius, R., Mayer, N.: A comprehensive reference model for blockchain-based distributed ledger technology. In: ER Forum/Demos (2017)
22. Skotnica, M., Pergl, R.: Das contract—a visual domain specific language for modeling blockchain smart contracts. In: Aveiro, D., Guizzardi, G., Borbinha, J. (eds.) Advances in Enterprise Engineering XIII, pp. 149–166. Springer, Cham (2020)
23. UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification. Technical report, Object Management Group (2005). formal-08-04-05
24. Lodderstedt, T., Basin, D., Doser, J.: Secureuml: a UML-based modeling language for model-driven security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML2002—The Unified Modeling Language, pp. 426–441. Springer, Berlin (2002)
25. Jürjens, J.: Secure Systems Development with UML. Springer, Berlin (2005). https://doi.org/10.1007/b137706
26. Rodríguez, R.J., Merseguer, J., Bernardi, S.: Modelling security of critical infrastructures: a survivability assessment. Comput. J. **58**(10), 2313–2327 (2015). https://doi.org/10.1093/comjnl/bxu096
27. Basin, D., Clavel, M., Doser, J., Egea, M.: Automated analysis of security-design models. Inf. Softw. Technol. **51**(5), 815–831 (2009). https://doi.org/10.1016/j.infsof.2008.05.011
28. Jürjens, J., Fox, J.: Tools for model-based security engineering. In: Osterweil, L.J., Rombach, H.D., Soffa, M.L. (eds.) 28th International Conference on Software Engineering (ICSE 2006), May 20–28, 2006, pp. 819–822. ACM, Shanghai (2006). https://doi.org/10.1145/1134285.1134423
29. Kent, S.: Model driven engineering. In: Butler, M.J., Petre, L., Sere, K. (eds.) IFM. Lecture Notes in Computer Science, vol. 2335, pp. 286–298. Springer, Berlin (2002)
30. Ivanov, I., Bézivin, J., Aksit, M.: Technological spaces: an initial appraisal. In: 4th International Symposium on Distributed Objects and Applications, DOA 2002—University of California, Irvine, United States, pp. 1–6 (2002). https://research.utwente.nl/en/publications/technological-spaces-an-initial-appraisal
31. Bézivin, J., Devedzic, V., Djuric, D., Favreau, J.-M., Gasevic, D., Jouault, F.: An m3-neutral infrastructure for bridging model engineering and ontology engineering. In: Konstantas, D., Bourrières, J.-P., Léonard, M., Boudjlida, N. (eds.) Interoperability of Enterprise Software and Applications, pp. 159–171. Springer, London (2006)
32. Bézivin, J., Kurtev, I.: Model-based Technology Integration with the Technical Space Concept. Metainformatics Symposium (2006). https://hal.archives-ouvertes.fr/hal-00483587
33. PlantUML. https://plantuml.com/en/sequence-diagram. Accessed 11 July 2021
34. Web Sequence Diagrams. https://www.websequencediagrams.com/. Accessed 11 July 2021
35. Cortellessa, V., Marco, A.D., Inverardi, P.: Model-Based Software Performance Analysis. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-13621-4
36. Bernardi, S., Merseguer, J., Petriu, D.C.: Model-Driven Dependability Assessment of Software Systems. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-39512-3
37. Bernardi, S., Gentile, U., Marrone, S., Merseguer, J., Nardone, R.: Security modelling and formal verification of survivability properties: application to cyber-physical systems. J. Syst. Softw. **171**, 110746 (2021). https://doi.org/10.1016/j.jss.2020.110746
38. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (1983). https://doi.org/10.1109/TIT.1983.1056650
39. Bernardi, S., Merseguer, J., Petriu, D.C.: A dependability profile within MARTE. Softw. Syst. Model. **10**(3), 313–336 (2011). https://doi.org/10.1007/s10270-009-0128-1
40. Bernardi, S., Flammini, F., Marrone, S., Mazzocca, N., Merseguer, J., Nardone, R., Vittorini, V.: Enabling the usage of UML in the verification of railway systems: the dam-rail approach. Rel. Eng. Sys. Saf. **120**, 112–126 (2013). https://doi.org/10.1016/j.ress.2013.06.032
41. Selic, B.: A systematic approach to domain-specific language design using UML. In: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), pp. 2–9 (2007)
42. Lagarde, F., et al.: Improving UML profile design practices by leveraging conceptual domain models. In: 22nd International Conference on Automated Software Engineering, pp. 445–448. ACM, Atlanta (2007)
43. Buccafurri, F., Lax, G., Nicolazzo, S., Nocera, A.: 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, pp. 386–393 (2017). https://doi.org/10.1007/978-3-319-60131-1_24
44. Keller, M.: Converting Alice&Bob Protocol Specifications to Tamarin. Bachelor's Thesis, Swiss Federal Institute of Technology Zurich (2014)
45. OMG: Unified Modelling Language: Superstructure. Object Management Group (2015). Object Management Group. Version 2.5, formal/15-03-01
46. Cortier, V., Delaune, S., Dreier, J.: 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, pp. 3–22 (2020). https://doi.org/10.1007/978-3-030-59013-0_1