Modeling and Verifying NDN-based IoV Using CSP

Ningning Chen[1], Huibiao Zhu[1,*], Jiaqi Yin[1], Lili Xiao[1], Yuan Fei[2,*]
[1]Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
[2] College of Information, Mechanical and Electrical Engineering,
Shanghai Normal University, Shanghai, China

***Abstract*—As a crucial component of intelligent transportation system, Internet of Vehicles (IoV) plays an important role in the smart and intelligent cities. However, current Internet architectures cannot guarantee efficient data delivery and adequate data security for IoV. Therefore, Named Data Networking (NDN), a leading architecture of Information-Centric Networking (ICN), is introduced into IoV. Although problems about data distribution can be resolved effectively, the combination of NDN and IoV causes some new security issues.**

In this paper, we apply Communicating Sequential Processes (CSP) to formalize NDN-based IoV. We mainly focus on its data access mechanism and model this mechanism in detail. By feeding the formalized model into the model checker Process Analysis Toolkit (PAT), we verify four vital properties including deadlock freedom, data availability, PIT deletion faking and CS caching pollution. According to verification results, the model cannot ensure the security of data with the appearance of intruders. To solve these problems, we adopt a method derived from Blockchain in our improvement. Through the analysis of the improved model, we can truly guarantee the security of NDN-based IoV.

***Index Terms*—NDN, IoV, CSP, Blockchain, Modeling Verification**

I. INTRODUCTION

Internet of Vehicles (IoV) [1] arouses wide public concern in both industry and academia sectors files. However, the current Internet is a point-to-point communication and channel-based security model. In IoV, the current Internet architectures cannot ensure high-efficiency data distribution and sufficient data security. To resolve this problem, Name Data Networking (NDN) [2] is introduced into IoV. NDN is a crucial architecture of Information-Centric Networking (ICN) [3]. NDN uses data names instead of IP addresses to retrieve and identify data. Due to this characteristic, NDN can better meet IoV's demand for big data processing.

There are some work on the related files of IoV. Abbas et al. proposed an road-aware estimation model for path duration in IoV [4]. In order to develop software-defined wireless network in IoV, Chien et al. created a SFC-based access point switching mechanism [5]. However, problems about data distribution and data security still exist. Therefore, NDN was applied to IoV in some researches. Su et al. presented a novel framework of a content-centric vehicular network (CCVN) [6]. Chowdhury et al. created a novel forwarding strategy CCLF [7]. Kalogeiton et al. defined a geographical aware routing protocol using directional antennas [8]. From these existing work, we find that they mainly focus on routing and forwarding. Unfortunately, there are still many security issues in NDN-based IoV.

In this paper, formal methods are used to verify and analyze NDN-based IoV. This paper uses Communicating Sequential Processes (CSP) [9] to formalize the system. Using Process Analysis Toolkit (PAT) [10], we verify four properties (deadlock freedom, data availability, PIT deletion faking and CS caching pollution). Under the interference of intruders, the last three verification properties are invalid for the model. This paper adopts a method based on Blockchain in the improvement. All properties are satisfied for the improved model.

The rest of this paper is organized as follows. Section II presents an overview of NDN-based IoV and Blockchain. In Section III, we formalize the model of NDN-based IoV. In Section IV, we verify four properties and give improvement to the model. Finally, conclusion and future work are described in Section V.

II. BACKGROUND

This section gives a brief introduction of NDN-based IoV and Blockchain, especially combinations of them. After that, we also describe process algebra CSP.

A. NDN-based IoV

Some studies apply NDN to IoV to improve its security and performance. To further improve its performance, a vehicle may obtain much data by sending an interest packet. So we must take separation of interest packets and aggregation of data packets into consideration. Vehicles could be consumers or it could be producers of data. This scheme contains the following entities :

- **Consumer and Producer:** Consumers are vehicles who request and consume data packets. As a producer, the vehicle produces and provides data packets.
- **RSU (Road Side Unit):** An RSU forwards interest packets and data packets for vehicles. An RSUC and an RSUP represent the behavior of an RSU which communicates with a consumer and a producer separately.
- **Router:** Routers manage the access processes.

When a consumer wants to get a data packet, the following sequence of actions occurs:

*Corresponding Author. E-mail address: hbzhu@sei.ecnu.edu.cn (H. Zhu), yuanfei@shnu.edu.cn (Y. Fei).

DOI reference number: 10.18293/SEKE2020-066

- A consumer sends an interest packet containing the required data names to an RUSC.
- The RSUC transmits the interest packet to a router.
- The router addresses the interest packet with process *Interest Processing*. If there is a data packet matching this interest packet in its CS, the router returns this data packet. Otherwise, this process goes to next step.
- A router obtains the corresponding data through network propagation.
- Then it deals with received data packets through process *Data Processing* and may forward a data packet made up of all received data to corresponding RSUC.
- The consumer gets the data packet from the RSUC and verifies the data packet.

The process of producing data is defined similarly, the following sequence of actions occurs:

- A router gets an interest packet from the network and disposes it with process *Interest Processing*.
- A special case is that the matching data packet can be got from a producer managed by this router.
- The router sends the interest packet to an RSUP.
- The RSUP obtains the data packet from a producer.
- The RSUP returns it to the router.
- The router addresses this data packet with process *Data Processing*.
- According to the verification results of this data packet, it might be injected into the network.

As shown in Fig.1, the interest packets and data packets forwarding processes are described as follows:

Interest Processing: Whenever a router receives an interest packet, it queries CS (Content Store) to find out a data packet matching received interest packet. If so, the data packet is returned to the corresponding request entities and the interest packet is discarded. Otherwise, the router finds matching entries in PIT (Pending Interest Table). If it finds an entry, it just adds the incoming interface of the interest packet to that entry. If not, the router queries its FIB (Forwarding Information Base) to find the outgoing interface for each data name. If the router finds out all interfaces, it adds a new entry to PIT including the data names and incoming interface of the interest packet. Then it puts the data names of the same forwarding interface into a new interest packet and forwards those new interest packets. Otherwise, this received interest packet is thrown away.

Data Processing: When a router receives a data packet, it traverses PIT with packet's name firstly. If there are interest packets waiting for the data packet, the router saves the packet into CS. If the router receives all the data required for a PIT entry, the entry is deleted. The data packet containing all data is emitted from the interfaces that the entry records. The data packet is cached into the CS.

B. Blockchain

We bring Blockchain into NDN-based IoV in this paper. The blockchain generation process contains two parts. On the one hand, transactions are produced, forwarded and verified.

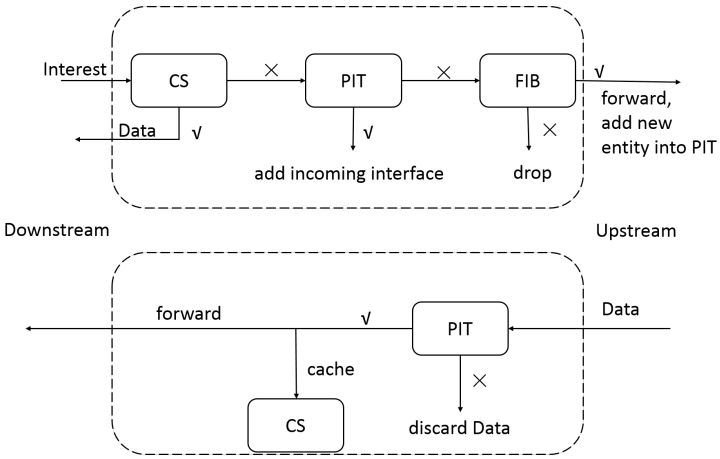

Fig. 1. Processing of interest and data packets in NDN [2]

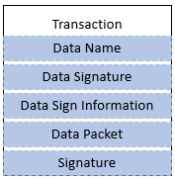

Fig. 2. Transaction

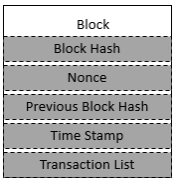

Fig. 3. Block

On the other hand, entities produce, forward, verify and store blocks. In Section IV, we give a detailed description for these two parts. The structures of transactions and blocks are shown in Fig.2 and Fig.3 respectively. *Data Packet* represents the result of our repeated hash calculation of hash values (*Data Name*, *Data Signature* and *Data Sign Information*). A producer signs these hash values using its private key. Those components make up a transaction. A blockchain is composed of blocks. A block contains *Block Hash*, *Nonce*, *Previous Block Hash*, *Time Stamp* and *Transaction List*. *Block hash* is the unique identifier distinguishing a block. All transactions of this block are stored in *Transaction List*.

In the improvement of NDN-based IoV, the system undergoes the following main changes:

- Producers create a transaction for each data packet and transmit the transaction to other entities.
- If a transaction passes verification, routers store and forward it.
- A router creates a block containing all transactions what it has saved so far. Then the router adds the block into its blockchain and forwards this block.
- According to verification results of a received block, entities add this block into their blockchains and delete their transactions that are repeated in this block.
- When an entity receives a data packet, it verifies received data packet using its blockchain firstly.

C. CSP

This section is used to introduce CSP (Communication Sequential Process). We give part of CSP syntax as follows:

$$P, Q \ ::= \ SKIP \mid STOP \mid a \rightarrow P \mid c?x \rightarrow P \mid c!u \rightarrow P \mid P; Q \mid \\ \mid P||Q \mid P \Box Q \mid P \lhd B \rhd Q \mid P[[a \leftarrow b]] \mid P[|c|]Q$$

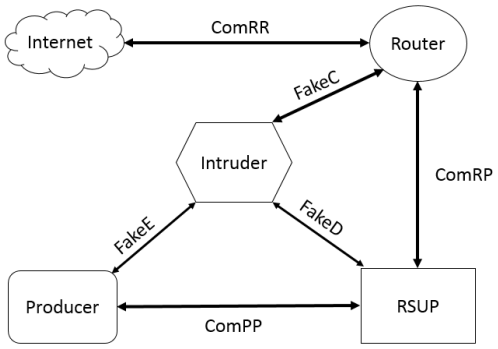

Fig. 4. Producer Modeling

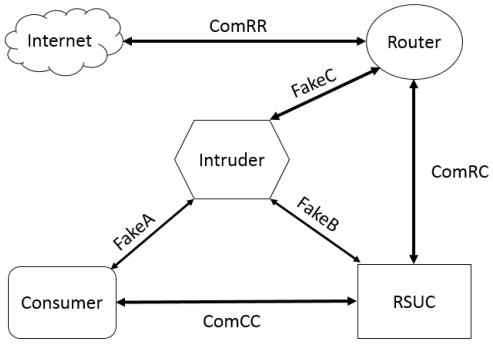

Fig. 5. Consumer Modeling

- $SKIP$ represents that a process terminates successfully.
- $STOP$ indicates that a process runs into a deadlock state.
- $a \rightarrow P$ denotes that a process P executes after event a.
- $c?x \rightarrow P$ represents that a process receives a value and assigns it to the variable x before executing process P.
- $c!x \rightarrow P$ describes that a process sends a value v through channel c, then process P is executed.
- $P;Q$ represents that process Q is executed after process P terminates successfully.
- $P||Q$ denotes that process Q and process P are executed in parallel.
- $P \square Q$ stands for general choice. Selecting process P or process Q depends on external environment.
- $P \lhd B \rhd Q$ is a conditional choice. If boolean expression B is true, process P will be executed. Otherwise, process Q is executed.
- $P[[a \leftarrow b]]$ indicates that a process changes event a for event b.
- $P[|c|]Q$ represents that processes P and Q execute the concurrent events on the set c of channels.

III. MODELING NDN-BASED IOV MODEL

In this section, we model NDN-based IoV by using CSP. This model is formalized based on Section II.

A. Sets, Messages and Channels

For convenience, this section gives some crucial information about sets, messages and channels models used in our formulation. Six sets are defined. *Entity* set contains entities including *Consumer*, *Producer*, *RSUC*, *RUSP* and *Router*. *Name* set denotes the names of data. **PRKey** set is composed of entities' public keys. **PUKey** set includes entities' private keys. The set *SigInfo* indicates signature information. *Content* represents other message contents.

In order to describe message packets transmitted between entities, this section defines some messages based on those definitions. *E(k,c)* indicates that key *k* encrypts content *c* . Each message includes a tag from the set $\{msg_{int}, msg_{data1}, msg_{data2}\}$. The messages are transmitted among entities as follows:

$$\begin{aligned} MSG_{int} &= \{msg_{int}.a.b.n \mid \ a, b \in Entity, n \in Name\} \\ MSG_{data1} &= \{msg_{data1}.a.b.n.E(K1^{-1}, c).sigin \mid \\ &\quad a, b \in Entity, K1^{-1} \in PRKey, \\ &\quad n \in Name, c \in Content, sigin \in SigInfo\} \end{aligned}$$

$$\begin{aligned} MSG_{data2} &= \{msg_{data2}.a.b.n1.n2.E(K1^{-1}, c1). \\ &\quad E(K2^{-1}, c2).sigin1.sigin2 \mid \\ &\quad a, b \in Entity, K1^{-1}, K2^{-1} \in PRKey, \\ &\quad n1, n2 \in Name, c1, c2 \in Content, \\ &\quad sigin1, sinin2 \in SigInfo\} \\ MSG_{data} &= MSG_{data1} \cup MSG_{data2} \\ MSG &= MSG_{int} \cup MSG_{data} \end{aligned}$$

MSG_{data} and MSG_{int} represent the messages of data packets and interest packets respectively. MSG denotes the messages between all entities.

To model the communication among components, we give the definitions of channels.

- channels between consumers, RSUC, routers and RSUP described by *COM_PATH*:
$$ComCC, ComRS, ComRR, ComRP, ComPP$$
- channels of intruders intercepting consumers, RSUC, routers and RSUP constituted by *INTRUDER_ PATH*:
$$FakeA, FakeB, FakeC, FakeD, FakeE$$

The declarations of the channels are as follows:

$$Channel \ \ COM\_PATH, INTRUDER\_PATH : MSG$$

B. Overall Modeling

We define a CSP model *System0* without intruders. *System0* is composed of *SystemC* and *SystemP*. *SystemC* and *SystemP* indicate a system which consumes and produces data respectively. The *SystemC* model is made up of three main entities including *Consumer*, *RSUC* and *Router*. *SystemP* model contains *Producer*, *RSUP* and *Router*. To take intruders into consideration, we create *SYSTEM* based on *System0*.

$$\begin{aligned} SystemC =_{df} & \\ & Consumer[|COM\_PATH|]RSUC \\ & [|COM\_PATH|]Router \\ SystemP =_{df} & \\ & Producer[|COM\_PATH|]RSUP \\ & [|COM\_PATH|]Router \\ System0 =_{df} & \\ & SystemC[|COM\_PATH|]SystemP \\ SYSTEM =_{df} & \\ & System0[|INTRUDER\_PATH|]Intruder \end{aligned}$$

Consumer, *Producer* and *Router* denote the behavior of the consumers, producers and routers respectively. *RSUC* and

RSUP describe the actions of an RSU which communicates with a consumer and a product respectively. Considering the existence of intruders, this part defines *Intruder* which intercepts and fakes the messages. Fig.4 and Fig.5 describe interprocesses communication between processes.

C. Router Modeling

In NDN-based IoV, a router is responsible for forwarding interest packets and getting data packets. A router incisions and distributes an interest packet on the grounds of its routing information. For simplicity, each interest packet contains a maximum of two data names in our formalization. Then the router aggregates and forwards received data packets on the basis of verification results. This section formalizes a process $Router_{io}$ without intruders.

$$
\begin{aligned}
&Router_{io}(CSTable_i, PITTable_i, FIBTable_i) =_{df}\\
&Initia\{inFIB = false; inCS = false; inPIT = false;\\
&inFIB1 = false; inFIB2 = false; delPIT = false;\\
&addcs = false; ininterface = 0; outface1 = 0; outface2 = 0\}\\
&\rightarrow ComRS_i?B.C.dn1.dn2 \rightarrow ininterfece := i \rightarrow\\
&\left(\begin{array}{l}(inCS = true)\\ \lhd(\exists entry \in CSTable_i \bullet entry.data1name == dn1\\ \wedge entry.data2name == dn2) \rhd (inCS = false)\end{array}\right);\\
&\left(\begin{array}{l}(ComRS_{interface}!C.B.dn1.dn2.CSTable_i[CSindex][2]\\ .CSTable_i[CSindex][3].CSTable_i[CSindex][4].\\ CSTable_i[CSindex][5].CSTable_i[CSindex][6])\\ \lhd(inCS == true) \rhd (NOCS_i)\end{array}\right);\\
&Router_{i0}
\end{aligned}
$$

There are several variables appeared in the process. *i* is the ID of a router and its channels. A router knows its CS table $CSTable_i$, PIT table $PITTable_i$ and FIB table $FIBTable_i$. *inFIB*, *inFIB2* and *inFIB2* represent querying results for FIB. Similarly, *inCS*, *inPIT* and *inPIT1* denote searching results for CS and PIT respectively. *delPIT* indicates the router deletes PIT entries. *addcs* represents the router adds a data packet into CS. *ininterface* records the incoming interfaces of interest packets. *outinterface1* and *outinterface2* are the outgoing interfaces of interest packets.

First, the router receives an interest packet including the names of required data and records the incoming interface for the interest packet. Then the router checks its CS to find if there is a data packet matching the interest packet. If the result is positive, the data packet is returned to the corresponding request nodes and the interest packet is discarded. Otherwise, we define process $NOCS_i$ to address the situation in which the desired data packet cannot be obtained from CS.

$$
\begin{aligned}
&NOCS_i =_{df}\\
&\left(\begin{array}{l}\left(\begin{array}{l}inPIT = true;\\ AddPIT(dn1, dn2, entry.index,\\ interface, PITTable_i)\end{array}\right)\\ \lhd(\exists entry \in PITTable_i \bullet entry.data1name == dn1\\ \quad \wedge entry.data2name == dn2) \rhd\\ (inPIT = false; NeedForward_i)\end{array}\right);
\end{aligned}
$$

$NOCS_i$ traversals $PITTable_i$ according to the names in the interest packet. If $NOCS_i$ finds a matching entry, the router adds incoming interface of the interest packet into the entry and discards the interest packet. If not, we give process $NeedForward_i$ to forward the interest packet.

$$
\begin{aligned}
&NeedForward_i =_{df}\\
&\left(\begin{array}{l}(inFIB1 = true; outface1 = entry2.outface)\\ \lhd(\exists entry2 \in FIBTable_i \bullet entry2.dataname == dn1) \rhd\\ (inFIB1 = false;)\end{array}\right);\\
&\left(\begin{array}{l}(inFIB2 = true; outface2 = entry2.outface)\\ \lhd(\exists entry2 \in FIBTable_i \bullet entry2.dataname == dn1) \rhd\\ (inFIB2 = false)\end{array}\right);\\
&\left(\begin{array}{l}(SKIP) \lhd (!(inFIB1 \wedge inFIB2)) \rhd\\ \left(\begin{array}{l}(AddPIT1(dn1, dn2, PITlength,\\ interface, PITTable_i); Forward_i\end{array}\right)\end{array}\right); SKIP
\end{aligned}
$$

$NeedForward_i$ queries $FIBTable_i$ with the names of the interest packet. If process $NeedForward_i$ finds an outgoing interface for each name, $NeedForward_i$ adds a new entry into $PITTable_i$ including the data names and incoming interface of the interest packet. Then we define process $Forward_i$ to forward the interest packet according to query results for each name. If $NeedForward_i$ cannot find all outgoing interfaces, the interest packet is dropped.

$$
\begin{aligned}
&Forward_i =_{df}\\
&\left(\begin{array}{l}\left(\begin{array}{l}ComRR_{outface1}!msg_{int}.C.D.dn1.dn2 \rightarrow\\ ComRR_{outface1}?msg_{data}D.C.dn1.dn2.\\ E(prk1\_Key, data1).E(prk2\_Key, data2).\\ siginfo1.siginfo2 \rightarrow Datapacketreceive_i\end{array}\right)\\ \lhd(outface1 == outface2) \rhd\\ \left(\begin{array}{l}(ComRR_{outface1}!msg_{int}.C.D.dn1 \rightarrow\\ ComRR_{outface1}?msg_{data}D.C.dn1.\\ E(prk1\_Key, data1).siginfo1 \rightarrow\\ ComRR_{outface2}!msg_{int}.C.D.dn2 \rightarrow\\ ComRR_{outface2}?msg_{data}D.C.dn2.\\ E(prk2\_Key, data2).siginfo2 \rightarrow\\ Datareceive_i\end{array}\right)\end{array}\right); SKIP
\end{aligned}
$$

If all data can be got by one outgoing interface, $Forward_i$ transmits the interest packet from this interface and waits a matching data packet. If not, $Forward_i$ produces new interest packets by putting the data names of the same outgoing interface into a new interest packet. Then $Forward_i$ forwards each interest packet from corresponding outgoing interfaces and waits data packets. $Datareceive_i$ deals with received data packets.

$$
\begin{aligned}
&Datareceive_i =_{df}\\
&\left(\begin{array}{l}\left(\begin{array}{l}inPIT = true; Datasend_i(entry3.index);\\ DelPIT(dn1, dn2, PITTable_i);\\ AddCS(dn1, dn2, E(prk1\_Key, data1),\\ E(prk2\_Key, data2),\\ siginfo1, siginfo2, CSTable_i)\\ delPIT = true; addcs = true\end{array}\right)\\ \lhd(\exists entry3 \in PITTable_i \bullet entry3.data1name == dn1\\ \wedge entry3.data2name == dn2) \rhd\\ (\ inPIT = false; delPIT = false; addcs = false\ )\end{array}\right);
\end{aligned}
$$

Before introducing process $Datareceive_i$, we define two functions. $DelPIT$ function denotes that the router deletes entries from the $PITTable_i$ according to the input data names. $AddCS$ function represents that the router adds a data packet into its CS. When $Datareceive_i$ receives data packets, $Datareceive_i$ checks $PITTable_i$. If there is an interest packet waiting for these data packets, the entry is deleted and these data packets are emitted through $Datasend_i(index)$. Otherwise, the received data packets are abandoned.

$$
\begin{aligned}
&Datasend_i(index) =_{df}\\
&Intail\{interfacelist = PITTbale[index].interfcelist;\\
&x = 0; y = \#interfacelist;\} \rightarrow
\end{aligned}
$$

$(0 \leq x < y)*$

$$\left(\begin{array}{l} ComRS_{interfecelist[x]}!msg_{data}C.B.dn1.dn2. \\ E(prk1\_Key, data1).E(prk2\_Key, data2). \\ siginfo1.siginfo2 \rightarrow x = x + 1 \end{array} \right); SKIP$$

$Datasend_i(index)$ gets the entry of $PTITable_i$ with index *index*. Then it records the incoming interface list *interfacelist* of the entry. Variables *x* and *y* represent entries' index variables and lengths of *interfacelist* respectively. $Datasend_i(index)$ queries *interfacelist* and sends a data packet, containing all data, from the incoming interfaces recorded in *interfacelist*.

$Router_i =_{df}$
$Router_{io}[[$
$ComRS_i!\{|ComRS_i|\} \leftarrow ComRS_i!\{|ComRS_i|\},$
$ComRS_i!\{|ComRS_i|\} \leftarrow FakeC_i!\{|ComRS_i|\},$
$ComRS_i?\{|ComRS_i|\} \leftarrow ComRS_i?\{|ComRS_i|\},$
$ComRS_i?\{|ComRS_i|\} \leftarrow FakeC_i?\{|ComRS_i|\},$
$ComRR_i!\{|ComRR_i|\} \leftarrow ComRR_i!\{|ComRR_i|\},$
$ComRR_i!\{|ComRR_i|\} \leftarrow FakeC_i!\{|ComRR_i|\},$
$ComRR_i?\{|ComRR_i|\} \leftarrow ComRR_i?\{|ComRR_i|\},$
$ComRR_i?\{|ComRR_i|\} \leftarrow FakeC_i?\{|ComRR_i|\}]]$

$\{|c|\}$ denotes the set of all communications over channel c. Whenever $Reader_{io}$ does an action on channel ***ComRS***, $Reader_{io}$ will execute actions on channel ***ComRS*** or channel ***FakeC***. $Reader_{io}$ carries out either actions on channel ***ComRR*** or ***FakeC*** when $Reader_{io}$ performs actions on channel ***ComRR***. Besides, $Reader_i$ performs the same actions as $Reader_{i0}$.

D. Intruder Modeling

In order to take intruders into consideration, this subsection builds *Intruder* process. It intercepts or fakes messages in the communication via channel ***ComCC, ComRS, ComRR, ComRP*** and ***ComPP***.

At first, we define the set of facts which intruders might learn.

$$Fact =_{df} Entity \cup MSG_{out} \cup Name \cup \{E(key, c) | key \in PRKey, c \in Content\}$$

Intruder can derive new facts from the set of facts which intruders have learned. Symbol $F \mapsto f$ is used to indicate that the fact f can be deduced from the set F of facts.

$$\{K, E(K^{-1}, c)\} \rightarrow c$$
$$\{K^{-1}, c\} \rightarrow E(K^{-1}, c)$$
$$F \mapsto f \wedge F \subseteq F' \Longrightarrow F' \mapsto f$$

The first two rules denote decryption and encryption respectively. The final rule represents that if a intruder can derive the fact f from the known fact F, the intruder can also deduce fact f from a larger set F'.

We define *Info* function to describe that intruders can obtain facts from messages, shown as follows

$Info(msg_{int}.a.b.n) =_{df} \{a, b, n\}$
$Info(msg_{data}.a.b.n.E(K^{-1}, c).sig) =_{df} \{a, b, E(K^{-1}, c), n, sig\}$
$Info(msg_{data}.a.b.c) =_{df} \{a, b, c\}$
where $a, b \in Entity,\ n \in Name,\ K^{-1} \in PRKey,$
$c \in Content, sig \in SigInfo$

We give a channel *Deduce* for intruders. Intruders can deduce new facts via channel *Deduce*, shown as follows :

Channel $Deduce : Fact.P(Fact)$

A intruder overheads all messages transmitted between entities. New facts can be deduced from the intruder's known facts. If a intruder gets all sub messages, it can fake some messages and send those messages to other entities. We formalize $Intruder_0$ as below:

$Intruder_0(F) =_{df}$
$_{m \in MSG_{out}} Fake?m \rightarrow Intruder_0(F \cup Info(m))$
$\square\square_{m \in MSG_{out} \cap Info(m) \subseteq F} Fake!m \rightarrow Intruder_0(F)$
$\square\square_{f \in Fact, f \notin F, F \mapsto f} Deduce.f.F \rightarrow Intruder_0(F \cup \{f\})$

This subsection uses set *Fake* to represent all channels of $INTRUDER\_PATH$. In the first part, a intruder gets messages via a channel of *Fake*. Then the intruder adds those messages to its knowledge. In the second part, a intruder fakes some messages according to its knowledge and sends faking messages to other entities. In the third part, a intruder can deduce some new facts from its knowledge via channel *Deduce*. Then the intruder adds the speculative results to its knowledge. We define *IK* to denote the initial knowledge of the intruder:

$$Intruder =_{df} Intruder_0(IK)$$
$$IK =_{df} \{A, B, C, D, E, Ipk\_Key, Irk\_Key\}$$

IV. VERIFICATION AND IMPROVEMENT

In this section, we use model checker PAT to verify four properties, including deadlock freedom, data availability, PIT deletion faking and CS caching pollution. According to verification results, we improve the system by referencing Blockchain.

A. Properties Verification

We use $System()$ to denote the original model. This subsection uses Linear Temporal Logic (LTL) formulas to describe four security properties. By using LTL formulas in PAT code, we give some assertions to help our verification.

Property 1: Deadlock Freedom

$$\#assert\ \ System()\ \ deadlockfree;$$

$System()$ should not run into a deadlock state. In PAT, there is a primitive to describe this situation.

Property 2: Data Availability

$$\#define\ \ Data\_Arival\_Success\ \ dataarive == true;$$
$$\#assert\ \ System()\ \ reaches\ \ Data\_Arive\_Success;$$

In NDN-based IoV, each entity should get the desired data. In order to guarantee this situation, we give this assertion.

Property 3: PIT Deletion Faking

$$\#define\ \ PITdelete\_Faking$$
$$getdatachange == true \wedge delPIT == true;$$
$$\#assert\ \ System()\ \ |= []!\ PIT\_Deletion\_Faking;$$

If a intruder intercepts an interest packet, it returns a faking data packet. When a router receives this faking packet, the router deletes corresponding PIT entries. Even though the router gets the legitimate data packet, it throws the data packet away. Using *always* operator [], we define this assertion to guarantee that routers are not vulnerable to such attacks.

Property 4: CS Caching Pollution

$$\#define\ \ CSCaching\_Pollution$$
$$getdatachange == true \wedge addccs == true;$$
$$\#assert\ \ System()\ \ |= []!\ CS\_Caching\_Pollution;$$

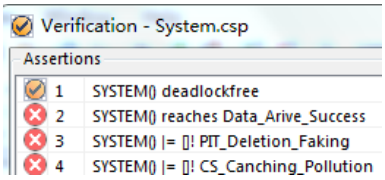

Fig. 6. Verification Result of Formalized Model

If a router receives a faking data packet, it may store the packet. When the router obtains an interest packet which contains the name of the faking packet, it returns the faking packet directly. This assertion is given to ensure that a router cannot store a faking data packet.

As shown in Fig.6, Property 1 (deadlock freedom) is valid. In other words, model cannot run into a deadlock state. Property 2 (data availability) is not satisfied for the system. This means that entities cannot get desired data with intruders intervention. Property 3 (PIT deletion faking) and Property 4 (CS caching pollution) are invalid. When a router obtains a faking data packet, it may delete its PIT entities and add the packet into its CS. Then the router can no longer obtain or provide a legal data packet which has the same name as the faking data packet.

B. Improvement

In order to ensure NDN-based IoV model to satisfy *Property 2*, *Property 3* and *Property 4*, we improve this model by using a method which is similar to Blockchain.

$$MSG_{data11} = \{msg_{data}.a.b.block, msg_{data}.a.b.trans \mid a, b \in Entity, block \in Block, trans \in Transcation\}$$

A new kind of message MSG_{data11} is defined to describe the transmission of blocks and transactions. After a producer creates a data packet, it constructs and transmits a transaction which includes some information about this data packet to an *RSUP*. Then the *RUSP* sends the transaction to a router. If the verification result of the transaction is positive, the router stores and forwards it. Meanwhile, the router produces a block which contains all transactions stored in it so far. Then the router stores the block and forwards it to other entities. After verifying this block, an entity adds it into its blockchain. Then the entity deletes duplicate transactions and forwards this block to others. Gradually all entities' blockchains become synchronous. When an entity obtains a data packets, its verifies the packet with its blockchain firstly as follows:

$$\begin{aligned} &Checkhash_i(BlockTable_i, dnhash, sighash, siginhash) = \\ &{}_{df} Intail\{check = fasle; x = 0; y = \#BlockTable_i; \\ &packethash = Hash(dnamehash, sighash, siginhash)\} \\ &\rightarrow (0 \le x < y)* \\ &\left(\begin{array}{l} translist = BlockTable_i[x].transcationlist; \\ \left(\begin{array}{l} (check = true) \lhd (\exists transaction \in translist \bullet \\ transaction.datanamehash == dnamehash \wedge \\ transaction.packethash == packethash) \rhd \\ (checkpacket = false) \end{array} \right) \end{array} \right); \end{aligned}$$

When the entity gets the data packet, it computes some hash values of this data packet. It queries its blockchain $BlockTable_i$ and gains some information about this data packet. It verifies the received data packet by comparing hash values obtained by calculation and results of querying. As we can see from Fig.7, the verification results of all properties are both valid for this model.

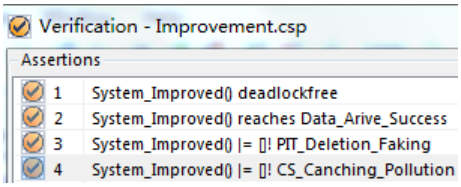

Fig. 7. Verification Result of Improved Model

V. CONCLUSION AND FUTURE WORK

NDN-based IoV is built by applying NDN into Internet of Vehicles. In this paper, we have formalized NDN-based IoV using CSP. Feeding the formalized model into PAT, we have verified four properties (deadlock freedom, data availability, PIT deletion faking and CS caching pollution). The last three properties are invalid. It means that the security of data has been compromised, once intruders appeared. In order to solve these problems, we improved the model with a method based on Blockchain. Then we verified the improved model. According to the verification results, the improved model can prevent intruders from invading the system.

In our future work, we will take the performance of NDN-based IoV model into consideration. To ensure the correctness and preciseness of the research results, formal methods will be used again in future work. Some related algorithms will be explored to improve the efficiency of this model.

Acknowledgements. This work was partly supported by National Key Research and Development Program of China (grant no. 2018YFB2101300), National Natural Science Foundation of China (grant no. 61872145), Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (grant no. ZF1213), the Fundamental Research Funds for the Central Universities of China and the Opening Project of Shanghai Trusted Industrial Control Platform (grant no. TICPSH202003007-ZC).

REFERENCES

[1] Juan Contreras-Castillo, Sherali Zeadally, Juan Antonio Guerrero Ibáñez: Internet of Vehicles: Architecture, Protocols, and Security. IEEE Internet of Things Journal 5(5): 3701-3709 (2018)
[2] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, Adaptive forwarding in named data networking, Computer Communication Review, vol. 42, no. 3, pp. 62C67, 2012
[3] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher,Börje Ohlman: A survey of information-centric networking. IEEE Communications Magazine 50(7): 26-36 (2012)
[4] Muhammad Tahir Abbas, Muhammad Afaq, Wang-Cheol Song: Road-Aware Estimation Model for Path Duration in Internet of Vehicles (IoV). Wireless Personal Communications 109(2): 715-738 (2019)
[5] Wei-Che Chien, Hung-Yen Weng, Chin-Feng Lai, Zhang Fan, Han-Chieh Chao, Ying Hu: A SFC-based access point switching mechanism for Software-Defined Wireless Network in IoV. Future Gener. Comput. Syst. 98: 577-585 (2019)
[6] Zhou Su, Yilong Hui, Qing Yang: The Next Generation Vehicular Networks: A Content-Centric Framework. IEEE Wireless Commun. 24(1): 60-66 (2017).
[7] Muktadir Chowdhury, Junaid Ahmed Khan, Lan Wang: Smart Forwarding in NDN VANET. ICN 2019: 153-154
[8] Eirini Kalogeiton, Domenico Iapello, Torsten Braun: A Geographical Aware Routing Protocol Using Directional Antennas for NDN-VANETs. LCN 2019: 133-136
[9] C. A. R. Hoare: Communicating Sequential Processes. Prentice-Hall 1985, ISBN 0-13-153271-5
[10] PAT, PAT: Process analysis toolkit. [Online]. Available: http://pat.comp.nus.edu.sg/