Algebraic Formalization and Verification of PKMv3 Protocol using Maude

Jia She, Xiaoran Zhu, Min Zhang(✉)
Shanghai Key Laboratory of Trustworthy Computing,
MoE International Joint Lab of Trustworthy Software, ECNU, 200062, China
melodyspencer@126.com, zhangmin@sei.ecnu.edu.cn

***Abstract*—PKMv3 is the third version of Privacy and Key Management protocol, which plays an important role by providing key distribution and security access control in IEEE802.16m, the standard of Worldwide Interoperability for Microwave Access. The protocol should be guaranteed safe in terms of confidentiality, authentication and integrity. In this paper, we develop an executable formal specification of PKMv3 in an algebraic language called Maude and verify safety properties of the protocol using state exploration and LTL model checking in Maude. Unlike existing approaches, we consider the behaviors of intruders and time feature in our verification and verify both safety properties and time-related properties of the protocol.**

I. Introduction

IEEE802.16m [4] is the newer generation of WiMAX, a communication standard for long-distance high-speed transmission of wireless data. In IEEE802.16m, PKMv3 protocol is defined for key management and used to control security access to the network by establishing encryption connections. Based on its first two generations, PKMv3 improve message hierarchical protection and its encryption algorithm. In addition, PKMv3 support Extensible Authentication Protocol (EAP) method and Cipher-based Message Authentication Code (CMAC), which largely reduce the possibility of attack.

The mobile WiMAX network faces more threats than traditional wireless networks. Security of the network must be guaranteed. Many efforts have been done to verify security properties of PVM protocol. In [8], BAN logic is used to analyze the key management protocols of the previous two generations, and the defects of one-way authentication in PKMv1 and the interspersed attacks in PKMv2 are pointed out. In [5], Scyther (an automated protocol testing tool) is used to implement the formal analysis of PKMv2 protocol to verify the defects of confidentiality, authenticity and integrity in the protocol. In [9], the PKMv2 protocol is described by the Casper protocol modeling tool and the output of the process communication is analyzed by FDR tool. It is found that the intruder can intercept the message and replay the attack. In [7], the security flaws of authentication and authorization have been in WiMAX, such as the possible DoS attack, and the defects of authentication and key space. Raju et. al. also use CasperFDR tool to model the process of transmitting plaintext messages between base station, mobile station and relay station in PKMv3 protocol, and get the attack model of stealing secret keys by analyzing outputs [6]. Zhu et al consider the time characteristics of PKMv3 protocol key lifecycle and model them using Promela language and use DT-Spin model checker to verify the protocol liveness, key period and message consistency [10].

In this work, we formalize PKMv3 protocol in an algebraic language called Maude [1] with considering both the behaviors of intruders and the time feature of the protocol. Six properties including three safety properties and three time-related properties are verified using the searching and LTL model checking functions provided by Maude. The verification results coincide with those in existing works [5], [7], [6], [10]. Besides that, we find the integrity vulnerability of PKMv3 by verification and propose a solution on the basis of the verification result.

II. PKMv3 Protocol

PKMv3 protocol provides security distribution mechanism of key material between the server called *base station* (BS) and the client called *subscriber station* (SS). More precisely, it generates traffic encryption keys (TEK) for stations to encrypt the messages they exchange. The generation of TEK consists of three procedures which are called *AK authentication*, *SA negotiation* and *TEK transmission*. Each procedure is achieved by exchanging key information between SS and BS. After a TEK is generated, secret keys i.e., *AK* and *TEK*, may expire because their lifetime is limited. After they expire reauthorization is needed. Figure 1 depicts the whole process of key generation and reauthorization. In this section, we give the details of three procedures and reauthorization.

A. AK Authentication

By this phase, SS and BS establish mutual authentication and SS receives a Pairwise Master Key (PMK) sent by BS. PMK is used by BS and SS to generate Authorization

This material is based upon work supported by National Natural Science Foundation of China (NSFC) project: No. 61502171.
DOI reference number: 10.18293/SEKE2017-061

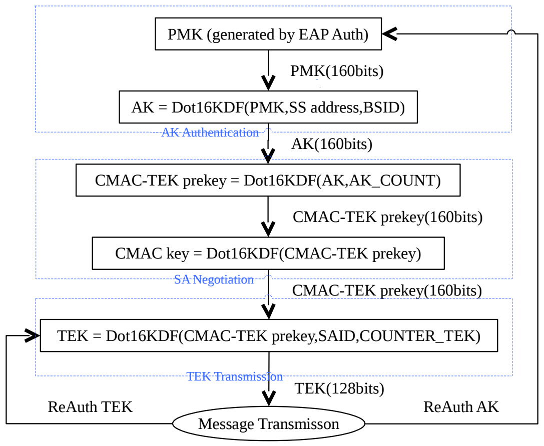
PMK (generated by EAP Auth)
PMK(160bits)
AK = Dot16KDF(PMK,SS address,BSID)
AK Authentication
AK(160bits)
CMAC-TEK prekey = Dot16KDF(AK,AK_COUNT)
CMAC-TEK prekey(160bits)
CMAC key = Dot16KDF(CMAC-TEK prekey)
SA Negotiation
CMAC-TEK prekey(160bits)
TEK = Dot16KDF(CMAC-TEK prekey,SAID,COUNTER_TEK)
TEK Transmission
TEK(128bits)
ReAuth TEK
Message Transmisson
ReAuth AK

Fig. 1. The process of key generation and reauthorization

Key (AK), which are used in the subsequent process. This phase consists of the following three message exchanges:

Msg1.	$SS \rightarrow BS$:	$Cert_{SS}\|N_S\|Capabilities\|SIG_{SS}$
Msg2.	$BS \rightarrow SS$:	$Cert_{BS}\|N_S\|N_B\|ENC_{SS}(PMK)\|$ $PMK\_SN\|SIG_{BS}$
Msg3.	$SS \rightarrow BS$:	$N_B\|SS\_Address\|CheckSum$

Firstly, SS sends BS an auth-request message, which is encrypted with the private key of SS as a digital signature, SIG_{SS}. The message includes a digital certificate of SS, a nonce N_S generated by SS and the encryption capabilities that SS supports. On receiving the message, BS decrypts it with the public key of SS, and then sends an auth-response message encrypted with the private key of BS. This message includes a certificate of BS, the nonce N_S received from SS, a nonce N_B generated by BS, a PMK encrypted with SS's public key, and the sequence number PMK_{SN} of PMK. SS decrypts the message and authenticates the legitimacy of the BS's digital certificate. Finally, SS sends BS an auth-success message containing the address of SS, with which an AK can be generated.

B. SA Negotiation

By the phase, SS and BS negotiate the security association (SA) through three message exchanges. SA is an information set that is required for secure communication. The three message exchanges are shown below:

Msg4.	$BS \rightarrow SS$:	$N_B\|PMK\_SN\|AKID\|CMAC$
Msg5.	$SS \rightarrow BS$:	$N_B\|N_SPMK\_SN\|AKID\|SNP\|$ $Security\ Capabilities\|CMAC$
Msg6.	$BS \rightarrow SS$:	$N_S\|PMK\_SN\|SAID\|SNP\|CMAC$

Before message exchange, SS and BS generate CMAC keys using AK separately. CMAC keys are used to calculate CMAC digest. BS firstly sends SS a key agreement message, including nonce N_B generated in this round, PMK_{SN}, and identifier $AKID$ of AK. In particular, BS generates a CMAC digest with the three arguments and sends along with the message. On receiving the message, SS calculate a CMAC digest using its CMAC key and compares it with the received one. If they are equal, SS sends BS the second key agreement message, including N_B, PMK_{SN}, AKID, security negotiation parameter (SNP), and newly generated nonce N_S and CMAC digest. Once BS checks the validity of the CMAC digest, it sends SS the identifier of SA (SAID) and other information. SAID is used to identify different security levels of security association. At this point, SS and BS have negotiated the necessary information required for secure communication.

C. TEK Transmission

By this phase, SS and BS generates TEK after three message exchanges, as shown below.

Msg7.	$SS \rightarrow BS$:	$SAID\|PMK\_SN\|TEK\ Refresh$ $Flag\|CMAC$
Msg8a.	$BS \rightarrow SS$:	$SAID\|PMK\_SN\|Counter\_TEK$ $EKS\|CMAC$
Msg8b.	$BS \rightarrow SS$:	$SAID\|Wrong\_Code\|CMAC$

SS first sends BS a TEK-Request message, including SAID, PMK_SN, TEK refresh flag and CMAC digest. After checking the correctness of CMAC digest and SAID, BS sends SS a request confirm message TEK-Reply, including SAID, PMK_SN, the counter of TEK (Counter_TEK), the encryption key sequence (EKS), and the CMAC digest. Counter_TEK is used to generate TEK together with SAID and the CMAC-TEK prekey that is derived by AK, and EKS is used to distinguish consecutive TEKs. If the TEK request fails, BS sends SS a request invalid message TEK-Invalid, including SAID, error code and the CMAC digest. In this case, SS needs to re-send TEK-Request message to BS.

D. Reauthorization

After the whole process of the three procedures, BS and SS establish a security connection for the transmission of communication messages encrypted by TEK. As mentioned above, because both AK and TEK have limited lifetime BS and SS have to be re-authorized to keep liveness and safety when the lifetime of the keys expire.

When AK's lifetime expires, SS needs to send an Auth-request message to BS to request a new AK. It ends the current process and restart from AK authentication, as shown in Figure 1. Because TEK's lifetime is embedded in AK's, when TEK becomes invalid but it corresponding AK is still alive, SS only needs to send TEK-request message to BS to start the process of TEK transmission.

III. Maude Nutshell

Maude is an algebraic specification language and also an efficient rewriting engine [1]. The underlying rewriting logic of Maude is a logic of concurrent changes well-suited to formalize states and concurrent computations [2].

A. Formalization in Maude

A system is formalized in Maude as a rewrite theory i.e., $(\Sigma, E \cup A, R)$, consisting of a signature Σ, a collection E of (possibly conditional) equations and memberships defined on Σ, a collection A of equational attributes, and a set R of rewrite rules. The two-tuple $(\Sigma, E \cup A)$ is called a membership equational theory, which specifies the "statics" of a system, i.e., the algebraic structure of the set of system states, and the rules in R specify the "dynamics" of the system, i.e., all the possible transitions that the system can perform. In Maude, a system state is usually represented as an algebraic inductive structure which can be a tuple, a *soup* of components, or even an object. Thanks to the inductive representation, Maude can naturally specify both finite-state and infinite-state systems. Transitions among the states are formalized by rewrite rules. Given two states v_1 and v_2, let t_{v_1} and t_{v_2} be their corresponding terms. There is one-step transition from v_1 to v_2 if there is a rewrite rule r such that t_{v_1} can be rewritten into t_{v_2} by applying r once.

Finally, we briefly summarize the syntax of Maude (see the work [1] for more details). Sorts and subsort relations are declared by the keywords `sorts` and `subsort`, respectively. Operators are declared with the `op` keyword in the form: `op` f `:` $s_1 \ldots s_n$`->` s, where $s_i (i = 1, \ldots, n)$ and s are sorts. Maude allows for user-defined mixfix operators and uses underbars in operators to indicate each of the argument positions. An equation in Maude is declared in the form of `eq` t `=` t', where t and t' are two terms of the same sort. An equation can be conditional, which is declared by the keyword `ceq` and ended with the keyword `if`, followed by a conjunction of conditions. A rewrite rule in Maude is declared in the form of `rl [label]:` t `=>` t'. A conditional rewrite rule is declared by the keywords `crl` and `if` with a conjunction of rewrite conditions.

B. Formal analysis in Maude

Maude programs are executable. For this feature Maude provides multiple formal analysis methods, mainly including simulation, reachability analysis, and model checking to formally analyze systems.

Simulation is achieved using Maude's `rewrite` command, which repeatedly applies the rewrite rules to transform a given term step by step. The transformation process simulates one behavior of the specified system.

Maude provides a `search` function to explore the reachable state space of specified systems. The `search` function can be used to verify invariant properties of systems. An invariant property states that something bad should never happen. We verify an invariant property by specifying the negation of the property as the condition in `search` command and using Maude to find if there are solutions. A solution can be interpreted as a counterexample of the property. If the reachable state space is infinite or finite but too large to be explored due to time or memory limitations, a bound on the searching depth is needed, and thereafter properties are partially verified in such cases.

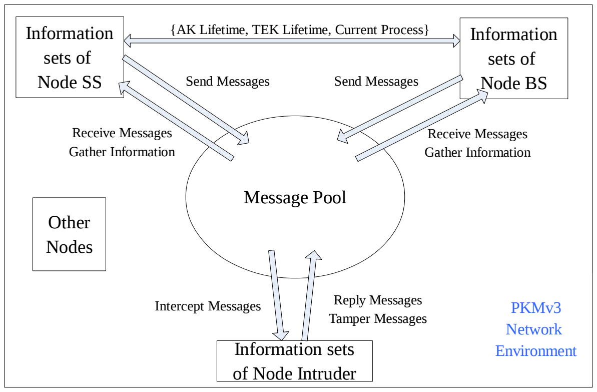
Information sets of Node SS
{AK Lifetime, TEK Lifetime, Current Process}
Information sets of Node BS
Send Messages
Send Messages
Receive Messages Gather Information
Receive Messages Gather Information
Message Pool
Other Nodes
Intercept Messages
Reply Messages Tamper Messages
PKMv3 Network Environment
Information sets of Node Intruder

Fig. 2. Configuration of PKMv3 network

Maude also provides an efficient LTL model checker [3] to verify LTL properties of systems. Given a Maude specification and a set of atomic propositions, the model checker takes an initial state and an LTL formula and returns true or a counterexample. A condition of doing model checking in Maude is that the set of states that are reachable from the given initial state must be finite.

IV. Formal Modeling of PKMv3 Protocol

This paper treats PKMv3 protocol and its environment as a dynamic system. In the network, there are stations e.g., SS, BS and other intruder, and message transmissions, as depicted in Figure 2. We assume the correctness of all the encryption algorithms used in the protocol and one BS communicates with only one SS at a time.

A. Definition of basic data types

In Maude we define abstract data types to formalize the objects in the network such as stations, messages, lifetime and nonce. Due to space limitation, we only explain the main data types as examples.

We declare a sort `Station` for stations and three subsorts `Ss`, `Bs` and `Intruder` of it for BS, SS, and intruder respectively. Three constructors `ss`, `bs` and `intruder` are defined to construct stations for SS, BS and intruder with a string as their argument.

```
sorts Station Ss Bs Intruder .
subsorts Ss Bs Intruder < Station .
op ss : String -> Ss [ctor] .
op bs : String -> Bs [ctor] .
op intru : String -> Intruder [ctor] .
```

We use random numbers to model nonce, as formalized by the following Maude codes. A random number can be a seed or a successor of an existing one. A nonce consists of a station which generates it, a station to which the nonce is to be sent, and a random number.

```
sorts Rand Nonce .
ops seed seed0 seed1 : -> Rand [ctor] .
op  next : Rand         -> Rand [ctor] .
op nonce : Station Station Rand -> Nonce [ctor] .
```

Keys and certificates of stations are formalized likewise. As defined below, constructor `pubkey` (resp. `prikey`) constructs a public (resp. secret) key, `macaddr` constructs a MAC address of a station, and `cert` constructs a certificate with a station, a MAC address and a public key.

```
sorts Pubkey Prikey MacAddr Cert .
op pubkey  : Station -> Pubkey  [ctor] .
op prikey  : Station -> Prikey  [ctor] .
op macaddr : Station -> MacAddr [ctor] .
op cert : Station MacAddr Pubkey -> Cert [ctor] .
```

B. *Formalization of system states*

A system state of PKMv3 protocol consists of a set of nodes, a pool of messages and the connection time between SS and BS.

A message is formalized as a triple, consisting of source, destination and a collection of contents such as certificates, keys and nonce. Message and contents are defined in Maude as follows:

```
sorts Content Contents Message .
subsorts Key Cert Nonce ... < Content < Contents.
op _,_ : Content Content -> Content [assoc comm].
op msgnil : -> Message [ctor] .
op from_to_send_: Station Station Content ->
    Message .
```

The mixfix operator `_,_` represent the union of two collection of contents, `msgnil` represents an empty message, and `from_to_send_` constructs a message with a source station, a destination station and a collection of contents.

A node consists of a station and a collection of contents. The duration time of a connection consists of remaining lifetimes of AK and TEK and a flag to indicate to which process the communication is proceeding. We use natural numbers to represent the lifetime of AK and TEK, and formalize duration time as a triple. The Maude definition of nodes and connection time is given below:

```
sorts Node LTime Process ConnTime .
subsort Nat < LTime .
op  node[_]:_ : Station Content -> Node .
ops pak psa ptek pmt : -> Process .
op  {_,_,_} : LTime LTime Process -> ConnTime .
```

Constructors `pak`, `psa`, `ptek` and `pmt` of sort `Process` respectively represent the processes AK authorization, SA negotiation, TEK exchange and message transmission. Lifetime of secret keys is defined as sort `LTime` with predefined sort `Nat`. The connection time is defined by sort `ConnTime` and represented using the constructor `{_,_,_}`. It is worth mentioning that before AK (resp. TEK) is generated, the time in the triple represents the AK (resp. TEK) grace time, i.e., the time that is allowed before AK (resp. TEK) is generated [10].

As defined below, sort `States` is declared for system states, and `Node`, `Message` and `ConnTime` are declared as its subsorts. A state is composed of nodes, messages and connection time by the constructor `__`.

```
sorts States .
subsorts Message Node ConnTime < States .
op __ : States States -> States [assoc comm] .
```

TABLE I
VARIABLES USED IN THE SPECIFICATION

Variables	Sorts	Descriptions
A, B, C	Stations	SS, BS, Intruder
R1, R2, R0	Rand	Random number
NC_X(X=A,B,C)	Nonce	Nonce sent by Station
CERT_X(X=A,B,C)	Cert	Certificate of Station
PMKSN	Pmksn	Key sequence of PMK
AKID	AkId	Identifier of AK
AKCOUNT	AkCount	Counter of AK
MSG2	AuthResponse	Ak-response message
MSG5	SAResponse	SA-response message
CMAC1,CMAC2	Cmac	CMAC digest
CMACKEY	Cmackey	Cmac key
SAID	SaId	Identifier of SA
FLAG	TRFlag	TEK refresh flag
C_TEK	CounterTek	Counter of TEK
AKLT,TEKLT	LTime	Life time of AK,TEK
PS	Process	Process of PKMv3
CS1,CS2	Content	Message contents

C. *Formalization of message exchanges*

We formalize the eight message exchanges explained in Section 2 by rewrite rules in Maude. There are totally 22 rewrite rules defined. We take three of them as examples. Table 1 shows all the variables that are used in rewrite rules as well as their data types and descriptions.

The first one specifies the behavior of sending an auth-request message from SS to BS for AK authentication.

```
rl [SendAuthRequestMsgReal] :(node[A]: R0,CERT_A)
(node[B]: R1,CERT_B) (msgnil) {AKLT,TEKLT,pak} =>
(node[A]: next(R0),CERT_A,nonce(A,B,R0))
(node[B]: R1,CERT_B)
(from A to B send sencrypt(authrequest(CERT_A,
    nonce(A,B,R0),capa(A)),prikey(A)))
{sd(AKLT,tt),TEKLT,pak} .
```

The term `pak` in the rule indicates that the behavior occurs at the AK Authentication process. The right-hand side of the rule says after the behavior a message is put into the network. The message consists of a certificate `CERT_A`, a nonce generated by station `A` and the encryption capabilities `capa(A)` of station `A`. It is encrypted by the private key `prikey(A)` of `A` and sent to `B`. After sending the message, AK grace time `AKLT` is decreased by a fixed number `tt` of time units which can be customized initially.

The second rewrite rule describes the behavior of BS receiving SA-response message from SS for SA negotiation process. The condition of the behavior is that there is an SA-response message sent to BS in the network. In the message, the AKID and nonce N_B should be same as the ones that `B` holds, and attached CMAC digest `CMAC2` is the same as the one calculated using the information in `MSG5` and `CMACKEY`. The rewrite rule is defined as follows:

```
crl [RecvSAResponseReal] :
(node[B]:NC_B,AKID,CMACKEY,CS1)
(from A to B send (MSG5,CMAC2)){AKLT,TEKLT,ps}=>
(node[B]:NC_B,AKID,CMACKEY,getsc(MSG5),getnon52(
MSG5),getsnp5(MSG5),CS1)(msgnil){AKLT,TEKLT,psa}
if getakid5(MSG5) == AKID /\
   equals(getnon51(MSG5),NC_B) == true /\
   equalm(cmac(MSG5,CMACKEY),CMAC2) .
```

The last rewrite rule specifies key reauthorization. Key reauthorization occurs when lifetime of AK `AKLT` is no less than a predefined value `tt` of time units and TEK is less than `tt` in message transmission procedure, as defined by the condition part of the following rewrite rule.

```
crl [CircleTEKMsgReal] :
(node[A]:PMKSN,CMACKEY,SAID,FLAG,TEK,C_TEK,CS1)
(node[B]:PMKSN,CMACKEY,SAID,FLAG,TEK,C_TEK,CS2)
{AKLT,TEKLT,PS} (msgnil) =>
(node[A]: PMKSN,CMACKEY,SAID,flag(1),CS1)
(node[B]: PMKSN,CMACKEY,SAID,CS2 )
(from A to B send(tekrequest(SAID,PMKSN,flag(0)),
cmac(tekrequest(SAID,PMKSN,flag(1)),CMACKEY)))
{sd(AKLT,tt),sd(tgt,tt),ptek}
if AKLT >= tt /\ TEKLT < tt .
```

The right-hand side of the rule means that by reauthorization `A` and `B` delete all the data gathered in TEK exchange such as `FLAG`, `TEK` and `C_TEK` first, and `A` generates new TEK refresh flag `flag(1)` and sends `B` a TEK-request message containing `SAID` received in SA negotiation, `PMK_SN` and TEK refresh flag and CMAC digest.

D. Formalization of intruders

Intruder is a part of the network environment. We assume intruders have three capabilities: (1) participating in the protocol communication with the same capacity of normal station; (2) eavesdropping on the messages in the network; and (3) replaying received messages and tampering with message contents. Intruders may intervene in each procedure of the protocol and hinder the normal communication of SS and BS.

We formalize the behavior of intruder as rewrite rules in the same way as we formalize those of normal stations. Due to space limitation, we only discuss two as examples.

The following rule formalizes the process of intruder `C` intercepting auth-response message in AK authentication. When `B` sends `A` auth-response message `MSG2`, `C` can intercept this message from message pool and encrypt its digital signature using the public key of `B`.

```
crl [RecvAuthResponseMsgFake] :
(node[C]: R2,CERT_C,CERT_A,NC_A)
(from B to A send sencrypt(MSG2,PRI))  =>
(node[C]: R2,CERT_C,CERT_A,NC_A,getcert2(MSG2),
    getnon22(MSG2),decrypt(getcipher(MSG2),prikey
    (C)),getpmksn2(MSG2)) (msgnil)
if equals(getnon21(MSG2),NC_A) == true /\
   getpub(getcert2(sdecrypt(sencrypt(MSG2,PRI_B),
       pubkey(B)))) == (B) .
```

The condition says that if the certificate of `B` and N_B are correct, intruder `C` retrieves corresponding data from `MSG2` and tries to decrypt PMK with its own secret key.

The second rewrite rule specifies the behavior of intruder `C` sending fake auth-confirm message in AK authentication process. After eavesdropping communication between SS and BS, intruder has gathered necessary information including nonce N_B, PMK_SN and address of station `A`. Hence, `C` can forge auth-confirm message with N_B and fake address `ssaddr(C)` and then disguise that SS sends BS this message.

```
crl [SendAuthConfirmMsgChange] :
(node[B]: AKCOUNT,PMKSN,CS1)
(node[C]: NC_B,PMKSN,ssaddr(A),CS2) (msgnil)
{AKLT,TEKLT,PS}  =>
(node[B]: AKCOUNT,PMKSN,CS1)
(node[C]: NC_B,PMKSN,ssaddr(A),CS2)
(from A to B send(authconfirm(NC_B,ssaddr(C)),
    checksum(authconfirm(NC_B,ssaddr(C)),iv)))
{sd(AKLT,tt),TEKLT,PS}
if AKLT >= tt /\ not(ssaddr(C) inc CS1) .
```

The condition says that the message can be sent if the lifetime of AK is no less than transmission time `tt` and `B` has not received auth-confirm message yet. In addition, intruder can reply auth-confirm message to `B` as the identity of station `A`, without changing original contents of auth-confirm message which sent by `A`.

V. Formal Verification of PKMv3's Properties

With the specification, we verify both safety properties and time-related properties by the searching function and Maude LTL model checker.

A. Definition of initial states

An initial state should be provided for verification. We assume that in the initial state there is a mobile station, a base station and an intruder, which have their own digital certificates. The message pool is empty. Message transmission time `tt` is set 4, and the grace time of AK `agt` and TEK `tgt` 60. We use `init` to denote the initial state of PKMv3 network, which is defined as follows:

```
op init : -> States .
eq init = (msgnil) {agt,tgt,pak} (node[ss("a")]:
    seed,cert(ss("a"),macaddr(ss("a")),pubkey( ss
    ("a"))))(node[bs("b")]: seed1,cert(bs("b"),
    macaddr(bs("b")),pubkey(bs("b"))))(node[intru
    ("c")]: seed0,cert(intru("c"),macaddr(intru("
    c")),pubkey(intru("c")))) .
```

B. Formalization of properties

We consider six properties of PKMv3, including three safety properties and three time-related properties.

The three safety properties are called *confidentiality*, *authentication* and *integrity*. By confidentiality it means that intruders can never obtain PMKs that are used by two normal stations. Without PMK intruders cannot calculate AK and participate in the subsequent process even if it has gathered other important key material. Authentication is another important property of the protocol. That is, even if there are attacks from intruders in the network, normal stations should still be able to verify each other's identity and carry on communication. Integrity property says that messages transmitted between the honest subjects should not occur transmission error or be tampered with, or at least wrong message data can be detected. In Maude, safety properties can be verified using `search` command and do not need to specify separately.

The three time-related properties are *succession*, *reauthorization*, and *key freshness*. Succession means that the four procedures should take place in turn. Reauthorization says that when AK and TEK expire, authorization needs to be re-established immediately. Key freshness says that transmitting those messages with expired AK and TEK is not allowed. The three properties can be specified as the following LTL formulas in Maude.

```
(eap-auth U nego-sa) /\ (<> nego-sa U exch-tek)
    /\ (<> exch-tek U msg-trans)) .
(<> msg-trans U exch-tek) /\ (<> msg-trans U eap-
    auth) ) .
[](~ inva-msg ) .
```

In the above formula, eap-auth, nego-sa, msg-trans and exch-tek are atomic propositions which are true in those states where the process tag is respectively pak, psa, ptek and pmt, and inva-msg is an atomic proposition which is true in those states where the lifetime of current AK or TEK is less than zero. The symbols U, <> and [] represent the temporal connectors **U**, **F** and **G** in LTL.

C. *Formal verification and result analysis*

We use searching function to verify the safety properties and Maude LTL model checker to verify the time-related properties of the protocol. For instance, the following search command is used to find a state where an intruder obtain a PMK that is not supposed to belong to it.

```
search init =>* (S:States) (node[intru("c")]:
pmk(ss("a"),bs("b"),nonce(ss("a"),bs("b"),seed),
nonce(bs("b"),ss("a"),seed1),algo),C:Contents).
```

Maude returns no solution, which means that the confidentiality holds. The other two safety properties can be verified likewise by searching. For model checking, one only needs to call modelCheck(init,F) with init the predefined initial state and F an LTL formula.

The verification results of the six properties are shown in Table II. For time-related properties, the succession and reauthorization of PKMv3 protocol can be satisfied, which means that the procedures of PKMv3 and period lifetime of secret keys are correct. Meanwhile, messages are always transmitted with valid secret keys which guarantee key freshness. For the safety properties, confidentiality and authentication can also be satisfied by our Maude model. The intruder can never obtain the PMKs and other secret keys that are not supposed to belong to it. SS and BS can always authenticate each other's identity. The verification results coincide with those in the work [5], [7], [6], [10].

Another finding in our verification is that the integrity of messages can not be guaranteed. Maude returns a case that BS receives a fake auth-confirm message with incorrect address of SS, and then generates invalid AK. The reason is that auth-confirm message is generated using common checksum algorithm, but intruder can tamper with this message and generate corresponding checksum. In this case, BS acknowledge that the received message does not have transmission error, but can not verify whether the message is tampered with or not by intruder.

The integrity vulnerability of PKMv3 can be improved by using AK to generate the checksum of auth-confirm

TABLE II
VERIFICATION RESULTS OF THE SIX PROPERTIES

Property	Method	Rewrite	Time	Result
Succession	Model checking	388	2ms	√
Reauthorization	Model checking	264	4ms	√
Key freshness	Model checking	1199	2ms	√
Confidentiality	Searching	1843	44ms	√
Authentication	Searching	1843	112ms	√
Integrity	Searching	420	4ms	×

message. More precisely, the modified message is described as: $SS \rightarrow BS$: $N_B|SS\_Address|AK(N_B|SS\_Address)$. BS generates AK with received SS_Address, and then compares the calculated checksum with the one that received. Verification results after the refinement show that the integrity of message transmission is satisfied.

VI. CONCLUSION

We have presented an algebraic approach to formal modeling of PKMv3 protocol and verification of its six safety properties and time-related properties using Maude. In our model, we consider both the behaviors of intruders and the time feature of the protocol. Verification results show that our model of PKMv3 can satisfy the succession of whole process, re-authorization, validation secret keys and authentication, which coincide with the results of other existing works. We also found that the integrity of messages can not be guaranteed due to auth-confirm message may encounter man-in-the-middle attacks. We proposed a solution to the problem and verified its validity.

REFERENCES

[1] Clavel, M., et al.: All about Maude, LNCS, vol. 4350. Springer (2007)
[2] Clavel, M., Durán, F., Eker, S., et al.: Maude: specification and programming in rewriting logic. Theor. Comput. Sci. 285(2), 187–243 (2002)
[3] Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker. In: 4th WRLA, ENTCS 71. pp. 162–187. Elsevier (2002)
[4] IEEE, B.E.: Ieee standard for local and metropolitan area networks part 16: Air interface for broadband wireless access systems amendment 3: Advanced air interface pp. 1–1112 (2011)
[5] Kahya, N., Ghoualmi, N., Lafourcade, P.: Formal analysis of PKM using scyther tool. In: International Conference on Information Technology and E-Services. pp. 1–6 (2012)
[6] Raju, K.V.K.: Formal Verification of IEEE802.16m PKMv3 Protocol Using CasperFDR. In: Information and Communication Technologies - International Conference. pp. 590–595 (2010)
[7] Sikkens, B.: Security issues and proposed solutions concerning authentication and authorization for WiMAX (IEEE 802.16e). In: Proc. of 8th Conference on IT Enschede University of Twente (2008)
[8] Xu, S., Huang, C.T.: Attacks on PKM Protocols of IEEE 802.16 and Its Later Versions. In: International Symposium on Wireless Communication Systems. pp. 185–189 (2006)
[9] Xu, S., Huang, C.T., Matthews, M.M.: Modeling and analysis of IEEE 802.16 PKM Protocols using CasperFDR. In: IEEE International Symposium on Wireless Communication Systems. pp. 653–657 (2008)
[10] Zhu, X., Xu, Y., Guo, J., Wu, X.: Formal Verification of PKMv3 Protocol Using DT-Spin. In: International Symposium on Theoretical Aspects of Software Engineering. pp. 71–78 (2015)