Using SPIN to Detect Vulnerabilities in the AACS Drive-Host Authentication Protocol

Wei Wang[1] and Dongyao Ji[2]

The State Key Laboratory of Information Security,
Graduate University of Chinese Academy of Science,
NO.19 Yuquan Road, Shijingshan District, Beijing, 100049, P.R.China
bessie19831109@163.com

Abstract. In this paper, we use SPIN, a model checker for LTL, to detect vulnerabilities in the AACS drive-host authentication protocol. Before the detection, we propose a variant of the Dolev-Yao attacker model [4] and incorporate the synthesis and analysis rules [7] to formalize the protocol and the intruder capabilities. During the detection, we check the authenticity of the protocol and identify a few weaknesses. Besides, we propose a novel collusion attack that seriously threaten the security of the protocol, and build a corresponding LTL formula. Based on the formula, SPIN detects a few relevant attack instances in the original scheme of the authentication protocol and a modified scheme advanced in [5].

Key words: AACS, SPIN, Model Checker, LTL, Authenticity, Collusion Attack.

1 Introduction

Nowadays, in the field of protocol verification, the formal verification techniques appear to be a popular method for analyzing the vulnerabilities of protocols. There are two major approaches: theorem-proving and model-checking. Compared with theorem-proving, model-checking seems to be more suitable to detect errors and find corresponding attack modes of the target protocols [3].

So far, some researchers have developed specific model checkers for particular properties verification, whereas others have shown the ability of the general purpose tools such as FDR and SMV to achieve the same purpose. In this paper, we would like to implement our protocol verification using the general purpose tool of SPIN, which is one of the most powerful general purpose model checkers. Until now, some researchers have already shown how it can be used to check the security properties such as secrecy and authenticity, and successfully found the known attack in the Needham-Schroeder Public Key Authentication Protocol [2, 8].

In this paper, we will use SPIN to verify the AACS drive-host authentication protocol. The Advanced Access Content System (AACS) is a content distribution system for recordable and pre-recorded media. This system consists of three entities: a drive, PC host and the AACS protected optical media. The AACS drive-host authentication protocol, a part of the AACS protection scheme, plays the role of practicing the mutual authentication between the PC host and the drive and letting them negotiate a shared key which is used for message authentication in the subsequent interaction between the PC host and the drive [1].

In the whole process of verifying the AACS drive-host authentication protocol, we not only make use of the technique of model-checking, but also adopt the method of static analysis. And according to the variant of the Dolev-Yao attacker model [4] advanced by us in order to match the specific scheme of the target protocol, we make the formalization of the protocol and the intruder's behavior. Besides checking the property of authenticity, we also define a novel collusion attack which poses a threat to the security of the AACS protection scheme. Through the verification, we have discovered several relevant attack instances in the original AACS drive-host authentication scheme. In addition, a modified AACS drive-host authentication scheme, proposed in [5], also reveals its vulnerability to the novel collusion attack in our further verification.

This paper is organized as follows. In section 2, we construct the formal model of the target protocol, which includes the description of a variant of Dolev-Yao attacker model. In section 3, we briefly describe the general process of verifying the target protocol using SPIN. In section 4 and section 5, we present the process of checking the authenticity and verifying the feasibility of the newly-defined collusion attack in SPIN.

2 A Formal Model for Security Protocols

In this section, we present a formal model of the AACS drive-host authentication protocol and describe a variant of Dolev-Yao attacker model [4]. Besides, we have made some modifications of the semantics of security protocols built in [9] to simplify the process of modeling the target protocol. More detailed description can be found in [10].

First, we would like to discuss the occasion of generating a fresh nonce in the AACS drive-host authentication scheme: when finish one session no matter whether it is succeed or not, both the host and the drive would call the random number generator to acquire a fresh nonce used in a next session. But in this paper, our concern is only about a single session. Therefore, for the purpose of simplifying the model of the AACS drive-host authentication protocol, we define nonces as constants.

Next, we describe the definitions of several necessary terms used in the following discussion. We start with the set of *agents* – Ag, which includes the *intruder* I and other agents who are called *honest agents*. K, denotes the set of *private keys* which are owned by corresponding agents, and we use k_p to denote the *private key* belongs to agent p. $Cert$, denotes the set of *certificates* which represent the identities of corresponding agents, and we use $Cert_p$ to denote the *certificate* of p. N, denotes the set of *nonces*. P, the set of *parameters*, is used for some particular purposes. $\mathcal{T}_0$, the set of *basic terms*, is defined to be $Ag \cup K \cup Cert \cup N \cup P$. $\mathcal{T}$, the set of *information terms*, is defined to be:

$$\mathcal{T} ::= m \mid (t, t') \mid \{t\}_k \quad . \tag{1}$$

where m ranges over $\mathcal{T}_0$, k ranges over K, t and t' range over $\mathcal{T}$; and (t, t') denotes the concatenation of t and t', and $\{t\}_k$ denotes using k to encrypt the term t.

In addition, we define a set of actions:

$$\Sigma = \{A!B{:}t, A?B{:}t \mid A, B \in Ag, A \neq B, t \in \mathcal{T}\} \quad . \tag{2}$$

As we can see, there are two actions which are denoted as "Send" and "Receive":

- Send: $A!B{:}t$, A is the sender, B is the intended receiver, and t is the message.
- Receive: $A?B{:}t$, A is the receiver, B is the purported sender, and t is the message.

Definition 1. *A* protocol *is a pair* $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$*, where* $\mathsf{C} \subseteq \mathcal{T}_0$ *is the set of constants of* Pr*, and* R *is the set of roles of* Pr.

Definition 2. *A* sequent *is of the form* $T \vdash t$ *where* $T \subseteq \mathcal{T}$ *and* $t \in \mathcal{T}$*. An* analz*-proof (*synth*-proof)* π *of* $T \vdash t$ *is an inverted tree whose nodes are labeled by sequents and connected by one of the* analz*-rules (*synth*-rules) in Fig. 1, whose root is labeled* $T \vdash t$*, and whose leaves are labeled by instances of the* Ax_a *rule (*Ax_s *rule). For a set of terms* T*,* $\mathsf{analz}(T)$ *(*$\mathsf{synth}(T)$*) is the set of terms* t *such that there is an* analz*-proof (a* synth*-proof) of* $T \vdash t$*. For ease of notation,* $\mathsf{synth}(\mathsf{analz}(T))$ *is denoted by* $\overline{T}$.

The definitions of analz and synth are due to [7].

$$\frac{}{T \cup \{t\} \vdash t}\ \mathsf{Ax_a} \qquad\qquad \frac{}{T \cup \{t\} \vdash t}\ \mathsf{Ax_s}$$

$$\frac{T \vdash (t_1, t_2)}{T \vdash t_i}\ \mathsf{split}_i\ (i{=}1, 2) \qquad\qquad \frac{T \vdash t_1 \quad T \vdash t_2}{T \vdash (t_1, t_2)}\ \mathsf{pair}$$

$$\frac{T \vdash \{t\}_{k_p} \quad T \vdash Cert_p \quad T \vdash t}{T \vdash true}\ \mathsf{verify} \qquad\qquad \frac{T \vdash t \quad T \vdash k_p}{T \vdash \{t\}_{k_p}}\ \mathsf{sign}$$

analz-rules synth-rules

Fig. 1. The Modified Analz and Synth Rules

As we can see, Fig. 1 shows the modified analz and synth rules, which are based on the variant of the Dolev-Yao model [4], consist of split, pair, verify and sign. Specifically, the verify rule has its practical significance: if one knows three items – the signature $\{t\}_{k_p}$ (denoting using k_p, the private key of p, to sign the term t), $Cert_p$ (the certificate of p), and the term t, it can confirm the validity of the signature, which means that the signature $\{t\}_{k_p}$ would be successfully decrypted by the public key extracted from $Cert_p$, and the decrypted term is the same with t.

Definition 3. *An* information state *s is a tuple* $(s_A)_{A \in Ag}$ *where* $s_A \subseteq \mathcal{T}$ *for each agent* A*. The notions of an action enabled at a state and update of a state on an action are defined as follows:*

- $A!B{:}t$ *is* enabled *at* s *iff* $t \in \overline{s_A}$.
- $A?B{:}t$ *is* enabled *at* s *iff* $t \in \overline{s_I}$.
- $update(s, A!B{:}t) \overset{\text{def}}{=} s'$ *where* $s'_A = s_A$*,* $s'_I = s_I \cup \{t\}$*, and for all agents* C *distinct from* A *and* I*,* $s'_C = s_C$.
- $update(s, A?B{:}t) \overset{\text{def}}{=} s'$ *where* $s'_A = s_A \cup \{t\}$ *and for all agents* C *distinct from* A*,* $s'_C = s_C$.

3 Protocol Verification Using Spin

Spin is designed to validate the logical consistency of concurrent and distributed systems, such as data communications protocols, and trace the logical design errors [11]. By constructing a LTL formula of a desired property and simulating a correct model of the target protocol, one could easily carry out the verification on SPIN; and when detecting a violation of the target property, SPIN could provide the counterexample run. In this fragment, we will discuss the model construction process, which amounts to two steps: the formalization of the protocol and the formalization of the intruder's behavior.

3.1 Formalization of the Protocol

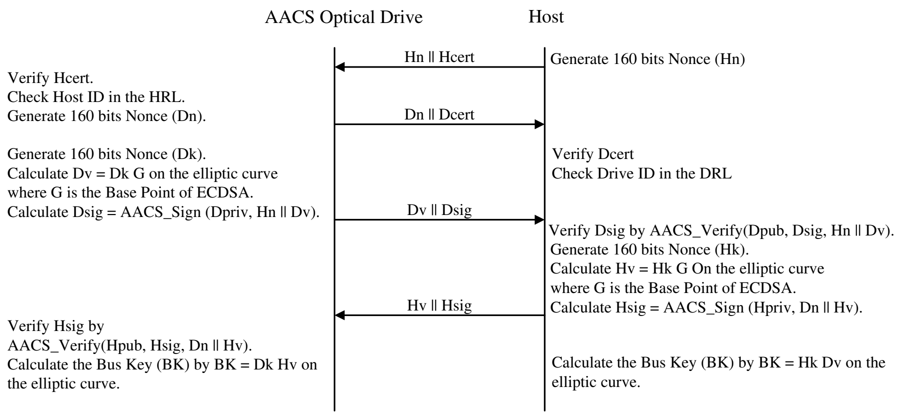
AACS Optical Drive
Host
Hn || Hcert
Generate 160 bits Nonce (Hn)
Verify Hcert.
Check Host ID in the HRL.
Generate 160 bits Nonce (Dn).
Dn || Dcert
Generate 160 bits Nonce (Dk).
Calculate Dv = Dk G on the elliptic curve
where G is the Base Point of ECDSA.
Calculate Dsig = AACS_Sign (Dpriv, Hn || Dv).
Verify Dcert
Check Drive ID in the DRL
Dv || Dsig
Verify Dsig by AACS_Verify(Dpub, Dsig, Hn || Dv).
Generate 160 bits Nonce (Hk).
Calculate Hv = Hk G On the elliptic curve
where G is the Base Point of ECDSA.
Calculate Hsig = AACS_Sign (Hpriv, Dn || Hv).
Hv || Hsig
Verify Hsig by
AACS_Verify(Hpub, Hsig, Dn || Hv).
Calculate the Bus Key (BK) by BK = Dk Hv on the elliptic curve.
Calculate the Bus Key (BK) by BK = Hk Dv on the elliptic curve.

Fig. 2. The Simplified AACS Drive-Host Authentication Protocol

First of all, we want to simplify the original flow representation of the drive-host authentication scheme [1] by abstracting the core steps. The simplified scheme is represented in Fig. 2.

In the model-checking approach, protocols can be described as patterns of messages exchanged between different agents, and each agent is described as a proctype in SPIN. In the init process, we would provide a fresh instance to each proctype. Sometimes an agent may play multiple roles in the practical operation of the protocol, thus, we need to construct multiple instances of it. In the init process of the model built for the AACS drive-host authentication protocol, we construct four instances in all: PHost(host, intruder, Hn, Hv, Hcert, Hsk), PHost(host, drive, Hn, Hv, Hcert, Hsk) for the host; PDrive(drive, Dn, Dv, Dcert, Dsk) for the drive; PIntruder() for the intruder.

```
init{
    ...
    atomic{
       if
```

```
        :: run PHost(host, intruder, Hn, Hv, Hcert, Hsk)
        :: run PHost(host, drive, Hn, Hv, Hcert, Hsk)
        fi;
        run PDrive(drive, Dn, Dv, Dcert, Dsk);
        run PIntruder();
     }
}
```

In addition, we need to model the certificate used in the protocol. Instead of introducing the widely accepted format of the X.509 digital certificate, we define a new data type – Cert. In the X.509 digital certificate, there are various fields storing necessary information; but, among them, what matters to us are merely “Subject Distinguished Name” and “Public Key”. So, in this model, the structure of the digital certificate is redefined like this:

```
typedef Cert {
    mtype identifier;
    byte pk
};
```

We also need to model the private key. Actually, private key is the value of number that could be worked with the corresponding public key to sign and verify a message in order to achieve authentication: if one receives a message encrypted with a private key and such message can be decrypted using the public key acquired from the certificate of a particular agent – *P*, it could confirm that the message is sent by agent *P*. So the match relation between the certificate and the private key is the key to the overall authentication procedure, and thus it is also the focus of our modeling. In our model, we choose to expand the data structure of the private key to abstract the match relation.

```
typedef Private_Key {
    mtype identifier;
    byte sk
};
```

As we can see, there are two fields in the structure of Private_Key. Obviously, the first one, Private_Key.identifier, represents the identity of the owner. By comparing it with the first field of Cert, we can verify the match relation and then carry out the signature verification process. The following is the core part of the process simulating the host, from which we can see the signature verification process more specifically.

```
Proctype PHost(mtype self; mtype party; mtype nonce; mtype v; Cert
cert; Private_Key hk){
    mtype g1, g2, g3;  Cert c;  Private_Key k;
    atomic{
        HostRunning(self, party);
        //Host initiates a session with the corresponding party
        ca! self, nonce, cert; //Host sends "Hn||Hcert"
    }
    atomic{
        ca? eval(self), g1, c;//Host receives "Dn||Dcert"
        cb? eval(self), eval(nonce), g2, k, g3;
        //Host receives "(Hn||Dv)SK(Dsk)||Dv"
        if
        ::(g2 == g3 && c.identifier == k.identifier)
        ->HostCommit(self, c.identifier);
        //Host commits the session with the corresponding party
        cb! self, g1, v, hk, v;//Host sends "(Dn||Hv)SK(Hsk)||Hv"
        ::else skip
        fi;
    }
}
```

Besides, SPIN also provides a data type – chan, to simulate the synchronous channels in the system. According to the different message modes used in the protocol, we build corresponding structure for each of them. In this protocol, there are two message modes: $x_1 \parallel x_2$ and $(x_1 \parallel x_2)SK(x_3) \parallel x_4$. So the channel structures are defined as follows:

```
chan ca = [0] of {mtype, mtype, Cert};//Message mode x1||x2
chan cb = [0] of {mtype, mtype, mtype, Private_Key, mtype};
//Message mode (x1||x2)SK(x3)||x4
```

3.2 Formalization of the Intruder

Based on Dolev-Yao attacker model [4], the intruder could non-deterministically intercept a message on some channel to update its knowledge, and generate a new message on some channel using the known information. The intruder updates its knowledge by using analz-rules and generates messages by using synth-rules.

In the subsequent discussion, we will describe the whole process of formalizing the intruder's behavior concretely, which consists of three parts: the initial knowledge, the analz-phase and the synth-phase. We need to note that, in this model, the intruder is a legitimate agent; in other words, the intruder is not a revoked device, and it has a valid certificate signed by AACS LA and can thus sign and verify the digital signatures specified in the AACS drive-host authentication protocol.

Before the commencement of the protocol, the intruder has its initial knowledge which is the basis of the later analz-phase and synth-phase. The intruder's initial knowledge is made up of the identities of all the principles in the system, i.e. host, drive, and intruder; moreover, the intruder also holds its private key Isk, its certificate Icert and the generic data gD. After the protocol begins running, the intruder would start on intercepting and generating messages.

In the analz-phase, the intruder breaks up a message into constituent parts and stores them; furthermore, it also verifies the signatures included in the messages using the certificates contained in its knowledge. And by doing these jobs, the intruder could increase its knowledge. For instance, if the intruder intercepts the message Hn||Hcert, it could acquire and store the nonce Hn and the certificate Hcert. In order to avoid storing redundant knowledge elements, we assume that the intruder always records the learned items in their most elementary forms. For example, if message Hn||Icert is intercepted, what the intruder records is Hn, rather than Icert or the whole message Hn||Icert; because Icert is not fresh to it, and message Hn||Icert can be built from Hn and Icert, which is not in the most elementary form. We also assume that the intruder records a complex message in its knowledge only if it cannot build that message. Taking another case for example, if the intruder intercepts the message (Dn||Hv)SK(Hsk)||Hv, besides recording the parameter Hv, it also needs to record the complex part (Dn||Hv)SK(Hsk). Since the intruder cannot get private key of the host – Hsk, and thus cannot generate the signature, though it might know the certificate of the host – Hcert, and decrypt that signature.

In the synth-phase, based on the synth-rules, the intruder can generate a message by choosing the recipient and the message type and filling in each field with the appropriate data item which is known to it; besides, the intruder also can simply replay an entire stored message. For example, if the intruder possesses the data items Hn and Hcert, it can concatenate them into Hn||Hcert and send it to the drive. Theoretically, the intruder can generate whatever it wants; but, to improve the efficiency of our model, we build a restriction in the synth-phase: the message the intruder generates should be valid and in accordance with the corresponding message mode. Obviously, the purpose of making this restriction is to prevent the intruder from generating invalid message. For instance, if the intruder generates and sends a message (Hn||Hv)SK(Hsk)||Dv, none of the agents in the system would accept it, since this message does not comply with the message mode which requires the second parameter should be identical with the fourth parameter (but Hv≠Dv).

4 Formalization of the Authenticity Property

4.1 Formalization of the Authenticity Property

Property formalization is another essential part of verifying security protocols with model checkers. In general, secrecy and authenticity are the properties often be checked when analyzing security protocols, but in this paper, secrecy does not need to be considered. Since in the AACS drive-host authentication protocol, data items contained in messages are transferred in two forms – plain text and cipher text. The plain text data could be seen by any agent. The cipher text data, encrypted with someone's private key, could be decrypted by anyone who possesses the corresponding certificate; and other agents without the proper certificate also could acquire the corresponding plain text of all the cipher text data, since in this protocol all the cipher text data has another copy which is transferred in the plain text. So secrecy is not the property worth verification in this protocol. The property we choose to verify here is authenticity.

During the verification, we firstly build three roles involved – Host, Drive and Intruder; then, we define six global Boolean variables:

```
bit HostRunningHD = 0,  HostCommitHD = 0,  HostKnowDv = 0;
bit DriveRunningHD = 0, DriveCommitHD = 0, DriveKnowHv = 0;
```

HostRunningHD is true iff Host takes apart in a session with Drive. DriveRunningHD is true iff Drive takes apart in a session with Host. HostCommitHD is true iff

Host commits to a session with Drive. DriveCommitHD is true iff Drive commits to a session with Host. HostKnowDv is true iff Host knows the parameter Dv. DriveKnowHv is true iff Drive knows the parameter Hv.

The authentication of Host to Drive can be expressed as that HostRunningHD must become true before the DriveCommitHD becomes true, whereas the converse authentication of Drive to Host is that DriveRunningHD must become true before the HostCommitHD becomes true. These properties can be expressed in the LTL formalism:

$$\Box((\Box!DriveCommitHD) \mid\mid (!DriveCommitHD \cup HostRunningHD)) \ . \tag{3}$$

$$\Box((\Box!HostCommitHD) \mid\mid (!HostCommitHD \cup DriveRunningHD)) \ . \tag{4}$$

Equation (3) means that the protocol suffices to the authentication of Host to Drive; (4) means that the protocol suffices to the authentication of Drive to Host.

4.2 The Experimental Result

After the verification on SPIN, we discover two attack instances (Attack 1 & Attack 2) violating (3) and two attack instances (Attack 3 & Attack 4) violating (4). Those instances are shown in Fig. 3, and the relevant experimental data is list in Table 1.

Table 1. Attack and Relevant Data

Attack	Attack 1	Attack 2	Attack 3	Attack 4
HostRunningHD	0	0	1	1
HostCommitHD	0	0	1	1
DriveRunningHD	1	1	0	0
DriveCommitHD	1	1	0	0
HostKnowDv	1	0	1	1
DriveKnowHv	1	1	0	1

Here, we want to make some notations about those attacks shown in Fig. 3. In Attack 1 and Attack 2, Host initially sends its random nonce Hn and certificate Hcert to Intruder rather than Drive, in order to initiate a session with Intruder; in Attack 3 and Attack 4, Host intends to send its random nonce Hn and certificate Hcert to Drive to initiate a session with Drive, but this message is intercepted by Intruder. In addition, in Attack 1 and Attack 4, Host and Drive can negotiate a shared Bus Key when the session finished; however, they cannot get the shared Bus Key in Attack 2 and Attack 3, since Host has no way to get Drive's parameter Dv in Attack 2 and Drive cannot get Host's parameter Hv in Attack 3 during the whole session.

5 Formalization of the Collusion Attack

5.1 Introduction of the Collusion Attack

AACS is applicable to a PC-based system. In such a system, a drive and PC host act together as the Recording Device and/or Playback Device for AACS protected content.

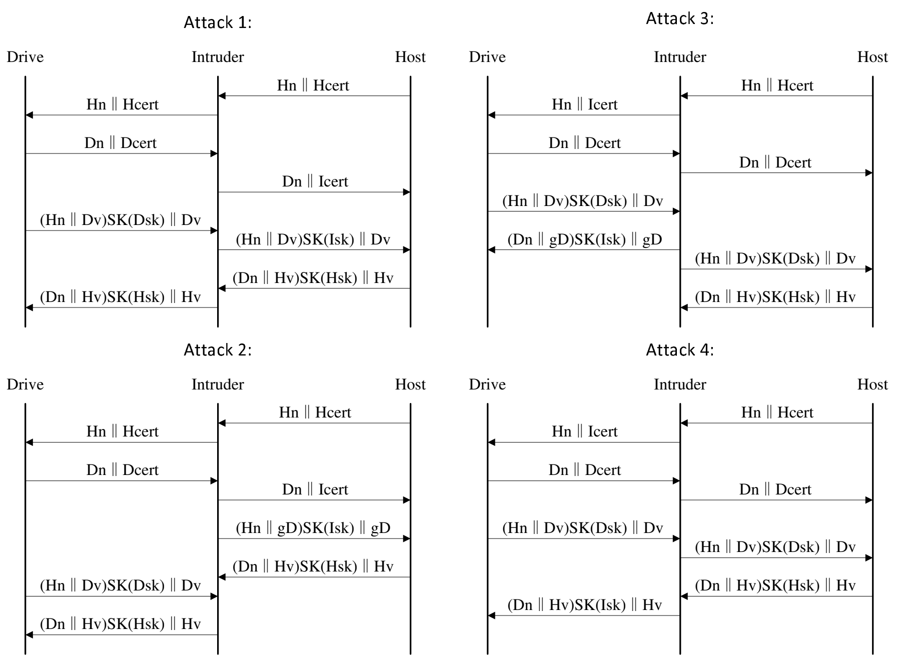

Fig. 3. The Attack Instances

Mutual authentication is the initial procedure in the whole system, by which the drive and the PC host verify each counterpart is an AACS compliant device which has a valid certificate signed by the AACS LA and can sign and verify digital signatures specified in the AACS drive-host authentication protocol.

In the AACS drive-host authentication scheme, the host's process of verifying the drive's legitimacy consists of three steps: first, the host verifies the signature of the Drive Certificate using the AACS LA Public Key; next, it checks the Drive Revocation List to ensure that the Drive ID of the Drive Certificate has not been revoked; then it verifies the second message sent by the drive to check whether the drive pass the nonce-challenging procedure or not. If the above verifications succeed, the host could confirm the validity of the drive, and exchange parameters with it to negotiate a shared Bus Key.

From the above analysis, we think of a special kind of attack aiming at offering a revoked drive the opportunity to bypass the authentication procedure, negotiate a shared Bus Key with a legal host and use this key to exchange necessary information with the host in order to play/record the protected content in the disc. And this attack could be successfully carried out by hiring a third party – a legitimate drive; obviously, this kind of attack, launched by a revoked drive and a hired drive, is what we called "the collusion attack". But the relationship between them is not mutual trust but mutual utilization. To the revoked drive, the hired drive is a lessor who leases out its legitimate identity; and to the hired drive, the revoked drive is just one of the lessees who pay for the use of its identity.

Therefore, there is a restriction lies in this attack mode: Bus Key is the secret merely known to the host and the revoked drive, which implies the hired drive cannot acquire it. Bus Key, in the AACS system, is used for message authentication in the subsequent interaction between the host and the drive, after the authentication protocol is over. Accordingly, in this attack mode, Bus Key is used for messages authentication between the host and the revoked drive after the authentication protocol completed running. Once the hired drive possesses Bus Key, it has the ability to tamper the messages, obstruct the host from acquiring parameters necessary to decrypt the encrypted content in the disc and prevent the revoked drive from playing/recording the disc, which contravenes the goal of the collusion attack. So the purpose of setting the restriction is to make the lessee (the revoked drive) get rid of the influence of the third party (the hired drive) after the authentication protocol completed running, and successfully interact with the host.

Considering the AACS adopts the technique of Elliptic Curve Cryptographic Signature Algorithm (ECDSA) in the key agreement procedure, the crucial factor of implementing the restriction specified above is the parties participating in the Bus Key negotiation. In this attack mode, besides the host and the revoked drive, there should not be a third party participating in the Bus Key negotiation. That is to say, the hired drive should not replace the revoked drive's parameter Dv by its own parameter in order to negotiate the Bus Key with the host itself; and the revoked drive should participate in the negotiation itself, rather than simply obtain a Bus Key generated by the hired drive.

5.2 Formalization of the Collusion Attack

There are three parties in this attack mode: Host, Drive-R and Drive-L.

- Host is just a legal host in the PC-based system;
- Drive-R is a revoked drive whose ID is in the Drive Revocation List located in Host;
- Drive-L is a legitimate drive.

Remarkably, in this attack mode, Drive-R plays the role of Drive and Drive-L plays the role of Intruder. The details about this attack are described as follows:

- If Drive-R wants to interact with Host, it hires Drive-L as its accomplice to employ the attack.
- During the protocol running, Host only can see the certificate of Drive-L and does not know the existence of Drive-R in the whole process of interaction.
- The parameter Dv, calculated by Drive-R based on its 160 bits nonce Dk, the elliptic curve and the Base Point G of ECDSA, could be finally acquired by Host in order to calculate the Bus Key.
- The parameter Hv, calculated by Host based on its 160 bits nonce Hk, the elliptic curve and the Base Point G of ECDSA, could be finally known by Drive-R for the purpose of calculating the Bus Key.
- Because Drive-L is the device hired by Drive-R, Drive-R can pick up necessary information from Drive-L. That is to say, the fact Drive-L knows the parameter Hv is equivalent to the fact that Drive-R knows Hv.
- During the whole process, Drive-L cannot get the Bus Key.

Based on the above analysis, we define eight global Boolean variables:

```
bit HostRunningHD = 0,  HostCommitHD = 0,  HostKnowDv = 0;
bit DriveRunningHD = 0, DriveCommitHD = 0, DriveKnowHv = 0;
bit IntruderKnowHv = 0, HostKnowDcert = 0;
```

HostRunningHD is true iff Host takes apart in a session with Drive. DriveRunningHD is true iff Drive takes apart in a session with Host. HostCommitHD is true iff Host commits to a session with Drive. DriveCommitHD is true iff Drive commits to a session with Host. HostKnowDv is true iff Host knows the parameter Dv. DriveKnowHv is true iff Drive knows the parameter Hv. IntruderKnowHv is true iff Intruder knows the parameter Hv. HostKnowDcert is true iff Host knows Dcert, the certificate of Drive.

The property we want to check could be expressed in the LTL formalism:

$$\diamond\,(HostKnowDv \&\& (DriveKnowHv \,||\, IntruderKnowHv) \&\& \\ !HostKnowDcert \&\& !HostRunningHD \&\& !HostCommitHD)\ . \quad (5)$$

If there is an instance existing in the protocol running that could suffice (5), we could confirm the feasibility of this collusion attack.

5.3 The Experimental Result

Using SPIN, we have found five attack instances shown in Fig. 4 and Fig. 5, and the relevant experimental data is list in Table 2.

Table 2. Attack and Relevant Data

Attack	Attack 1	Attack 2	Attack 3	Attack 4	Attack 5
HostRunningHD	0	0	0	0	0
HostCommitHD	0	0	0	0	0
HostKnowDv	1	1	1	1	1
HostKnowDcert	0	0	0	0	0
DriveRunningHD	0	1	1	0	1
DriveCommitHD	0	1	0	0	0
DriveKnowHv	1	1	1	0	0
IntruderKnowHv	1	1	1	1	1

First, let us focus on Attack 1 and Attack 2. In these two attack instances, the protocol could be successfully completed. In the end of the session, Drive (Drive-R) acquires Hv, and Host gets Dv. (Attack 2 has already been found by J.Sui and D.R. Stinson by simulating the "Unknown Key-Share Attack" in [5].)

In Attack 3, the session does not proceed well. The last message Drive (Drive-R) received has an improper data item: when verifying the signature (Dn||Hv)SK(Isk) by using Hcert received in the first message, Drive (Drive-R) would detect the invalidity of the signature. According to the rule of this protocol, if the verification fails, Drive (Drive-R) will determine the counterpart is not compliant and abort this session. But in this model, Drive (Drive-R) is not a good boy but a revoked drive. So, after the failure of the signature verification, it would not only keep the session, but also pick up the parameter Hv from the last message, regardless of the validity of the signature. For the purpose of checking the authenticity of the parameter Hv, Drive (Drive-R) is still required to check the validity of the signature using the certificate of Intruder Icert, which could be got from Intruder before the protocol running, since they are partners.

From the analysis of Attack 3, we can conclude that, in this attack mode, Drive (Drive-R) does not care the rule of the protocol; what it actually cares is whether or not it can get the parameter Hv. And this conclusion could also explain the feasibility of Attack 5, in which Drive (Drive-R) encounters the same problem as in Attack 3.

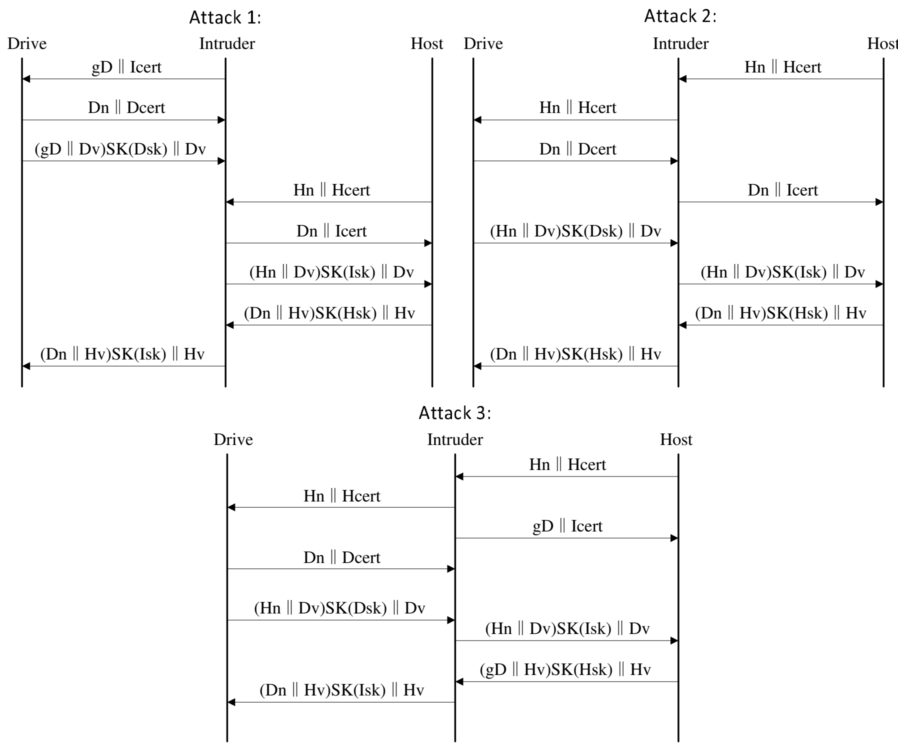

Fig. 4. The Attack 1 & Attack 2 & Attack 3

Moreover, we want to discuss the instances of Attack 4 and Attack 5. It is easy to discover that Drive (Drive-R) could not obtain the parameter Hv at the end of the protocol running; whereas Intruder (Drive-L) has already got Hv at this time. Thus, what Drive (Drive-R) needs to do is just asking Intruder (Drive-L) to send Hv to it. And for the convenience of checking the authenticity of the parameter Hv sent by Intruder (Drive-L), this additional message, written in italics in both attack instances shown in Fig. 5, is requested to follow one of the message modes in the protocol as (Dn||Hv)SK(Isk)||Hv. After receiving this additional message, Drive (Drive-R) would check the validity of the

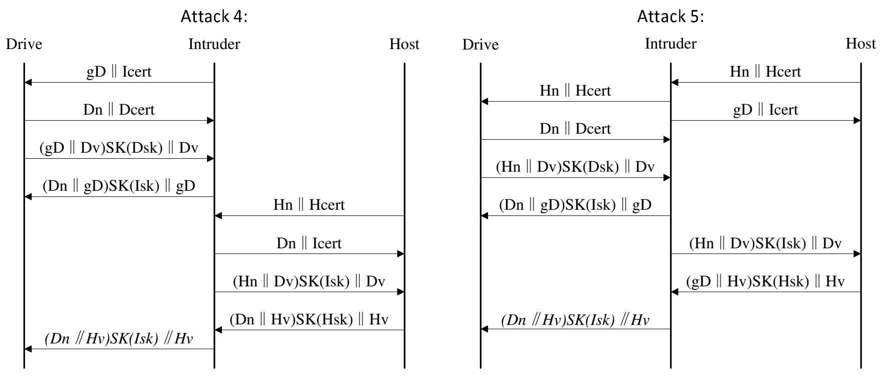

Fig. 5. The Attack 4 & Attack 5

signature using Intruder's certificate Icert which it can get before the protocol running, and pick up the parameter Hv.

5.4 The Modified Scheme

So far, there are several papers about the AACS drive-host authentication protocol. Particularly, a modified scheme has been advanced, which is declared to be competent of resisting the Unknown Key-Share attack and the Man-In-The-Middle attack in [5].

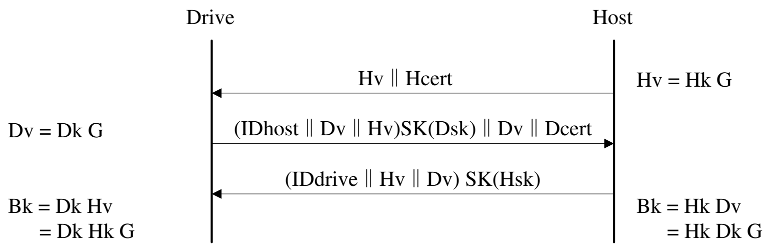

Fig. 6. The Modified Scheme

The modified version is shown in Fig. 6.

Compared with the original one, this modified scheme has two remarkable features: first, Hv and Dv play the role as random challenges, instead of Hn and Dn; second, the ID of the message receiver has been added into the signature.

In this paper, we also carry out an experiment on this modified scheme, and find out that this scheme cannot resist the collusion attack either. In our experiment, in order to check the effectiveness of the modified scheme, we construct a corresponding model on SPIN, and make use of (5) to verify the feasibility of the collusion attack. During the process of verification, we have found four attack instances. These instances are shown in Fig. 7 and Fig. 8, and the relevant experimental data is list in Table 3.

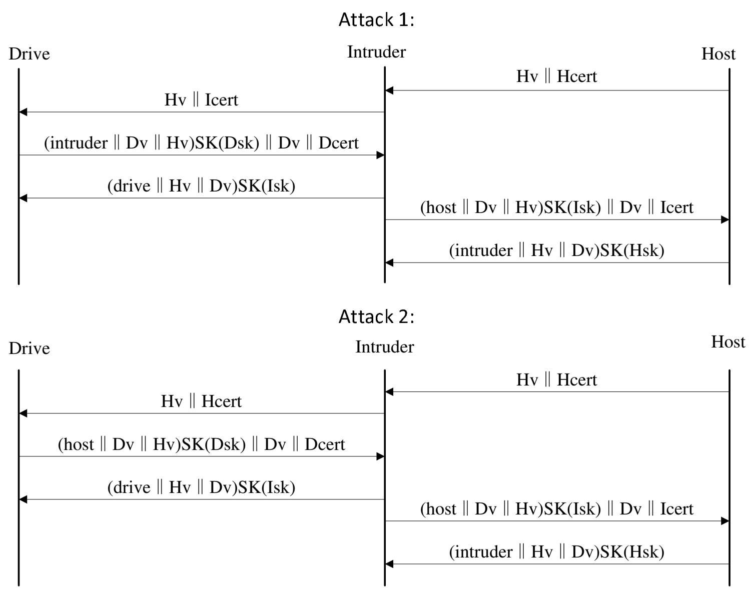
Attack 1:
Drive
Intruder
Host
Hv || Hcert
Hv || Icert
(intruder || Dv || Hv)SK(Dsk) || Dv || Dcert
(drive || Hv || Dv)SK(Isk)
(host || Dv || Hv)SK(Isk) || Dv || Icert
(intruder || Hv || Dv)SK(Hsk)
Attack 2:
Drive
Intruder
Host
Hv || Hcert
Hv || Hcert
(host || Dv || Hv)SK(Dsk) || Dv || Dcert
(drive || Hv || Dv)SK(Isk)
(host || Dv || Hv)SK(Isk) || Dv || Icert
(intruder || Hv || Dv)SK(Hsk)

Fig. 7. The Attack 1 & Attack 2

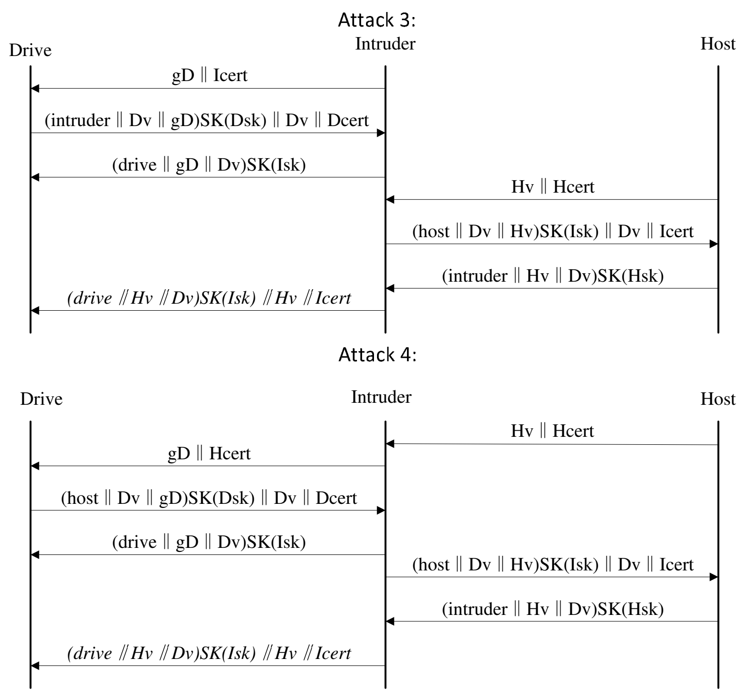
Attack 3:
Drive
Intruder
Host
gD || Icert
(intruder || Dv || gD)SK(Dsk) || Dv || Dcert
(drive || gD || Dv)SK(Isk)
Hv || Hcert
(host || Dv || Hv)SK(Isk) || Dv || Icert
(intruder || Hv || Dv)SK(Hsk)
(drive || Hv || Dv)SK(Isk) || Hv || Icert
Attack 4:
Drive
Intruder
Host
Hv || Hcert
gD || Hcert
(host || Dv || gD)SK(Dsk) || Dv || Dcert
(drive || gD || Dv)SK(Isk)
(host || Dv || Hv)SK(Isk) || Dv || Icert
(intruder || Hv || Dv)SK(Hsk)
(drive || Hv || Dv)SK(Isk) || Hv || Icert

Fig. 8. The Attack 3 & Attack 4

As mentioned earlier, Drive (Drive-R) and Intruder (Drive-L) are allies in the collusion attack, and the purpose of the attack is to let Drive (Drive-R) and Host exchange the parameters Hv and Dv and prevent Host from knowing the existence of Drive in the whole process of the protocol running. In Attack 1 and Attack 2, this purpose could be successfully reached; and in Attack 3 and Attack 4, though failing to get Hv at the end of the protocol, Drive (Drive-R) could get it from Intruder (Drive-L). Similarly, for the convenience of checking the parameter Hv, the additional message, written in italics in both attack instances shown in Fig. 8, is still required to comply with one of the message modes used in the modified scheme – (drive||Hv||Dv)SK(Isk)||Hv||Icert.

Table 3. Attack and Relevant Data

Attack	Attack 1	Attack 2	Attack 3	Attack 4
HostRunningHD	0	0	0	0
HostCommitHD	0	0	0	0
HostKnowDv	1	1	1	1
HostKnowDcert	0	0	0	0
DriveRunningHD	0	1	0	1
DriveCommitHD	0	0	0	0
DriveKnowHv	1	1	0	0
IntruderKnowHv	1	1	1	1

5.5 Relevant Analysis

Apparently, the collusion attack would make a revoked drive bypass the authentication procedure, negotiate a shared Bus Key with a legitimate host and use this key to exchange necessary information with the host in order to play/record the protected content in the disc released by AACS LA after the drive is revoked. All this work could be done with the assistance of a valid drive with a legitimate certificate. This situation is just like once getting the valid serial number and password from a licensed user, one can install and use an unauthorized copy of a legally released software, downloaded from the internet illegally or obtained from illegal DVD/VCD duplicators, unless the software releaser detects this improper action and locks that compromised serial number. Similarly, with the help of a legitimate drive, one could freely use a revoked drive along with a PC host to play/record discs until the AACS LA detects this illegitimate action, which has threatened the security offered by the AACS system seriously.

6 Conclusion

Through the strict model-checking analysis of the AACS drive-host authentication protocol, depending on the variant of the Dolev-Yao attacker model [4], we have discovered a few weaknesses of the target protocol in providing authenticity. Besides, we have advanced a novel collusion attack and found several corresponding attack instances in the original scheme of the target protocol and a modified scheme [5].

In this paper, we have not yet advanced a new scheme to resist that collusion attack. Future work might focus on the modification of the AACS drive-host authentication protocol by introducing the threshold decryption scheme which is mainly used to prevent the collusion attack.

7 Acknowledgement

This research was funded by grant 90604010 from the National Nature Science Foundation and grant 2007BC311202 of the National Key Foundation Research Plan of China.

References

1. Intel et al. Advanced Access Content System (AACS) – Introduction and Common Cryptographic Elements. Revision 0.91, pages 32-34 (2006)
2. A.S. Khan, M. Mukund and S.P. Suresh. Generic Verification of Security Protocols. Technical Report, Chennai Mathematical Institute (2005)
3. D. Basin et al. An On-the-fly Model-Checker for Security Protocol Analysis. Proc. ESORICS 2003, pp.253-270. (2003)
4. D. Dolev and A.C. Yao. On the security of public key protocols. IEEE Transactions on Information Theory, 29(2):198C208, 1983. (1983)
5. J. Sui and D.R. Stinson. A Critical Analysis and Improvement of AACS Drive-Host Authentication. Centre for Applied Cryptographic Research (CACR) (2007)
6. Kevin Henry, J. sui, G. Zhong. An Overview of the Advanced Access Content System (AACS). Centre for Applied Cryptographic Research (CACR), 2007 April 12 (2007)
7. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. Journal of computer security, 6:85C128, 1998. (1998)
8. P. Maggi and R. sisto. Using SPIN to Verify Security Properties of Cryptographic Protocols. In Proceedings of the 9th International SPIN Workshop on Model Checking of Software, number 2318 in Lecture Notes in Computer Science, pages 187-204 (2002)
9. R. Ramanujam and S.P. Suresh. A decidable subclass of unbounded security protocols. In Proc. IFIP Workshop on Issues in the Theory of Security (WITS03), pages 11C20, Warsaw (Poland) (2003)
10. R. Ramanujam and S.P. Suresh. Decidability of context-explicit security protocols. Journal of Computer Security, 13(1):135C165, 2005. (2005)
11. Spin Workshop. Spin-Formal Verification, http://spinroot.com/spin/whatispin.html (2008)

Appendix: The Sequence Charts of Attack Instances in SPIN

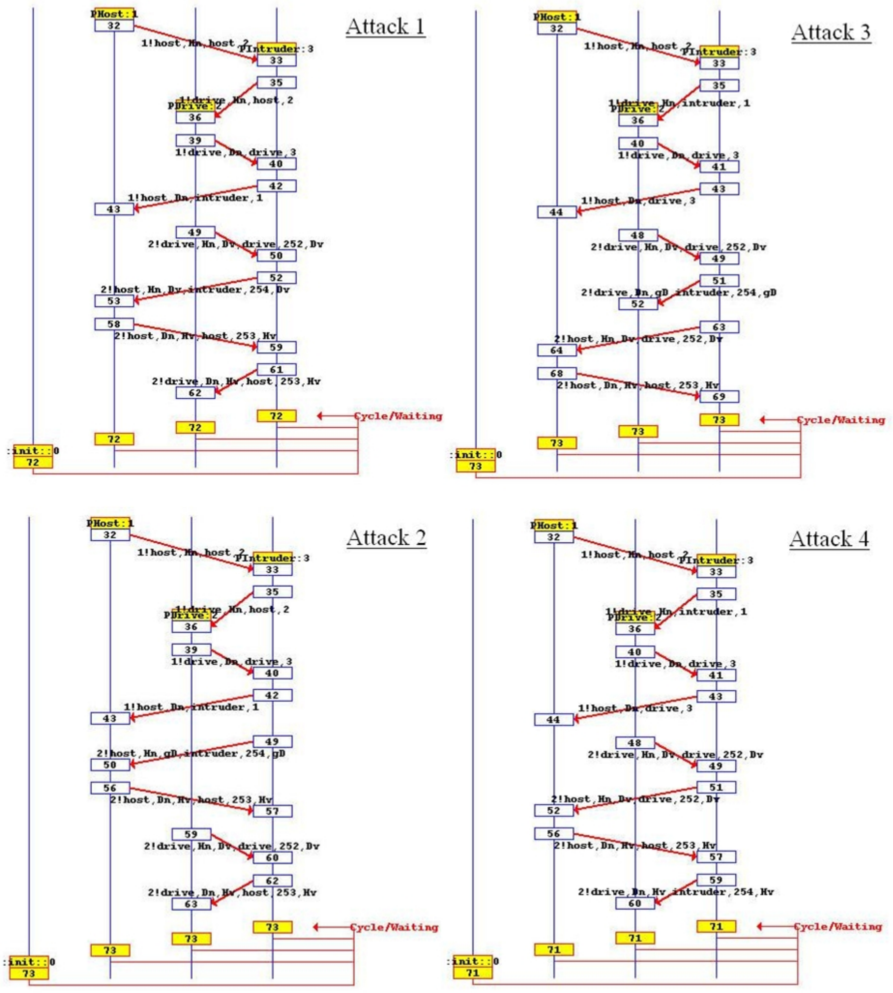

Fig. 9. The Attack Instances Violating the Property of Authenticity

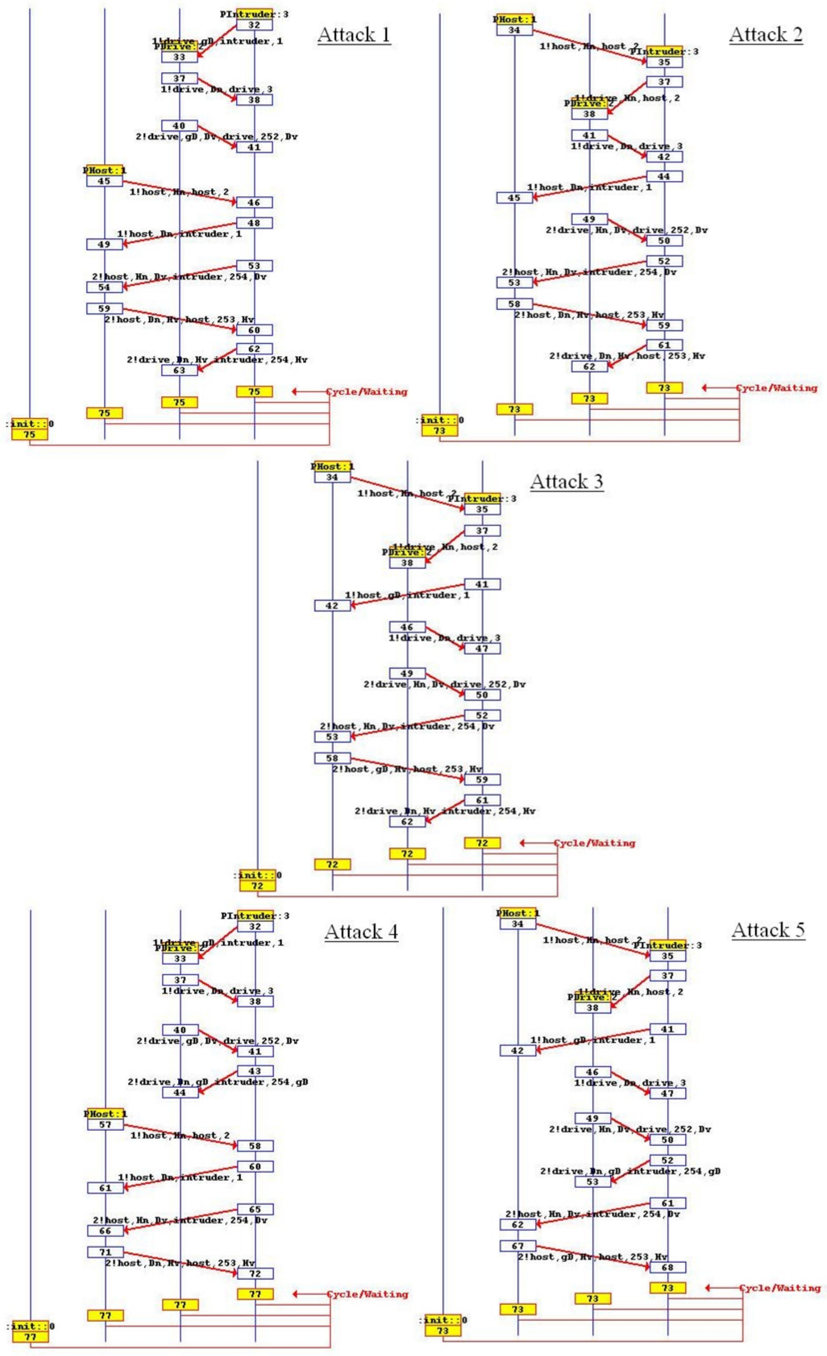

Fig. 10. The Collusion Attack Instances in the AACS Drive-Host Authentication Scheme

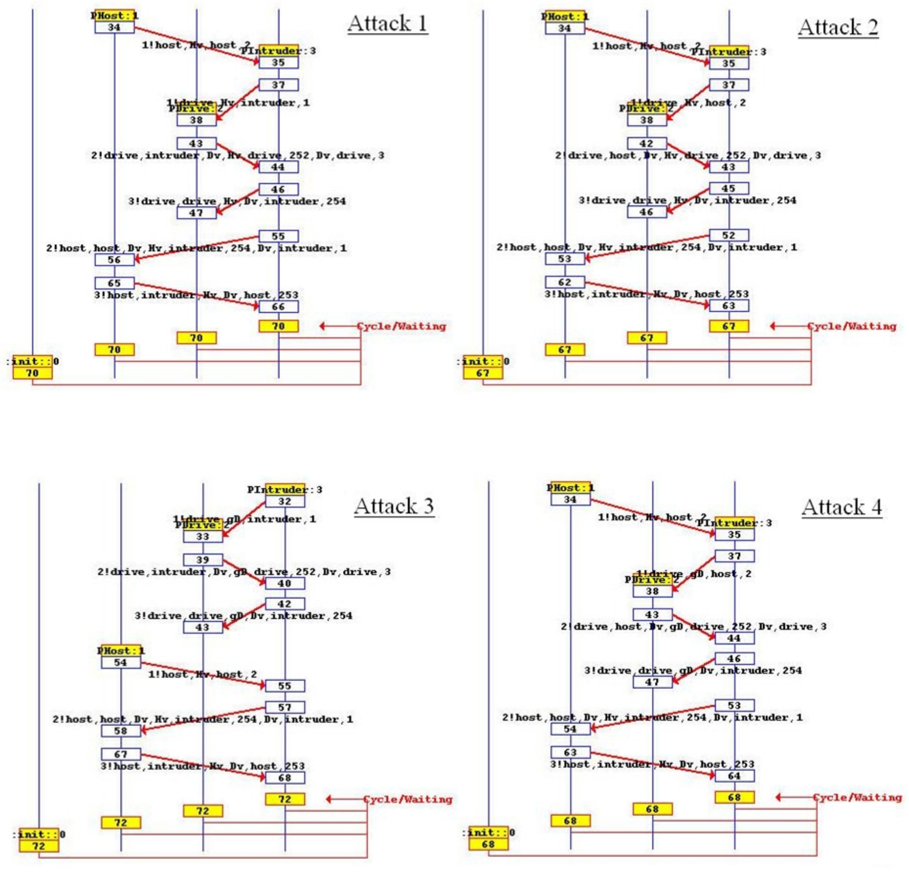
Attack 1
PHost:1
34
1!host,Hv,host,2
PIntruder:3
35
37
1!drive,Hv,intruder,1
PDrive:2
38
43
2!drive,intruder,Dv,Hv,drive,252,Dv,drive,3
44
46
3!drive,drive,Hv,Dv,intruder,254
47
55
2!host,host,Dv,Hv,intruder,254,Dv,intruder,1
56
65
3!host,intruder,Hv,Dv,host,253
66
70
Cycle/Waiting
70
70
:init::0
70
Attack 2
PHost:1
34
1!host,Hv,host,2
PIntruder:3
35
37
1!drive,Hv,host,2
PDrive:2
38
42
2!drive,host,Dv,Hv,drive,252,Dv,drive,3
43
45
3!drive,drive,Hv,Dv,intruder,254
46
52
2!host,host,Dv,Hv,intruder,254,Dv,intruder,1
53
62
3!host,intruder,Hv,Dv,host,253
63
67
Cycle/Waiting
67
67
:init::0
67
Attack 3
PIntruder:3
32
1!drive,gD,intruder,1
PDrive:2
33
39
2!drive,intruder,Dv,gD,drive,252,Dv,drive,3
40
42
3!drive,drive,gD,Dv,intruder,254
43
PHost:1
54
1!host,Hv,host,2
55
57
2!host,host,Dv,Hv,intruder,254,Dv,intruder,1
58
67
3!host,intruder,Hv,Dv,host,253
68
72
Cycle/Waiting
72
72
:init::0
72
Attack 4
PHost:1
34
1!host,Hv,host,2
PIntruder:3
35
37
1!drive,gD,host,2
PDrive:2
38
43
2!drive,host,Dv,gD,drive,252,Dv,drive,3
44
46
3!drive,drive,gD,Dv,intruder,254
47
53
2!host,host,Dv,Hv,intruder,254,Dv,intruder,1
54
63
3!host,intruder,Hv,Dv,host,253
64
68
Cycle/Waiting
68
68
:init::0
68

Fig. 11. The Collusion Attack Instances in the Modified AACS Drive-Host Authentication Scheme