A Critical Analysis and Improvement of AACS Drive-Host Authentication

Jiayuan Sui and Douglas R. Stinson*
David R. Cheriton School of Computer Science,
University of Waterloo,
Waterloo, ON, N2L 3G1, Canada
E-mail: {jsui, dstinson}@uwaterloo.ca

Abstract: This paper presents a critical analysis of the AACS drive-host authentication scheme. A few weaknesses are identified which could lead to various attacks on the scheme. In particular, we observe that the scheme is susceptible to unknown key-share and man-in-the-middle attacks. Modifications (based on the ISO and the SIGMA protocols) of the scheme are suggested in order to provide better security. A proof of security of the modified scheme based on the ISO protocol is also presented. The modified schemes achieve better efficiency than the original scheme.

Keywords: AACS; mutual authentication; key agreement.

Reference

Biographical notes: Jiayuan Sui has completed his Master's degree in computer science under the supervision of Dr. Douglas R. Stinson at the David R. Cheriton School of Computer Science at the University of Waterloo.

Douglas Stinson obtained his BMath degree from the University of Waterloo in 1978, his MSc from Ohio State in 1980 and his PhD in Combinatorics and Optimization from the University of Waterloo in 1981. He previously has held academic positions at the University of Manitoba, where he was an NSERC University Research Fellow, and the University of Nebraska-Lincoln. Currently he holds the position of Professor and University Research Chair in the David R. Cheriton School of Computer Science at the University of Waterloo. He held the NSERC/Certicom Industrial Research Chair in Cryptography from 1998-2003.
Stinson's research interests include cryptography and computer security, combinatorics and coding theory, and applications of discrete mathematics in computer science. Doug is the author of over 250 research papers as well as the popular textbook "Cryptography: Theory and Practice", the third edition of which was published in 2005. He was one of the founding co-editors of the Journal of Combinatorial Designs, which is published by John Wiley & Sons, and he has served on the editorial boards of numerous other journals.

1 Introduction

Advanced Access Content System (AACS) is a content distribution system for recordable and pre-recorded media. It has been developed by eight companies: Disney, IBM, Intel, Matsushita (Panasonic), Microsoft, Sony, Toshiba, and Warner Brothers. Most notably, AACS is used to protect the next generation of high definition optical discs such as Blu-ray and HD-DVD.

To design a media protection scheme that is able to run on open platforms like PCs, designers have to make sure that the scheme is not susceptible to the "virtual device attack". A virtual device can mimic a physical hardware device in all respects, so that the CPU is tricked into believing that a device exists when actually it does not. To deploy a virtual device attack on a media system such as the DVD playback system, the attacker can build software that implements a virtual DVD drive. The content of the optical disc is moved onto the computer's hard drive as a disc image. The attacker can then play back this "DVD

*Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the grant NSERC-RGPIN #203114-06

disc" through the virtual DVD drive on a legitimate DVD player software.

The attacker can certainly duplicate the disc image into multiple copies and disseminate them illegally, even though he never learns the content of the DVD in the clear. In order to defend against this attack, the drive has to have the ability to prove to the host (e.g. the playback software) that it is a legitimate drive. This can be done through a cryptographic authentication protocol.

The AACS drive-host authentication scheme achieves mutual authentication, which means that the drive proves to the host its legitimate identity and the host has to prove its identity to the drive. After the drive and the host complete a successful session of the protocol, a shared secret key is established between them. Therefore, AACS drive-host mutual authentication protocol is combined with a key agreement protocol. The shared secret key is then used for message authentication purposes.

1.1 Mutual Authentication Protocol and Key Agreement Protocol

In a mutual authentication protocol, the two participating entities need to prove their identities to each other. If an entity has successfully proven its identity to the other entity, the other entity is required to "accept". A session of a mutual authentication protocol is a successfully completed session if both participants have accepted by the end of the session. Mutual authentication protocols can be devised by using either symmetric or asymmetric key cryptographic primitives. Stinson (2002) (Chapter 9) provides some good background on mutual authentication protocols.

After two entities have authenticated themselves to each other, most likely they will want to communicate with each other. It therefore makes sense to combine a key agreement protocol or a key exchange protocol with a mutual authentication protocol, because a shared secret key provides confidentiality and/or data integrity to both communicating entities. In a key agreement protocol, both honest participating entities contribute information that is used to derive a shared secret key. A key agreement protocol most often uses asymmetric-key primitives. In a key exchange protocol, the shared secret key is not necessarily a product of the information contributed by both participating entities, and it is hard to achieve perfect forward secrecy in a key exchange protocol. We concentrate our study on key agreement protocols.

A key agreement protocol is said to provide *implicit key authentication* to both entity A and entity B if A is assured that no one other than B can possibly learn the value of the shared secret key (likewise, B is assured that no one other than A can learn the value of the key), assuming the peer is honest (e.g., doesn't reveal the key). Note that this property does not necessarily mean that A is assured of B actually possessing the key nor is A assured that B can actually compute the key. A key agreement protocol with implicit key authentication is called an *authenticated key agreement* (AK) protocol.

A key agreement protocol is said to provide *implicit key confirmation* if A is assured that B can compute the secret key while no others can, and vice versa. A protocol provides *explicit key confirmation* if A is assured that B has computed the secret key and no one other than B can compute the key, and vice versa. A key agreement protocol that provides key confirmation (either implicit or explicit) to both participating entities is called an *authenticated key agreement with key confirmation* (AKC) protocol. For example, explicit key confirmation can be achieved by using the newly derived key to encrypt a known value and to send it to the other entity. In most cases, using a key agreement protocol with implicit key confirmation is sufficient. For more information on key agreement protocols, please refer to (Stinson, 2002, Chapter 11).

1.2 Our Contributions

In this paper, we present a rigorous analysis of the AACS drive-host authentication scheme. Specifically, we identify a few weaknesses present in the scheme which could lead to various attacks. It is yet to be known whether those weaknesses will lead to piracy of multimedia content. Nevertheless, we believe that it is not desirable for such a widely-deployed system to employ a weak cryptographic protocol if it can be made secure fairly easily. We propose an improvement of the original scheme based on the well-established ISO key agreement protocol. The improved scheme provides secure mutual authentication as well as authenticated key agreement with key confirmation. We also discuss the security of the improved scheme. The improved scheme is designed with the goal of requiring little change to be made to the original scheme, so implementation of the improved scheme is straightforward. In addition, the improved scheme requires less interaction between the drive and the host, and therefore it is more efficient than the original scheme. Furthermore, we briefly present another improved scheme based on the SIGMA protocol. This scheme provides careful binding between the entity ID and the shared secret key; however, it requires additional cryptographic primitives to be added to the original scheme.

1.3 Organization

In Section 2, we introduce the AACS drive-host authentication scheme. Our analysis of the AACS drive-host authentication scheme is presented in Section 3, where we identify several weaknesses in the scheme and provide corresponding improvements based on the ISO protocol. In Section 4, we discuss the security of the improved drive-host authentication scheme. In Section 5 we briefly introduce an improved scheme based on the SIGMA protocol, followed by a conclusion in Section 6.

2 AACS Drive-Host Authentication scheme

2.1 The Basic Procedure

When using AACS in a PC-based system where the drive and the host are separate entities, both the drive and the host are issued certificates from the AACS LA (AACS Licensing Administrator). This allows each entity to verify whether or not the other is trustworthy and in compliance with the AACS specifications. These certificates, called the *drive certificate* and *host certificate*, each contain fields stating the capabilities of the device, a unique identifier, the device's public key, and a signature from the AACS LA verifying the integrity of the certificate signed with an AACS LA private key. Both the drive and the host have the corresponding AACS LA public key for signature verification. A full description of the certificate format can be found in the AACS Introduction and Common Cryptographic Elements specification (AACS Specification, Chapter 4).

Authentication between the drive and the host occurs each time new media is placed into the drive. This is necessary because the new disc may contain updated revocation lists. Each compliant disc contains a data structure called the *media key block* (MKB), which holds the necessary information needed to derive the keys to decrypt the content. It also contains the latest *drive revocation list* (DRL) and *host revocation list* (HRL) which, respectively, contain a list of IDs of the revoked drives and a list of IDs of the revoked hosts. A drive may only communicate with a host that has not been revoked, and a host may only communicate with a drive that has not been revoked.

Appendix A shows a flow representation of the AACS drive-host authentication scheme. A detailed description can be found in (AACS Specification, Section 4.3). The original scheme consists a total of twenty-nine steps. The numbers on each line in Appendix A correspond to the number given to that step in (AACS Specification, Section 4.3). A simplified version consisting only the core steps involved in authentication and key agreement is shown in Figure 3.

After successfully completing the drive-host authentication algorithm, the drive and the host have established a shared *bus key* based on an elliptic curve Diffie-Hellman key agreement protocol (NIST, 2007). It is interesting to note that while this key could be used to encrypt messages between the drive and the host, it is not actually used for this purpose. Instead, the bus key is used solely for message authentication by including a MAC for any message traveling between the drive and the host. The current AACS specifications do not require either the drive or the host to be capable of encrypting and decrypting bus messages; however there is a flag in each certificate stating whether or not an entity is capable of performing bus encryption.

2.2 Requesting Media-Specific Information

In the previous section, we mentioned that the drive and the host can establish a shared bus key. This bus key is used to ensure that communications between the drive and the host remain unaltered. In this section, we briefly describe how the bus key is used when requesting certain values from the disc, such as the *volume ID* or the *pre-recorded media serial number*. These are necessary when decrypting a disc, as they are used to derive the *title key*, which in turn is used to decrypt the content.

The protocol is simply a direct exchange of messages using MACs to ensure that the message is authentic. Figure 1 demonstrates the process in a protocol diagram. When the host makes a request for M, the drive reads it from the disc, computes a MAC using the bus key, and sends M in plaintext along with the MAC. Upon receiving the M, the host verifies the MAC and decides whether or not the message is valid.

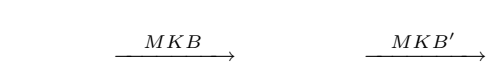
3 Analysis of the AACS Drive-Host Authentication scheme

In this section, we analyze the AACS drive-host authentication scheme. Several weaknesses are identified which could lead to various attacks, and corresponding improvements are provided to strengthen the original scheme.

Our discussion of security is based on the standard security model for authentication and key agreement schemes, which was first proposed by Bellare and Rogaway (1993) in the symmetric-key setting. Blake-Wilson, Johnson, and Menezes (1997) later generalized this model to the public-key setting. In the standard model, the adversary has enormous power and controls all communication between entities. The adversary can read, modify, create, delay and replay messages, and he/she can initiate new sessions at any time.

3.1 Weakness 1: Design Error

This weakness is present in the first four steps of the drive-host authentication scheme. Suppose that the DRL in the MKB is newer than the DRL stored in the host. A malicious party, Oscar, can change the MKB version number to an older one, and send the modified MKB′ to the host. This modification might not be detected during the authentication procedure, because according to the specification, the host first checks the MKB version number, and if the version number is older than its DRL's, it skips over step 2, which involves verifying the signature on the DRL in the MKB.



If the drive has already been revoked, it could maliciously alter the MKB version number in order not to let

the host update its DRL, so that it can keep interacting with the host.

The altered MKB might eventually be detected when the host processes the MKB during content decryption. However, it is undesirable for a revoked drive to be able to talk to the host until then.

The fix to this weakness is simple: The host should verify the MKB and DRL signatures before checking the version numbers. The same modification can be made to the drive side. Figure 2 shows the modification.

3.2 Weakness 2: Unknown Key-Share Attack

A basic requirement for a key agreement protocol is consistency. Consistency requires both honest participating entities having a consistent view of who the intended peers to the session are. For example, let A and B be two honest participating entities trying to set up a shared secret key K through a key agreement protocol. If A establishes K and believes the peer to the session to be B, then when B establishes K it needs to believe that the peer to the session is A, and vice-versa. If a key agreement protocol does not satisfy the consistency requirement, it could be susceptible to the unknown key-share attack.

Let O be an active malicious entity. An unknown key-share attack on a key agreement protocol is an attack through which O causes one of the two honest entities, say A, to believe that it shares a key with O, but it actually shares the key with the other honest entity B, and B believes that the key is shared with A. So, at the end of the protocol, O can act on behalf of B to interact with A. There are a number of papers studying unknown key-share attack and its application on a number of protocols, e.g. (Baek and Kim, 2000; Blake-Wilson and Menezes, 1999; Diffie, van Oorschot, and Wiener, 1992; Kaliski Jr., 2001).

AACS drive-host authentication scheme does not satisfy the consistency requirement. We can simplify the original flow representation of the drive-host authentication scheme displayed in (AACS Specification, Section 4.3) into the one shown in Figure 3 by taking into consideration only the core steps involved in authentication and key agreement. A similar flow diagram is also provided in (AACS Specification, Section 4.3).

1. Host initiates a session with Drive. It sends a random nonce H_n and its certificate H_{cert} to Drive. Drive verifies the signature of the Host certificate using the AACS LA public key. If the verification fails, Drive shall abort this authentication procedure.

2. Drive replies to the Host with a random nonce D_n and its certificate D_{cert}. Host verifies the signature of the Drive certificate using the AACS LA public key. If the verification fails, Host shall abort this authentication procedure.

3. Drive generates a 160-bit random number D_k and uses it to calculate a point D_v on the elliptic curve (G is the base point of the elliptic curve). Drive then creates a signature of the concatenation of H_n and D_v. Drive sends the digital signature and D_v to Host. Host verifies the signature, and aborts the session on failure.

4. Host generates a 160-bit random number H_k and uses it to calculate a point H_v on the elliptic curve. Host then creates a signature of the concatenation of D_n and H_v. Host sends the digital signature and H_v to Drive. Drive verifies the signature, and aborts the session on failure.

On the last step, both Drive and Host calculate the shared secret bus key B_k.

An attacker, $\text{Drive}_{\text{Oscar}}$, which is also a legitimate drive, can use a parallel session to deploy an unknown key-share attack. Figure 4 shows the diagram of the attack.

The attack works in this way:

1. Host initiates a session with $\text{Drive}_{\text{Oscar}}$. It sends its random nonce H_n and certificate H_{cert} to $\text{Drive}_{\text{Oscar}}$.

2. $\text{Drive}_{\text{Oscar}}$ relays the traffic to Drive as if Host is initiating a session with Drive. Drive receives H_n and H_{cert} and verifies that H_{cert} is valid.

3. Drive sends back its random nonce D_n and certificate D_{cert} to Host, which of course get intercepted by $\text{Drive}_{\text{Oscar}}$.

4. $\text{Drive}_{\text{Oscar}}$ relays the random nonce D_n to Host, however, it does not relay the Drive's certificate. Instead, it sends its own certificate D_{O_cert} to Host. Host receives D_{O_cert} as well as D_n. It is tricked into believing that $\text{Drive}_{\text{Oscar}}$ has generated this random nonce. Host verifies $\text{Drive}_{\text{Oscar}}$'s certificate, and the verification should pass because $\text{Drive}_{\text{Oscar}}$ is a legitimate drive.

5. Following the AACS drive-host authentication protocol, Drive generates a random number D_k and calculates a point D_v on the elliptic curve. Drive then creates a signature of the concatenation of H_n and D_v. Drive sends the digital signature and D_v to Host.

6. $\text{Drive}_{\text{Oscar}}$ relays D_v to Host. However, it creates its own signature of the concatenation of H_n and D_v using its private key. It can do so because both H_n and D_v are available to it. It sends this signature instead of Drive's signature to Host. Host verifies the signature using $\text{Drive}_{\text{Oscar}}$'s public key obtained from D_{O_cert}. The verification should pass.

7. Host generates a random number H_k and calculates a point H_v on the elliptic curve. Drive then creates a signature of the concatenation of D_n and H_v. Drive sends the digital signature and H_v to $\text{Drive}_{\text{Oscar}}$.

8. $\text{Drive}_{\text{Oscar}}$ relays the traffic to Drive. Drive verifies the signature, and the verification should pass.

By the time the session is complete, Drive has accepted Host, and it can calculate the shared bus key B_k. On the other hand, Host does not accept Drive because it simply does not know the existence of Drive from this interaction. Instead, it has accepted $\text{Drive}_{\text{Oscar}}$. Host can also calculate the same shared bus key B_k.

Although $\text{Drive}_{\text{Oscar}}$ does not know the secret bus key B_k in the end, it has tricked Host into believing that it shares the bus key with $\text{Drive}_{\text{Oscar}}$. Host thinks that it is talking to $\text{Drive}_{\text{Oscar}}$ while actually it is interacting with Drive.

This attack could be exploited in practice. For example, suppose that Drive_{A} is revoked. Then it can employ this attack to ask Drive_{B}, which is not revoked, to impersonate it. Since the host only sees Drive_{B}'s certificate, the authentication procedure should complete successfully. In this way, Drive_{A} can still interact with the host after the authentication procedure. It has effectively bypassed the authentication procedure.

Such an attack is enabled due to the fact that in the last two flows $\text{Drive}_{\text{Oscar}}$ can simply copy the traffic. To satisfy the consistency requirement, the protocol needs to provide careful bindings between the shared secret key and the entity IDs. For example, the aforementioned problem can be fixed by including the entity IDs in the signature. (See Section 3.4).

3.3 Weakness 3: Man-In-The-Middle Attack

The adversarial goal in an attack to a mutual authentication protocol is to cause an honest participant to "accept" after a flow in which the adversary is active. To consider a mutual authentication protocol secure, it has to satisfy the following two conditions:

1. Suppose A and B are the two participants in a session of the protocol and they are both honest. Suppose also that the adversary is passive. Then A and B will both "accept".

2. If the adversary is active during a given flow of the protocol, then no honest participant will "accept" after that flow.

Figure 5 shows an attack which might not be as powerful and practical as the previous one. Nonetheless, it shows a weakness in this protocol.

In this case, Oscar could be a polynomial time adversary with the ability to listen and to modify the traffic. Notice that in step 2 when Oscar relays the traffic from Drive to Host, it modifies the random nonce D_n generated by Drive into a different one D'_n. This does not make Host terminate the session. In step 3, after Host has successfully verified Drive's signature, it "accepts". This violates condition 2 mentioned above, hence the protocol should not be considered secure.

A moment of reflection regarding this attack reveals that we do not necessarily need the two nonces "H_n" and "D_n".

3.4 Improved Scheme

Since the scheme makes use of certificates, we can improve it using the ISO key agreement protocol. This protocol is a key agreement scheme based on Diffie-Hellman scheme that provides mutual authentication. For more information on the ISO protocol, please refer to (ISO/IEC IS 9798-3, 1993; Krawczyk, 2003).

Figure 6 shows the improved drive-host authentication scheme based on the ISO protocol. This modification solves both problems stated in weakness 2 and 3 (a security proof is given in the next section). In addition, it improves the efficiency of the original protocol, because the number of interactions between Drive and Host is reduced.

1. Host initiates a session with Drive. It generates a 160-bit random number H_k and uses it to calculate a point H_v on the elliptic curve. It sends the H_v and its certificate H_{cert} to Drive. Drive verifies the signature of the Host certificate using the AACS LA public key. If the verification fails, Drive shall abort this session.

2. Drive generates a 160-bit random number D_k and uses it to calculate a point D_v on the elliptic curve. Drive then creates a signature of the concatenation of the Host ID, D_v, and H_v. Drive sends the digital signature, D_v, and its certificate D_{cert} to Host. Host verifies the signature created by Drive: $ver_{drive}(ID_{host}||D_v||H_v, \text{Drive}'\text{s signature}) \stackrel{?}{=}$ true, and it also verifies the signature of the Drive certificate. If any of the two verifications fail, Host shall abort the session.

3. Host creates a signature of the concatenation of the Drive ID, H_v, and D_v and sends it to Drive. Drive verifies the signature: $ver_{host}(ID_{drive}||H_v||D_v,$ Host$'$s signature$) \stackrel{?}{=}$ true, and aborts the session on failure.

At the end of the protocol, both Drive and Host are able to establish the shared secret bus key B_k. Points H_v and D_v in this protocol also play a role as random challenges.

The new protocol solves all the aforementioned problems. Since the random challenges H_n and D_n are omitted, it enables the drive and the host to perform fewer interactions, and is therefore more efficient.

Appendix B shows a flow representation of the entire improved drive-host authentication protocol.

4 Security of the Improved Drive-Host Authentication Scheme

The improved scheme protects against the unknown key-shared attack mentioned earlier.

In Figure 7, a question mark following a signature indicates that the adversary is unable to compute this signature. At step 3, the signature which Host sends to $\text{Drive}_{\text{Oscar}}$ contains $\text{Drive}_{\text{Oscar}}$'s ID not Drive's ID

because Host believes that it is talking to $\text{Drive}_{\text{Oscar}}$. $\text{Drive}_{\text{Oscar}}$ cannot compute Host's signature on the string $ID_{drive}||H_v||D_v$ because he does not know Host's private signing key. As a result, unknown key-share attack is thwarted.

After step 2, Host "accepts" the authentication because it should successfully verify $\text{Drive}_{\text{Oscar}}$'s signature and certificate. This does not violate the second condition of considering a mutual authentication protocol secure mentioned in Section 3.3, because Host is authenticating with $\text{Drive}_{\text{Oscar}}$.

The improved scheme also protects against man-in-the-middle attack.

As shown in Figure 8, if Oscar modifies H_v, he then would not be able to produce Host's signature on $ID_{drive}||H'_v||D_v$ because he does not know Host's private signing key. Likewise, if Oscar modifies D_v, he then would not be able to produce Drive's signature on $ID_{host}||D'_v||H_v$ because he does not know Drive's private signing key.

Of course, we want to show that the improved scheme is secure against all possible attacks, not just two particular attacks. Hence, we need to show that the improved scheme is a secure mutual authentication scheme, and that it provides assurances regarding knowledge of the shared secret key. For the proof of security of our improved scheme, an informal treatment based on (Stinson, 2002, Chapter 11) is given in the rest of this section.

4.1 Secure Mutual Authentication

A secure mutual authentication has to satisfy the two conditions described in Section 3.3. Let us first show that our improved scheme satisfies the first condition.

Since no one is modifying the traffic, if the adversary is passive and the two participants are honest they should successfully authenticate themselves to each other and both compute the shared secret key as in the Diffie-Hellman key agreement scheme. Assuming the intractability of the *Decision Diffie-Hellman* problem, the inactive adversary cannot compute the share secret key.

To prove that our improved scheme satisfies the second condition, let us assume that the adversary is active. The adversary wants to fool at least one of the two participants to "accept" after a flow in which he is active. We show that the adversary will not scceed in this way, except with a very small probability.

Definition 1. A signature scheme is (ϵ, Q, T)-secure if the adversary cannot construct a valid signature for any new message with probability greater than ϵ, given that he has previously seen at most Q different valid signatures, and given that his computation time is limited to T.

Definition 2. A mutual authentication scheme is (ϵ, Q, T)-secure if the adversary cannot fool any honest participants into accepting with probability greater than ϵ, given that he has observed at most Q previous sessions between the honest participants, and given that the his computation time is at most T.

Time T is usually chosen to be very long so that by the time the adversary successfully computes the correct result the value of the result has decreased to an insignificant level. For simplicity of notation, we omit the time parameter. Q is a specified security parameter. Depending on the application, it could be assigned with various values. The probability ϵ is usually chosen to be so small that the chance of success is negligible.

Theorem 1. *Suppose that Sig is an (ϵ, Q)-secure signature scheme, and suppose that random challenges H_v and D_v are k bits in length. Then the scheme shown in Figure 6 is a $(Q/2^{k-1} + 2\epsilon, Q)$-secure mutual authentication scheme.*

Proof. The adversary, Oscar, observes Q previous sessions of the protocol before making his attack. A successful attack by Oscar is to deceive at least one honest participant in a new session into accepting after he is active in one or more flows.

1. Oscar tries to deceive Host. In order to make Host accept, it has to receive a signature signed by Drive containing the Host ID and the random challenge H_v. There are only two ways for Oscar to acquire such a signature: either from a previously observed session or by computing it himself.

   To observe such a signature from a previous session, H_v has to be used in that session. The probability that Host has already used the challenge in a specific previous session is $1/2^k$. There are at most Q previous sessions under consideration, so the probability that H_v was used as a challenge in one of these previous sessions is at most $Q/2^k$. If this happens, Oscar can re-use Drive's signature and D'_v (which may or may not be the same as D_v) from that session to fool Host.

   To compute such a signature himself, Oscar has at most a chance of ϵ, since *Sig* is (ϵ, Q)-secure.

   Therefore, Oscar's probability of deceiving Host is at most $Q/2^k + \epsilon$.

2. Oscar tries to deceive Drive. This is quite similar to the case we have discussed above. In order to fool Drive, Oscar has to have a legitimate signature signed by Host. As in the previous case, the two ways for Oscar to acquire such a signature are either from a previously observed session or by computing it himself.

   To observe such a signature from a previous session, Oscar re-uses a H_v from a previous session S to send to Drive, and hopes that Drive will reply with the same D_v as in S so that he can re-use the corresponding signature. This happens with probability $1/2^k$. The best case scenario for the adversary would be that all Q previously observed sessions have the same H_v. Because if any D_v from the Q sessions is re-used by Drive, Oscar can then re-use the corresponding signature to fool Drive. Hence, Oscar has at most $Q/2^k$ probability to re-use Host's signature to deceive Drive.

Again since Sig is (ϵ, Q)-secure, Oscar can compute such a signature with a probability of at most ϵ.

Therefore, Oscar's probability of deceiving Drive is at most $Q/2^k + \epsilon$.

Summing up, the probability for Oscar to deceive one of Host or Drive is at most $(Q/2^k+\epsilon)+(Q/2^k+\epsilon) = Q/2^{k-1}+2\epsilon$. □

4.2 Implicit Key Confirmation

Now, let us see what we can infer about the improved scheme if Host or Drive "accepts". Firstly, suppose that Host "accepts". Because the improved scheme is a secure mutual authentication scheme, Host can be confident that it has really been communicating with Drive and that the adversary was inactive before the last flow. Assuming that Drive is honest and that it has executed the scheme according to the specifications, Host can be confident that Drive can compute the value of the secret bus key, and that no one other than Drive can compute the value of the bus key.

Let us consider in more detail why Host should believe that Drive can compute the bus key. The reason for this belief is that Host has received Drive's signature on the values H_v and D_v, so it is reasonable for Host to infer that Drive knows these two values. Now, since Drive is a honest participant and executed the scheme according to the specifications, Host can infer that Drive knows the values of D_k. Drive is able to compute the value of the bus key, provided that he knows the values of H_v and D_k. Of course, there is no guarantee to Host that Drive has actually computed the bus key at the moment when Host "accepts". We can be sure that no one else can compute the bus key because D_k is meant to be known to Drive only.

The analysis from the point of view of Drive is very similar. If Drive "accepts", then it is confident that it has really been communicating with Host, and that the bus key can be computed only by Host and no one else.

The improved scheme does not make immediate use of the new bus key, so we do not have explicit key confirmation. However, it does achieve implicit key confirmation. Moreover, it is always possible to augment any key agreement scheme with implicit key confirmation so that it achieves explicit key confirmation, if so desired. In essence, the improved scheme provides authenticated key agreement with key confirmation.

5 Improved Scheme Based on the SIGMA Protocol

The improved scheme based on the ISO protocol is susceptible to the following situation shown in Figure 9. The malicious man-in-the-middle consists of two entities, a legitimate host Host_O and a legitimate drive Drive_O. The result of the involvement of this man-in-the-middle is that Drive believes that it shares the key with Host_O while it actually shares the key with Host, and Host thinks that it shares the key with Drive_O but it actually shares the key with Drive. However, this does not contradict the consistency requirement provided by the ISO protocol, because the intended peer of Drive is really Host_O, and the intended peer of Host is Drive_O. There are in fact two sessions established in this attack, one between Drive and Host_O and one between Host and Drive_O. Thus, Host_O and Drive_O are participating entities in their corresponding sessions of the protocol. However, they are not honest. There is no guarantee of security in a session of a protocol with dishonest participating entities. Therefore, this is not an attack on the scheme. Nonetheless, the scenario illustrated in Figure 9 may be a concern in practice.

A variant of the SIGMA protocol (Krawczyk, 2003) which is very similar to the ISO protocol prevents this situation by introducing a message authentication code (MAC) on the IDs and the Diffie-Hellman parameters in the signatures, as shown in Figure 10. The key K_m used in the MAC is computed from a one-way function which takes as input the shared secret $D_k H_k G$. As a result, the SIGMA protocol provides explicit confirmation on the shared secret, which means that the intended peer must know the shared secret. Therefore, the previous scenario illustrated in Figure 9 cannot occur on the SIGMA protocol. The bus key B_k is computed from a different one-way function which also takes $D_k H_k G$ as input. Both the MAC key and the bus key are derived from the shared secret, and yet they are "computationally independent". This improved scheme based on the SIGMA protocol provides implicit key confirmation. It is also plausible to say that it provides explicit key confirmation if the "key" refers to the shared secret rather than the bus key. Security proofs of the SIGMA protocol and its variants can be found in (Canetti and Krawczyk, 2002).

The implementation of the improved scheme based on the SIGMA protocol requires extra cryptographic primitives such as MACs and one-way functions to be added to the original AACS drive-host authentication scheme. Nevertheless, it achieves careful binding between the entity ID and the shared secret key. On the other hand, the implementation of the improved scheme based on the ISO protocol is easier and requires nothing to be added to the original scheme.

6 Conclusion

We have described three weaknesses in the AACS drive-host authentication scheme. Specifically, the scheme is susceptible to unknown key-share attack and man-in-the-middle attack. As a goal to improve the scheme to resist all kinds of attacks, we have modified the original scheme based on the ISO and the SIGMA key agreement protocol to provide secure mutual authentication as well as authenticated key agreement with key confirmation. In addition, our modified scheme achieves better efficiency than the

original scheme.

Acknowledgment

We would like to thank Ian Goldberg for his constructive comments and suggestions on this work, especially his insight on the SIGMA protocol.

REFERENCES

AACS LA (2008), *Advanced Access Content System (AACS) - Introduction and Common Cryptographic Elements, Revision 0.91.* `http://www.aacsla.com/specifications/specs091/AACS_Spec_Common_0.91.pdf`. [accessed 06/06/2008]

Baek, J. and Kim, K. (2000) 'Remarks on the Unknown Key Share Attacks', *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E83-A, No. 12, pp. 2766-2769.

Bellare, M. and Rogaway, P. (1993) 'Entity Authentication and Key Distribution', *CRYPTO 1993*. LNCS, Vol. 773, pp. 232-249.

Blake-Wilson, S., Johnson, D. and Menezes, A. (1997) 'Key Agreement Protocols and Their Security Analysis', *Proceedings of the Sixth IMA International Conference on Cryptography and Coding*. LNCS, Vol. 1355, pp. 30-45.

Blake-Wilson, S. and Menezes, A. (1999) 'Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol', *Proceedings of PKC 1999*. LNCS Vol. 1560, pp. 154-170.

Canetti, R. and Krawczyk, H. (2002) 'Security Analysis of IKE's Signature-based Key-Exchange Protocol', *Crypto 2002*. LNCS Vol. 2442, pp. 27-52.

Diffie, W., van Oorschot, P.C. and Wiener, M.J. 'Authentication and Authenticated Key Exchanges', *Designs, Codes and Cryptography*, Vol. 2, Issue 2, pp. 107-125.

ISO/IEC IS 9798-3 (1993), 'Entity Authentication Mechanisms - Part 3: Entity Authentication Using Asymmetric Techniques'.

Kaliski Jr., B.S. (2001) 'An Unknown Key-Share Attack on the MQV Key Agreement Protocol', *ACM Transactions on Information and System Security*, Vol. 4, No. 3, pp. 275-288.

Krawczyk, H. (2003) 'SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols', *CRYPTO 2003*. LNCS, Vol. 2729, pp. 400-425.

National Institute of Standards and Technology (2007), *Special Publication 800-56A, Recommendation for Pair-Wise Key Establish Schemes Using Discrete Logarithm Cryptography.*

Stinson D.R. (2006) *Cryptography Theory and Practice, Third Edition*, Chapman & Hall/CRC.

A Original Drive-Host Authentication Scheme

The AGID sent in step 5 is the Authentication Grant Identifier, which is used to identify a specific session in the case where a drive can support connections from multiple hosts. H_{cert} is the host certificate, and D_{cert} is the drive certificate.

	Drive		Host
1.		$\xrightarrow{MKB}$	Compare version of stored DRL to DRL in MKB. If DRL in MKB is not newer, use stored DRL.
2.			If DRL in MKB is newer, verify DRL signature. Use DRL in MKB. Abort if signature is not valid.
3.	Compare version of stored HRL to HRL in MKB. If HRL in MKB is not newer, use stored HRL.		
4.	If HRL in MKB is newer, verify HRL signature. Use HRL in MKB. Abort if signature is not valid.		
5.		$\xrightarrow{AGID}$	
6.			Generate 160-bit nonce H_n.
7.		$\xleftarrow{H_n,\ Hcert}$	
8.	Verify host certificate type and length. Abort on failure.		
9.	Verify signature on host certificate. Abort on failure.		
10.	Check HRL and abort if Host ID is found.		
11.			Request nonce D_n and drive certificate.
12.	Generate 160-bit nonce D_n.		
13.		$\xrightarrow{D_n,\ Dcert}$	
14.			Verify drive certificate type and length. Abort on failure.
15.			Verify signature on drive certificate. Abort on failure.
16.			Check DRL and abort if Drive ID is found.
17.			Request a point on the elliptic curve D_v and its signature.
18.	Generate 160-bit random value D_k.		

19.	Calculate $D_v = D_k G$ where G is the base point of the elliptic curve.		
20.	Calculate D_{sig} as the signature of $H_n \| D_v$ using the drive's private key.		
21.		$D_v,\ D_{sig}$ →	
22.			Verify D_{sig} and abort on failure.
23.			Generate 160-bit random number H_k.
24.			Calculate $H_v = H_k G$.
25.			Calculate H_{sig} as the signature of $D_n \| H_v$ using the host's private key.
26.		← $H_v,\ H_{sig}$	
27.	Verify H_{sig} and abort on failure.		
28.	Calculate Bus Key B_k as the 128 least significant bits of x-coord$(D_k H_v)$.		
29.			Calculate Bus Key B_k as the 128 least significant bits of x-coord$(H_k D_v)$.

B Improved Drive-Host Authentication Scheme Based on the ISO Protocol

	Drive		Host
1.		$\xrightarrow{MKB}$	Verify MKB and DRL signatures. Abort if signatures are not valid.
2.			Compare version of stored DRL to DRL in MKB. If DRL in MKB is not newer, use stored DRL. Otherwise, use DRL in MKB, and store it for later reference.
3.	Verify MKB and HRL signatures. Abort if signatures are not valid.		
4.	Compare version of stored HRL to HRL in MKB. If HRL in MKB is not newer, use stored HRL. Otherwise, use HRL in MKB, and store it for later reference.		
5.		$\xrightarrow{AGID}$	
6.			Generate 160-bit random number H_k.
7.			Calculate $H_v = H_k G$ where G is the base point of the elliptic curve.
8.		$\xleftarrow{H_v,\ Hcert}$	
9.	Verify host certificate type and length. Abort on failure.		
10.	Verify signature on host certificate. Abort on failure.		
11.	Check HRL and abort if Host ID is found.		
12.			Request a point on the elliptic curve D_v, signature, and drive certificate.
13.	Generate 160-bit random value D_k.		
14.	Calculate $D_v = D_k G$ where G is the base point of the elliptic curve.		
15.	Calculate D_{sig} as the signature of $ID_{host} \| D_v \| H_v$ using the drive's private key.		

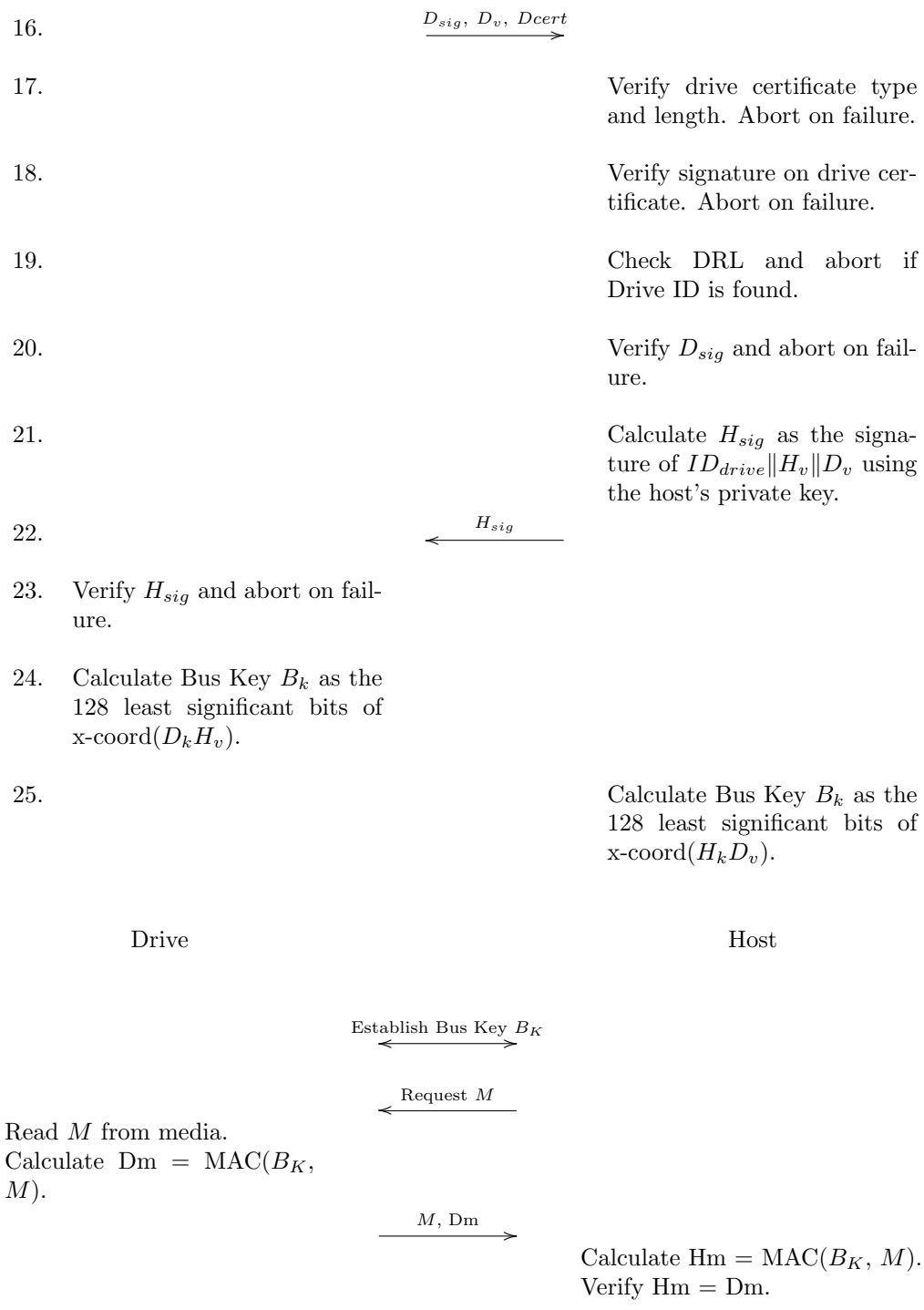


Figure 1: Protocol for transferring media-specific information

	Drive		Host
1.		$\xrightarrow{MKB}$	Verify MKB and DRL signatures. Abort if signatures are not valid.
2.			Compare version of stored DRL to DRL in MKB. If DRL in MKB is not newer, use stored DRL. Otherwise, use DRL in MKB, and store it for later reference.
3.	Verify MKB and HRL signatures. Abort if signatures are not valid.		
4.	Compare version of stored HRL to HRL in MKB. If HRL in MKB is not newer, use stored HRL. Otherwise, use HRL in MKB, and store it for later reference.		

Figure 2: Improved first four steps

	Drive		Host
1.		$\xleftarrow{H_n,\ H_{cert}}$	
2.		$\xrightarrow{D_n,\ D_{cert}}$	
3.	$D_v{=}D_kG$	$\xrightarrow{Sig_{drive}(H_n\|\|D_v),\ D_v}$	
4.		$\xleftarrow{Sig_{host}(D_n\|\|H_v),\ H_v}$	$H_v{=}H_kG$
5.	$B_k{=}D_kH_v{=}D_kH_kG$		$B_k{=}H_kD_v{=}H_kD_kG$

Figure 3: Simplified AACS drive-host authentication protocol

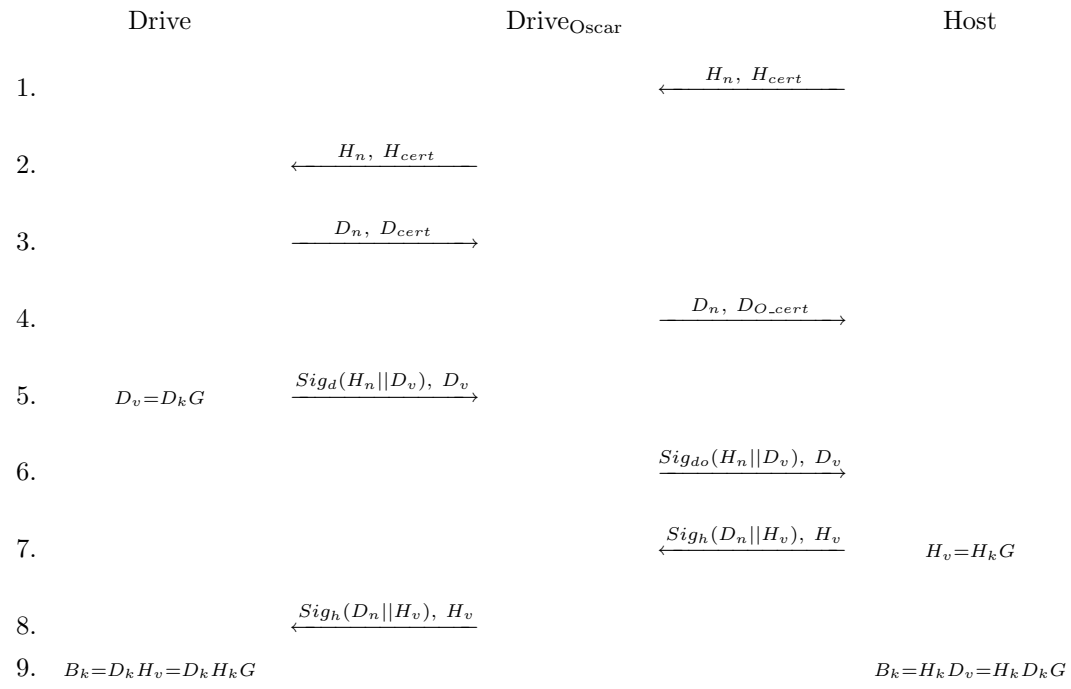


Figure 4: Unknown key-share attack on AACS drive-host authentication protocol

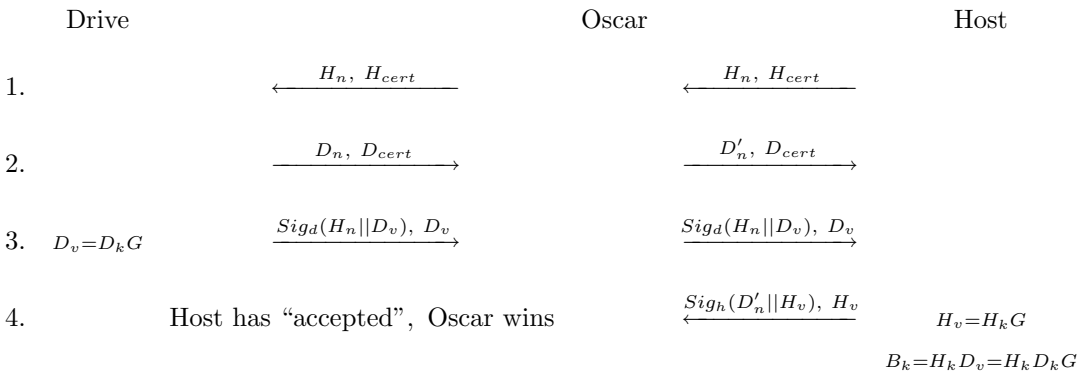


Figure 5: A trivial man-in-the-middle attack

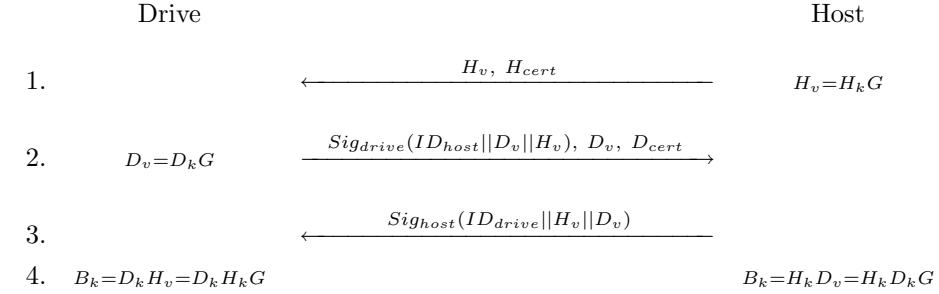


Figure 6: Improved scheme based on the ISO protocol

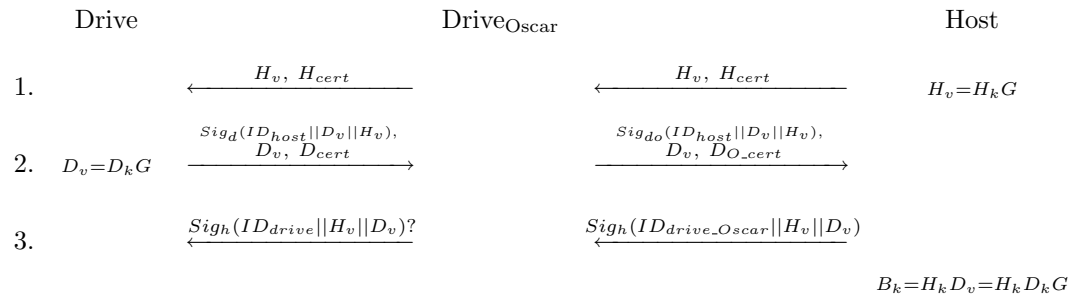


Figure 7: Protection against unknown key-share attack

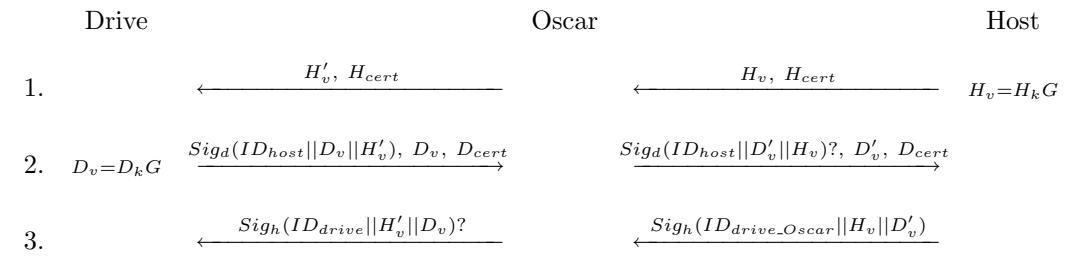


Figure 8: Prevention of man-In-the-middle attack

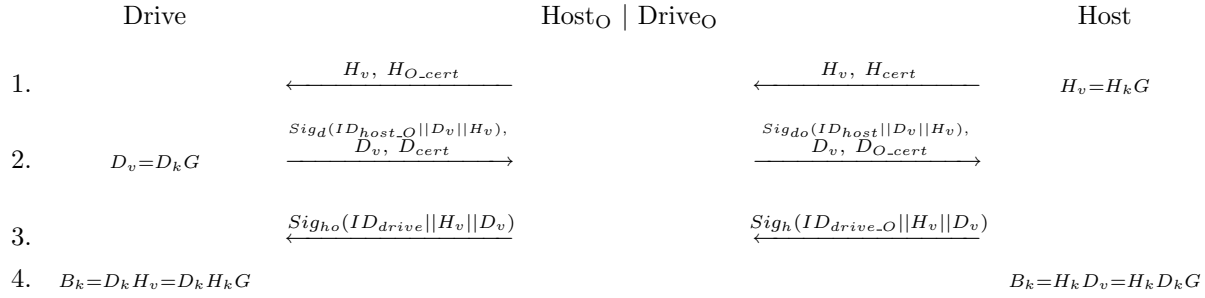


Figure 9: A concern with the ISO protocol

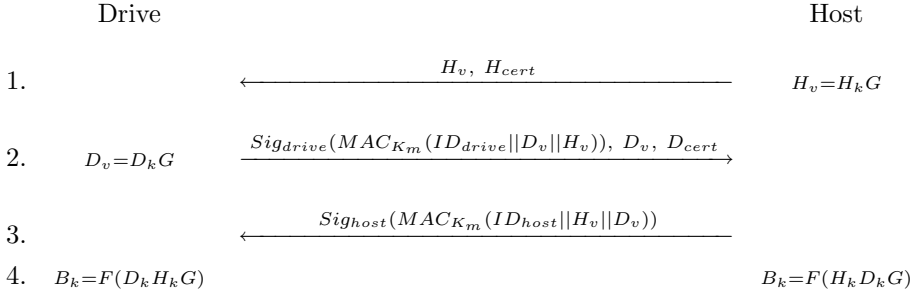


Figure 10: Improved scheme based on the SIGMA protocol