

Four Simple Cryptographic Attacks on HDCP

by Keith Irwin (KeithIrwin@yahoo.com).

Introduction:

Intel has developed a new standard for device authentication and digital content encryption. It was developed specifically for encrypting uncompressed digital video on its way to a monitor (either a computer monitor or a television) specifically with DVI (digital video interface). It is, however, general enough to be used for other forms of digital information. The standard is available from [Digital Content Protection, LLC](#), a corporation formed to act as a holding company for the standard. My sources have told me that the MPAA has been pressuring the direct satellite services (DirecTV and the DishNetwork) to require that all movies be transmitted from the satellite set-top box to the monitor via DVI protected with HDCP. They believe it to be more secure than the existing high-definition analog connections. I respectfully must disagree.

Because the standard is freely available, I am not, in this paper, going to repeat or summarize the contents of the standard except when it seems immediately necessary. Rather, I shall usually talk about specific properties or qualities which it has which lead to certain attacks being useful against it. That the standard has the properties which I ascribe to it can be verified by reading it.

In normal cryptography an attack would specifically be a manner of determining the content of an encrypted message. In digital content protection the approximate content of the message is known by the viewer, however, the owners of the content want to keep the viewer from being able to record or share or otherwise reuse the digital version of the content. So in this circumstance, an attack is any means of either recovering the unencrypted message or of reusing or sharing it without unencrypting it. As it happens, both of these things are possible with HDCP encoded messages, as I will outline. Specifically, the attacks outlined are described as simple because they do not involve compromising the physical integrity of any of the devices involved or cracking the fundamental cipher of the system. This should not be taken as an endorsement of the fundamental cipher. It may be quite vulnerable to cryptanalysis, and you may see a later paper from me entitled "One Complex Attack on HDCP". Nonetheless, the available attacks seem to me sufficiently fundamental and important to detail here.

Definitions:

The transmitter is the device which generates the video signal, such as the set-top box, a video cassette recorder, a digital video recorder, or a computer. The receiver is the device which receives and displays the video signal, such as a monitor or television equipped with the appropriate connection. A repeater is a device which sits in between the transmitter and one or more receivers which unencrypts and reencrypts the digital feed appropriately, acting as both a transmitter and a receiver.

Notation: (I'm sticking with ASCII for this paper for the moment, so I feel I should be clear)

_ is subscript, i.e. x_1 is to be read as a variable x with a subscript 1.

^ is exponentiation, i.e. x^n is x raised to the n th power.

* is multiplication, i.e. $a * b$ is a times b .

Assumptions:

The following assumptions are made:

- 1) The digital stream between the two devices can be recorded. If this is impossible then there is not really any point in bothering to encrypt it to begin with.
- 2) It can be tapped without either device being aware of this. If this is impossible, then again, encryption is unnecessary.
- 3) The transmitter can be made to transmit a purely black or other precisely known signal (such as a menu display, for instance) for an arbitrary amount of time.

4) The standard will be known to or adequately guessed by at least some potential attackers. Since it has been made available to anyone with web access at this point, I think that this is a given.

The First Attack - Just Record

The first attack is a very simple one. The receiver does not in any way verify the validity or computation abilities of the transmitter and does not contribute anything to the cryptographic function which differs from session to session. Consequentially, should an attacker record the complete communications of the transmitter, the attacker can then at some arbitrary later time play it back to the same monitor and the monitor will display it. The playback device would have to have a little bit of sophistication since there are times when the transmitter must wait for a response from the receiver, but that is only a timing issue; no new information of any sort would need to be generated. This would allow one to compile a video collection. Further, the receiver in question does not need to be a monitor, it could be a repeater, in which case, the attacker could say take the recorded video over to a friend's house along with the repeater and use it to display the video on any monitor there available or even potentially sell the recorded content to someone else far away by shipping them both the recording of the information and the repeater together.

What this attack does not allow for is the possibility of realistically exchanging video over the internet both because the information is tied to a particular receiver and because it would be difficult to efficiently compress the still-encrypted signal.

The Second Attack - XOR with Black

HDCP encrypts the signal by XORing it with the output of a cipher function. The cipher function is, of course, deterministic. There is precisely one quantity which is unique to a particular session, which is a pseudo-random number, A_n . All other inputs to the cipher function do not vary so long as the same pair of devices is connected. So, if an attacker can get the transmitter to generate the same A_n in two cases, one can then XOR the resultant bit streams and get a result equal to the XOR of the two unencrypted bit streams. Our third assumption is that the transmitter can be made to transmit nothing but black or some other known signal. So, as long as the attacker can cause the same A_n to be generated, she can first record the bitstream associated with total blackness or other known signal and then the one which she wishes to decrypt, she can then XOR them and the result will be the unencrypted bit stream from the desired program XORed with the known signal and then by XORing out the known signal, you recover the original (in the case of pure black, the latter part is probably not even necessary since pure black should most likely be all zeros).

Clearly, the relevant question is, "How can she guarantee that the same A_n will be generated?" As it happens, the value of A_n is determined using the amount of time between the transmitter powering on and the receiver requesting that the authentication begin (which should generally happen immediately after the connection is established). So one would need to build a device with a digital timer which controls both the timing of the connection of the power to the device and the timing of the connection of the devices to each other. So long as the timing of these events is consistent, then the same A_n will be generated, and the unencrypted stream is easily attained.

The Third Attack - Offline Cracking

This attack relies on both the third and fourth assumptions. All information needed to decode with the exception of a 56 bit key particular to the pair of devices connecting is transmitted between the two devices. By recording a few seconds of video one could then set computers to trying all 2^{56} possible keys simply by doing the same calculations that the chip would do. Certainly this is a long process, but it has been demonstrated that keys of only fifty-six bits can be cracked in the real world. Once it has been cracked once, all communications between the two devices in question can be completely unencoded. Although the standard contains the possibility of listing particular devices as no longer being secure, there is nothing which would indicate any way for the authority to know when a device has been compromised by offline cracking.

The Fourth Attack - Complete System Breaking

The fourth attack builds off of the third attack, using repeated instances of the third attack to break the entire system.

Every individual device has a unique 40 bit key-selection vector (KSV), which has twenty 1's and twenty 0's. It also has a unique table of forty 56-bit keys which is particular to the KSV. When two devices connect, each publically relays to the other its KSV. Then each combines the other's KSV with its own table. They sum up entries from their tables modulo 2^{56} according to the presence or absence of 1's. That is, if the first bit of the KSV is 1 then the first 56-bit table entry is added in, and if it is 0 then the first 56-bit table entry is not added to the sum. The second through fortieth bits are, of course, treated likewise. Because of the way the tables and KSV's are calculated and assigned, it is the case that the two 56-bit numbers generated are mathematically guaranteed to be equal.

So, let us then use the third attack to discover one of these keys (discover it the painful way, certainly, but discover it no less). What have we learned? Essentially, we now have information equivalent to one fortieth of the table of each of these two devices, but not in an immediately useful form. So, should we break forty different of these keys for a particular device, we may have access to the complete information in the table. Except that realistically, forty randomly chosen KSV's are quite unlikely to be linearly independent, meaning that some of them will not yield new information. Consider instead that we have enough devices to have a linearly independent set of forty KSV's. I did a statistical sampling of 1000 matrices to approximate how many KSV's would be needed on average. The result of such was 72.4, so roughly 73 KSV's are needed. So let us assume that we have 74 or so devices which we can potentially crack. Each device has its own KSV. KSV's are not shared amongst devices of the same make or model. We can then determine each device's KSV and assuming that the KSV's are randomly distributed easily select a set of 40 of them which are linearly independent. Clearly we can brute-force crack the keys of these as in attack 3, so let us assume that we have forcibly cracked the connection between one other device and the set of 40 devices with linearly independent KSV's.

We can now reasonably represent the information which we have as a set of forty linear equations. Should we solve this, we find can then recover our forty unknowns which make up the table of secret information in the device. There is really only one difficulty hampering us in solving this problem, which is that we were not working with normal math, but rather with modulo 2^{56} math. Fortunately, this is not a serious problem. Addition, subtraction, and multiplication obey the same rules as normal and so can be treated normally. Division almost does. The only difference is when one must divide by 2. When division by 2 is required, ambiguity is caused, since there are two different values modulo 2^{56} which when doubled will result in the given number. What this effectively means is that each time we have to divide by 2 we lose the left most bit of the result. So, keeping that in mind, let us proceed in a normal manner by expressing our set of linear equations in terms of matrix algebra. $M * x = c$, where M is the matrix formed by the KSV's, x is our vector of unknowns, and c is our vector of cracked keys. We know that the inverse matrix of M (noted as M^{-1}) can be expressed as the adjoint matrix of M (which requires no division to compute and is noted as $\text{adj}(M)$) divided by the determinant of M (which also requires no division to compute and is noted as $\text{det}(M)$). So we multiply both sides by this on the left and get $M^{-1} * M * x = (1/\text{det}(M)) * \text{adj}(M) * c$ which yields $x = (1/\text{det}(M)) * (\text{adj}(M) * c)$, which, if it is not clear, says that the vector representing our unknowns is equal to a scalar constant $(1/\text{det}(M))$ times the vector formed by multiplying the adjoint matrix by the vector of our cracked keys. So, we now have values for our secret information precisely, except that we have to divide by our determinant. So, we are going to lose from each number of the secret information a number of bits equal to the largest power of two which divides the determinant of M (which I will temporarily call n). We are not going to attempt to reclaim this missing information, but rather use the results which we do have. So now when one has a new device that one wishes to use, one just uses the information we have to compute the proper key except for its n leftmost bits. The obvious question is how big is n ? If n is quite large (especially if it were very near or larger than 56) then this attack would be useless since we've only narrowed down our new secret key to 2^n different values. However, again I turned to a statistical sampling of 1000 linearly independent matrices which had the appropriate properties. The average largest power of two in the determinant of these matrices was 3.6. So, in the average case, we lose 4 bits, meaning that we have, on average, successfully narrowed any new key down to 16 possibilities, few enough even to try by hand if necessary.

The steps above compromise a particular device completely. Having done this, if we were given the KSV of any other device, we can easily find the proper encryption key which is used when connecting to it. So, let us assume now that a set of forty devices with linearly independent KSV's have each been completely compromised through our method. Let us also say that we have a new device which we've never seen before, but we know its KSV. We can then immediately calculate secret keys for it (minus four bits) which correspond to forty linearly independent KSV's and apply the methods from the previous paragraph to recover its secret information (minus eight bits). So, without ever having done anything more to it physically than determining its KSV, we can calculate the secret key that it would use for connecting to any arbitrarily chosen other

device except for the leftmost eight bits, on average. These missing eight bits mean $2^8 = 256$ possibilities, certainly few enough that one could do it by hand if necessary, and a properly built device could do find the correct key in a matter of seconds. Also, the forty devices with linearly independent KSV's would potentially be compromised using a different method, such as information being leaked by the manufacturer or the device being physically broken into, in which case we'd only be missing four bits at this stage.

What this means, in summary, is that if 40 properly chosen devices each had 40 properly chosen keys cracked, then we can compute the encryption key used for any given connection to within a smaller number of possibilities, and, essentially, the whole system fails to provide any protection. This is not really what one would call robust. We are talking about a system involving millions of devices, but even when randomly selected only an average of 73 or so need be cracked to destroy the effectiveness of the entire system. With proper selection and without the aid of corporate leaks or hardware taps this would require 820 cracks of 56-bit keys, so it would not, for instance, happen the day that the system was released. But it does mean that realistically the system has a maximum effective lifetime. In as little as a few years, likely after everyone had locked into the system, the whole thing would become ineffective.

In case I lost anyone, the reason that the above number is 820 and not 1600 is that once one device is cracked, it can be used to crack one line from any device, including the others that you are trying to crack. Consequentially, we have to break 40 keys by brute force the first time, but only 39 the second, 38 the third, and so forth. So the number of total brute force attacks is $(40*41)/2 = 820$.

Also, it means that the presence of a list of revoked KSVs is virtually useless, because should more than an average of 73 become compromised, the whole system is compromised. Adding a particular device to the list may stop it from being an immediate security leak, but it does not stop its information from being used to crack the whole system.

I glossed over one detail above, and I would like to take a moment to touch on it for the sake of completeness and to explain why I glossed over it. The mathematically astute amongst you may have noticed that I assumed that a set of KSV's which comprise a basis for the space of forty dimensional vectors could be obtained. It is possible that only a subset of the full KSV space could be issued and that that subset would not contain a basis for the entire space. I am not dealing greatly with this issue because there is no indication in the standard that this would be the case and because I do not believe that this change would make the problem significantly more difficult. Instead of taking forty linearly independent vectors, one would take a set sufficient to comprise a basis of the subspace being issued, and then solve for a more convenient expression of that basis similar to how we previously solved for the identity matrix. This would be a little more of a hassle, but should be quite possible, and although I do not have a proof of such, I am going to assert that it would not require more divisions by two (that is, losses of bits) on average to do so. So, hopefully that successfully clears the issue up for anyone who may have been wondering why I had not addressed that possibility.

Addendum:

I am here inserting an addendum to this attack. I also made some corrections above, fixing some typos and changing the 1600 to 820 and added the explanation of why. That was a fairly minor change, so I did not make a big fuss about it. There is also something else that I missed the first time through, which is that there's an even better way to assemble the necessary keys for the attack. The third attack is about breaking some particular key. But we don't need to break some particular set of keys, we need 40 arbitrary linearly-independent keys. So what we can do is to use a birthday paradox style attack.

There is another assumption necessary for such, which is that we can feed nonsense to the monitor and that this will not result in the HDCP chip disconnecting. From reading the specification, there is no indication that there is any way for the video hardware to signal the HDCP module, so I feel this assumption justified. The way this attack works is that we fix a pseudo-random A_n and then feed a set of random KSVs to the receiver and see what it gives back. Then we simulate a set of random 56-bit session keys with the same A_n and see what they predict that the receiver would give back. Then we see if they overlap. Overlaps indicate that we have almost certainly matched a KSV with a session key. To be absolutely certain, for any discovered overlap we can further get as many bits from either source as we want to ensure the match to whatever standard we should desire, or we could give them a different A_n and ensure that they match given that one as well.

You will note that this requires that several things be true. First, it is required that as outlined in the first attack, the receiver

does no authentication of the transmitter. Second, it must be the case that the encryption function has no way to know that it is being fed gibberish. Since it's a simple one-time pad algorithm which XOR's the signal with a pseudo-random stream, it cannot. Third, the assumption mentioned in the previous paragraph is required, since we have no actual encrypted signal to feed to it. As a result, what comes out of the HDCP decoder will almost certainly be gibberish to the display circuitry. Fourth, it is needed that the input of the cryptographic function not include the KSV. It does not. Fifth, we require that the receiver (which mind you is a television or other display device) send some output back to the transmitter. It does. However, it only sends out 16 bits every 2 seconds. The purpose of these 16 bits are to provide continuing authentication of the receiver, but they will serve adequately well for our purpose.

So, we've establish that we can do this sort of attack, but what I have not yet addressed how large the two sets need to be. Unless my math is mistaken, the expected number of intersections between two randomly chosen subsets of a larger set is the product of the size of the subsets divided by the size of the larger set. So we need to select two sets such that the product of their sizes would be $73 * 2^{56}$ which I'm going to generously round up to 2^{63} . So, for instance, one set being 2^{31} and the other being 2^{32} would be quite likely to work. However, for real world speed, it is probably wiser to disbalance the sets more, making the one which depends on real hardware smaller since it will be much slower going. If, for instance, we compile the hardware-based set first and take 320 bits from the receiver to be sure, we could compile 129,600 KSVs per month, or roughly 2^{17} . That would then lead us to 2^{46} simulations to run, which we could potentially even do in a distributed manner if it is slow going. Alternately, if we compile the simulated ones first (which would require much more space to store) then we would not have to take 320 bits. We could take only enough bits to exclude known values (which would average 48 bits or less, i.e. 6 seconds) and therefore do only a third as many simulated values and still likely come up with enough within a month. But either way we chose, a particular device can probably be broken in a month or less, and as we can work on multiple devices in a parallel manner, the whole system could be broken by devoted efforts in just one month or less.

Conclusions:

HDCP is not the worst cryptographic system ever. Although this paper does not really show it, there were some interesting ideas that went into this system, and it is a serious and legitimate attempt at solving the problems it seeks to address. However, it does contain fundamental vulnerabilities which render the system unfit for the job for which it is proposed.

I would like to take a moment to thank Intel and Digital Content Protection, LLC for releasing this specification to the public so that it can be reviewed by independent researchers. It is rare that cryptographic standards intended for use in consumer devices have been publically released. It is my sincere hope that more companies will follow their lead and give themselves a chance deal with problems in their copy-protection standards before release instead of attempting to plug holes through lawsuits brought under the Digital Millennium Copyright Act against those who exploit the holes.

Special Thanks:

I would like to especially thank Matt Hoover and Ryan Ingram for aiding me when my mathematics knowledge was failing me, and Ryan Ingram again and Ross Cohen for help in debugging the program I wrote to find the needed statistics, and John Prevost for actually running it for me when my system refused to work correctly. Also, thanks to the others who volunteered to help out and to my other friends and family for listening to me yammer about this for the last couple months. And a very big special thanks goes to everyone who has helped by proof-reading and giving me feedback on my drafts.

Addendum Thanks:

Thanks to John Prevost and Michael Poole for helping with the math, and thanks to Tom Barry for asking the right questions on the AVSforum.

Shameless plug:

I am looking for a job in cryptography in the Raleigh/Durham, NC area. Please contact me if you are interested.

Copyright 2001 Keith Irwin.

Date of first release: July 30, 2001.

Addendum added: August 15, 2001.

I can haz liberty? DRM, like HDCP, is unAmerican