Implementing a Key Recovery Attack on the High-Bandwidth Digital Content Protection Protocol

Rob Johnson
rob@cs.sunysb.edu

Mikhail Rubnich
rubnich@gmail.com

Andres DelaCruz
andresfdlc@gmail.com

Stony Brook University

Abstract

We describe our experiences implementing the master-key recovery attack[9] against the High-Bandwidth Digital Content Protection Protocol (HDCP)[10]. We recovered the private keys from 41 HDCP-capable monitors using the key extraction attack of Irwin[12]. We then used the master-key recovery attack of Crosby, et al.[9] to compute the HDCP global secret. The attack used commodity hardware and did not damage the monitors in any observable way. We derive several lessons for DRM producers and consumers from our experience. We propose that DRM systems should have strong renewability in order to manage risks for all parties: content-producers, device manufacturers, and consumers.

1. Introduction

Intel created the High-bandwidth Digital Content Protection (HDCP) system to prevent pirates from capturing unencrypted digital video transmitted over the DVI bus that connects computers and DVD players to video monitors[10]. Without HDCP, the DVI bus could be the weakest link in an all-digital content distribution framework. Pirates could circumvent the encryption on DVDs and Blu-ray Disks or the DRM on internet-delivered video by playing the content over the DVI bus and recording the perfect digital, unencrypted, uncompressed video stream for later playback. HDCP standards are currently managed by Digital Content Protection (DCP), LLC[2], which has adapted HDCP for other digital video transports, such as HDMI, WHDI, and IP-based protocols. DCP has licensed over 1 billion HDCP keys to manufacturers, making HDCP one of the most widely-deployed DRM schemes.[2].

Several researchers have published or claimed attacks against HDCP [9], [12], [11]. Crosby, et al., showed that an attacker that learned the private keys in 40 HDCP devices could compute a master secret for the entire HDCP system, enabling him to decrypt HDCP-encrypted content and subvert other HDCP security goals[9]. Irwin described a method for extracting the required private keys from monitors through repeated executions of the HDCP protocol[12]. Niels Ferguson has refused to publish his attacks because of the DMCA[11]. This paper describes our experiences and the lessons we learned while implementing the Irwin and Crosby attacks. As predicted, we were able to recover the HDCP master secret after extracting the private keys from 41 HDCP-capable monitors.

With this master secret, an attacker could generate the private key corresponding to any public key in the HDCP system. Thus, an attacker can eavesdrop on HDCP communications, manufacture HDCP-compliant devices without signing an HDCP licensing contract, and impersonate any existing HDCP device. Since the attacker can choose to use any key he wants, the HDCP key revocation functionality could not stop him without revoking every possible HDCP key, which would render the system useless.

Our attack proceeded in three phases. In the first phase, we used custom software to collect data from 41 different monitors that we borrowed from friends and colleagues. Data collection took about 1 day per monitor. In the second phase, we analyzed the data on a 128-node cluster to compute the private HDCP keys for each monitor we attacked. This took about about 4 days. Finally, we executed the previously published master key recovery attack on a single desktop PC. This last step took less than a second.

We draw several lessons from our experience

- DRM designers should consider their entire customer base as potentially colluding enemies, and therefore should not use protocols that are only secure against a small number of colluders.
- DRM schemes must be designed to remain secure

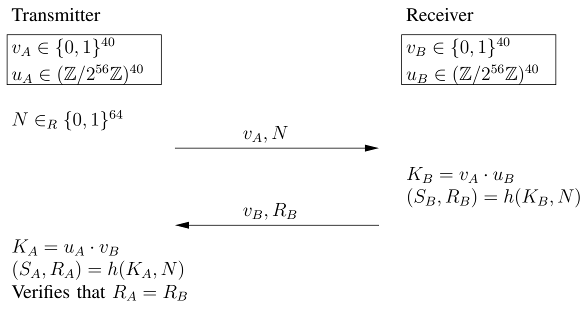

Figure 1. A simplified version of the HDCP protocol. When key agreement is successful, the 56-bit pair keys $K_A = K_B$, and devices confirm this by comparing the 16-bit values R_A and R_B. The hash function h is specified as part of the protocol. They then use $S_A = S_B$ as a session key.

even in the presence of legacy devices that do not implement the DRM system.

- DRM schemes should have renewability functionality that is well-suited to the weaknesses of the underlying DRM system.
- DRM systems should have very strong renewability, i.e. ideally it should be possible to change the master secret of the entire DRM system and update all deployed devices to use the new secret.
- DRM schemes with low security and poor renewability are particularly bad for consumers, since such schemes may be abandoned, leaving consumers with useless video cards, monitors, and other electronics.

2. HDCP

The Digital Visual Interface (DVI) created a new opportunity for movie and television piracy. Since the video signal sent of DVI is digital, a pirate could circumvent the DRM on a DVD, Blu-ray Disc, or an internet-distributed video file by playing it in a DVI-enabled device and recording the raw, unencrypted DVI video stream to a storage device. In response to this threat, Intel created the High-bandwidth Digital Content Protection system to enable compliant devices to transmit video securely. Intel created the Digital Content Protection (DCP), LLC to manage the HDCP protocol, and they have adapted it to several other video interconnects.

Each HDCP-capable device has a public key and a private key. Public keys are 40-dimensional bit vectors with exactly 20 zeros and 20 ones. Private keys are 40-dimensional vectors over $\mathbb{Z}/2^{56}\mathbb{Z}$. Keys are unique to individual devices so, for example, two identical monitors from the same manufacturer and with the same model number will contain distinct keypairs. When two devices, A and B, need to establish an HDCP-secured connection, they exchange public keys, v_A and v_B, and a nonce, N, as shown in Figure 1. They then compute "pair keys" K_A and K_B by taking the dot-product of their own private key and their peer's public key. For two HDCP-compliant devices, the public/private keypairs are created in such a way that $K_A = K_B$. The devices hash their pair keys with the nonce to produce session keys S_A and S_B and confirmation values R_A and R_B. They confirm successful key agreement by comparing the 16-bit values R_A and R_B. The devices use the session key S_A to encrypt the video data on the DVI bus.

The HDCP specification does not explain how public/private key pairs are generated, but Crosby, et al., derived the procedure from the protocol[9]. DCP possesses a secret 40×40 matrix A with entries in $\mathbb{Z}/2^{56}\mathbb{Z}$. Let v_T be a transmitter's public key and v_R be a receiver's public key. Both are vectors of length 40, but the transmitter's key is a row vector and the receiver's is a column vector. The transmitter's private key is $u_T = v_T A$ and the reciever's private key is $u_R = A v_R$. Thus $u_T \cdot v_R = v_T A v_R = v_T \cdot u_R$.

Only DCP knows the master secret A. This enables DCP to control who can manufacturer HDCP-compliant devices. Without this control, a rogue manufacturer could decide to build HDCP receivers that strip off the encryption and retransmit or record the video in unencrypted form. Only DCP can generate public/private keypairs, and manufacturers can obtain keypairs for their devices only after signing a license agreement in which they agree to take certain measures to protect the security of HDCP-transmitted content.

HDCP also contains a key revocation mechanism. DCP may issue signed revocation certificates indicating that the private key corresponding to a specific public key has been leaked and that HDCP devices should refuse to inter-operate with any device claiming to have the revoked public key. Revocation certificate lists can be distributed to HDCP-capable devices via the internet, by embedding them into DVDs, by embedding them into newer HDCP-manufactured devices, and by forwarding them from one device to another over the HDCP channel.

3. HDCP Attacks

Crosby, et al., described a method for recovering the master secret, A, given the public/private keypairs for 40 HDCP devices[9]. For example, given 40 receiver keypairs $(v_1, u_1), \ldots, (v_{40}, u_{40})$, we can construct the matrices

$$V = [v_1, \ldots, v_{40}] \quad \text{and} \quad U = \begin{bmatrix} u_1 \\ \vdots \\ u_{40} \end{bmatrix}$$

which must satisfy the equation $U = AV$. From this, we can solve for $A = UV^{-1}$. Since each HDCP public key contains exactly 20 zeros and 20 ones, V will never be invertible, so A will not have a unique solution. However, it is possible to compute a solution A' that satisfies the given equation and such that $v_T A' v_R = v_T A v_R$ for all valid transmitter and receiver public keys v_T and v_R.

The Crosby attack requires 40 device keypairs, but they did not specify how to extract the private key from a device. Irwin presented a birthday attack for recovering the private key from a monitor[12]. Eugene Leitl has speculated that this may be similar to the unpublished attack claimed by Ferguson[13], [11].

The attacker begins by picking four arbitrary distinct 64-bit nonce values N_1, N_2, N_3, N_4. For a given 56-bit pair key K, let

$$f(K) = (h(K, N_1), h(K, N_2), h(K, N_3), h(K, N_4))$$

so that we can view f as a function from 56-bit keys to 64-bit strings. The structure f depends on h, which is a complex hash function defined in the HDCP specification. We will model f as a random function.

To attack a monitor with public key v_B and private key u_B, the attacker picks ℓ random transmitter public keys, $v_1, \ldots, v_\ell$ and, for each public key v_i, executes the HDCP authentication protocol four times, once for each N_j value. From these interactions, the attacker can build a database D_B of triples

$$\{(v_i, v_B, f(v_i \cdot u_B))\}_{i=1}^{\ell}$$

The attacker sorts this table on its last column.

The attacker then begins picking random keys K and, for each key, computes $y = f(K)$ and looks for an entry of the form $(v_i, v_B, y) \in D_B$. When he finds such an entry, he adds it to a table of hits,

$$H_B = \{(v_i, v_B, K)\}_{i=1}^{n}$$

In this case, we have $f(v_i \cdot u_B) = y = f(K)$. Since f behaves like a random function from 56-bit keys to 64-bit strings, f is nearly injective. In fact, the probability that $f^{-1}(y)$ is unique is approximately $255/256$, so we may make a working assumption that $K = v_i \cdot u_B$.

The probability that a random key will have a hit in D_B is $\ell/2^{56}$, so the expected number of hits after trying m keys is $m\ell/2^{56}$. Note also that the attacker can combine the databases from several monitors without significantly slowing down the speed of lookups.

Each entry in H_B is a linear equation on u_B, so once the attacker has found 40 linearly independent equations, he can solve for u_B, similar to the master key recovery attack describe above. As with the previous attack, the attacker cannot recover u_B exactly – he can only recover an equivalent key u'_B. He can test the recovered key u'_B against other entries in D_B to confirm that he has recovered a valid key. If H_B happens to contain a false hit, i.e. an entry (v_i, v_B, K) such that $f(K) = f(v_i \cdot u_B)$ but $K \neq v_i \cdot u_B$, then he will discover it during the tests. When this happens he can find a 41st hit and try computing u_B using every possible subset of 40 hits until he finds a key that passes all the tests against D_B. There are only 40 such subsets, so this will take less than a second.

Alternatively, the attacker can use the hits from several different monitors to generate linear equations on the master secret, A. Each valid hit of the form (v_i, v_B, K) implies that $v_i A v_B = K$. After collecting 1600 linearly independent equations, the attacker can solve for A. Note that the attacker can obtain at most 40 such equations from a single monitor, but he can obtain the equations from more than 40 monitors, enabling him to spread the data collection phase over many monitors. As before, the attacker can deal with false hits by testing the recovered master secret A' against the database D_B. By collecting a few extra hits, he can repeat the computation of A' with random subsets of 1600 hits until he finds the right value. Since each hit is false with probability 2^{-8}, the probability that a subset contains no false hits is approximately $(1 - 2^{-8})^{1600} \approx 0.002$, so an attacker will have to try about 500 subsets to find the right master secret. This can be done in just a few seconds.

4. Implementing the Attack

We first needed to implement the data collection phase from the Irwin key extraction attack. For this step, we needed a method for executing the HDCP protocol over the DVI bus. HDCP protocol messages travel on the Display Data Channel (DDC), version 2, which is based on the I^{2}C bus. Many PCI video cards expose this bus to the host operating system, enabling software control of the channel. The open-source ddccontrol program[1] contains drivers for the I^{2}C buses on many popular PCI video cards, so we extended ddccontrol to implement the basic HDCP protocol shown in Figure 1.

We tested our program using several different video cards, but only one card that we tested allowed us to send and receive HDCP messages: an ATI RV370 5B60 (Radeon X300SE). Curiously, it was the oldest video card we tested, so we suspect that, as HDCP became more common, video card manufacturers began blocking software-level access to the HDCP channel. This could be for several reasons. Chipset manufacturers might disable software access to the HDCP channel in order to ensure that HDCP-enabled versions of their chipset operate correctly. When they produce a non-HDCP-capable version of the same chipset, it may still block software access to the channel. Alternatively, the manufacturers may be attempting to prevent attacks on HDCP. Our program may also be buggy, but we only used high-level $\mathrm{I}^2\mathrm{C}$ library interfaces provided by the ddccontrol software, and the software was able to access other $\mathrm{I}^2\mathrm{C}$ functions on all the cards we tested.

Since we only found one workable video card, we could perform the data collection on only one monitor at a time. This was not a significant obstacle, though, since we only attacked monitors that we borrowed from friends and colleagues. This kept the project's costs down, but it took over a year to collect enough monitors. The monitors we tested could perform about 10 HDCP authentications per second, so we could collect about 2.5 D_B entries per second, since each entry requires 4 authentications.

We collected data from each monitor for about 24 hours. During that time, we could use the monitor normally, although we did have to prevent the screensaver from putting the monitor in sleep mode. The monitors were not observably damaged during the attack. We extracted the keys from several brands and models of monitor: Dell, HP, KDS, Samsung, and Eizo, among others. We attacked three different Dell models, including nine 3007WFP monitors. Every monitor, even monitors of the same make and model, had a unique public key. Overall, we collected data from 41 monitors.

The key-search phase requires an implementation of h, the hash function from the HDCP specification. The HDCP hash function is designed to be efficient in hardware, but is difficult to implement efficiently in software because it uses bit operations extensively. We implemented a bit-sliced[8] version of the function, enabling us to execute 64 copies of the function simultaneously. Since computing f requires 4 executions of h, we could have used our bit-sliced implementation to test 16 keys per execution, but we performed an additional optimization. In our attack, D_B contained $\ell << 2^{32}$ entries, so we eliminated most candidate keys K by computing $y' = (h(K, N_1), h(K, N_2))$ and searching D_B for an entry of the form $(v_i, v_B, y'||*)$. Only a small fraction of keys would pass this test, in which case we would compute the full $f(K)$ and look for a hit in D_B. This optimization doubled the throughput, testing 32 keys during each execution of the bit-sliced implementation of h.

The data analysis was performed on a 128-node cluster of 2.8GHz Intel Xeon processors. Each node was able to test about 65000 keys per second. Our data anslysis code could process the collected data from multiple monitors at once but, since we collected data over such a long period, we ran the data analysis in several batches. It took about 4 days to extract the keys from a batch of monitors.

Once we extracted the private keys from 41 monitors, we used 40 of those keys to compute a matrix A' as described in the previous section. This computation was performed on a desktop computer and took less than a second. We then successfully verified the validity of A' by using it to generate the private key of the 41st monitor and comparing the result with the key recovered directly from the monitor.

After we submitted this paper, an anonymous Internet user posted a copy of the HDCP master secret, which Intel has confirmed is valid[6]. We have confirmed that our recovered key is equivalent to the anonymously posted master key.

5. Lessons

To understand the lessons of this attack, we should first consider how a medium-to-large group of cooperating users, as opposed to a handful of academics, might carry out the attack. We implemented the attack using borrowed monitors, a spare video card and desktop computer, and a small cluster of computers. A group of customers would have no trouble finding enough monitors, especially now that HDCP is almost a standard feature on new monitors. Although we only found one video card that supported our attack, there are almost certainly others. A cooperating group of customers could quickly compile a list of compatible cards and recruit new members with those cards or simply buy a few used ones on eBay[5].

Each customer with a compatible video card and an HDCP monitor could perform the data collection on his own monitor. Each member would transmit his collected data to the other members and receive collected data from all other members. The total amount of data required is less than 500MBs, independent of the group size, so the communication costs are reasonable even for a geographically distributed group. Based on the performance we observed on our cluster, the total time for the attack is roughly $2^{26}/g$ seconds, where g is the number of cooperating attackers. Thus, it would take 64 users less than 13 days to break HDCP, 1024 users could break it in under a day, and 131000 users could break it in 9 minutes, plus the time it takes to communicate the data. Extremely large groups could trade off communication and computation costs to minimize the total time to break the system.

These results suggest several lessons for DRM manufacturers and consumers. Manufacturers should consider their entire customer-base as potentially colluding adversaries. Projects like distributed.net[4] demonstrate that this level of cooperation can occur on the internet. DRM manufacturers should not choose protocols that are only secure against a small number of colluders. Similarly, key sizes should be chosen to resist large-scale attacks. The HDCP choice of 56-bit keys is too small, as has demonstrated by successful distributed attacks on DES.[3].

DRM schemes should have strong renewability and the renewability mechanism should complement the weaknesses of the underlying scheme. Any DRM scheme that has a master secret should have a mechanism for updating devices to work with a new master secret. Given the sophistication of HDCP's existing key revocation scheme, it seems feasible to implement a mechanism for updating devices with new keys instead of merely revoking old keys. Key revocation is a poor fit for the weaknesses of HDCP. The designers knew that 40 stolen keys were sufficient to recover the master secret[7], but their revocation scheme focuses on containing the damage from individual lost keys, not on protecting the system as a whole. If the HDCP manufacturers had chosen 72-bit keys and included a method for updating the master secret, then even a large group of attackers would need over a year to break the system, giving DCP plenty of time to deploy a new master key. By updating the master key every year, HDCP could continue to serve its purpose for several years after it is technically broken.

Weak DRM can be both a boon and a bane to consumers. Weak DRM can enable customers to restore traditional rights that the DRM scheme attempts to take away. Once a weak DRM system is broken, though, it is often abandoned, leaving consumers with obsolete devices. A good renewability mechanism can enable a DRM system to continue functioning after it is broken.

DRM schemes have a poor track record, so every new scheme has a significant risk of being broken, making devices and software that support that scheme useless. This risk is currently borne largely by consumers, since they have no recourse when manufacturers and content-producers abandon a DRM scheme. Content-producers are unlikely to share this risk by supporting DRM systems after they are broken. Device manufacturers are unlikely to issue product recalls for broken DRM systems. Strong renewability can improve reliability and fairness for consumers, manufacturers, and content providers.

6. Conclusion

We have described our implementation of the HDCP master key recovery attack. We have recovered the HDCP master secret by extracting the private keys from 41 HDCP monitors. From our experiences, we conclude that DRM schemes should have strong renewability, since they can be deployed in hardware that consumers may expect to use for over a decade. Currently, much of the risk of a security breach in a DRM scheme rests on the consumers' shoulders, but strong renewability could reduce that risk for consumers, content-producers, and device manufacturers.

References

[1] Ddccontrol. http://ddccontrol.sourceforge.net/.

[2] Digital content protection, llc. http://digital-cp.com/.

[3] distributed.net des attacks. http://distributed.net/.

[4] The distributed.net project. http://distributed.net/.

[5] Ebay. http://www.ebay.com/.

[6] Anonymous. Hdcp master key. http://pastebin.com/kqD56TmU, September.

[7] Adam Barth. Personal communication.

[8] Eli Biham. A fast new des implementation in software. In *Proceedings of Fast Software Encryption*, 1997.

[9] Scott Crosby, Ian Goldberg, Robert Johnson, Dawn Song, and David Wagner. A cryptanalysis of the high-bandwidth digital content protection system. In *Proceedings of the 2002 ACM Digital Rights Management Workshop*.

[10] LLC Digital Content Protection. High-bandwidth digital content protection system: Interface independent adaptation, October 2008.

[11] Niels Ferguson. Censorship in action: why i don't publish my hdcp results. http://macfergus.com/niels/dmca/cia.html, August 2001.

[12] Keith Irwin. Four simple cryptographic attacks on hdcp. http://www.angelfire.com/realm/keithirwin/HDCPAttacks.html, August 2001.

[13] Eugene Leitl. Dtv content protection. http://www.mail-archive.com/cryptography@metzdowd.com/msg03834.html, April 2005.