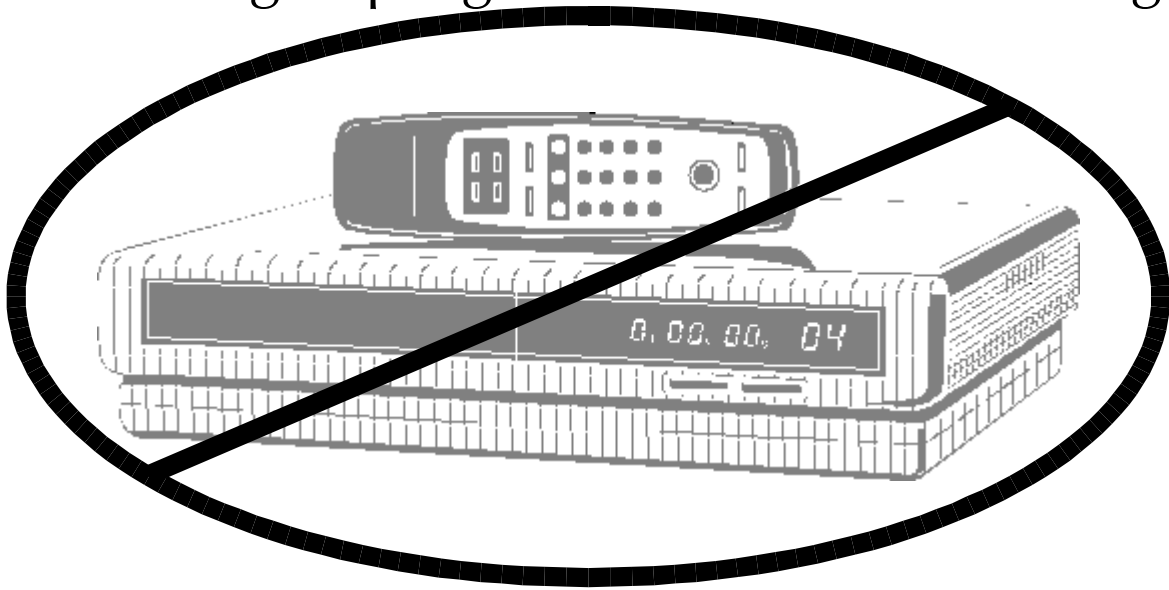


If the movie industry had got its way in 1981, time-shifting tv programs would now be illegal.



# And now they're up to the same old dirty tricks.

DVD is the newest format for home movie watching. And the movie industry is trying to limit what you can do with it.

DeCSS is a legally developed computer program designed to let people watch DVDs -- DVDs that they have legally bought or rented. But the major studios want to have complete control of your DVD-watching experience. Already, approved DVD players force you to sit through the commercials at the beginning of the disc. Future players may impose even stricter restrictions on DVD playback.

In 1981, the consumers won, and time-shifting your favorite TV program is legal today. In 2000, we can win again.

For further information, see:

2600 Magazine, [www.2600.com](http://www.2600.com)

The Electronic Frontier Foundation, [www.eff.org](http://www.eff.org),  
1550 Bryant Street, Suite 725, San Francisco CA 94103 USA

“The Right to Read”, [www.fsf.org/philosophy/right-to-read.html](http://www.fsf.org/philosophy/right-to-read.html)

```

/* css_descramble.c
 * Released under the version 2 of the GPL.
 * Copyright 1999 Derek Fawcus
 * This file contains functions to descramble CSS encrypted DVD content
 * Still in progress: Remove the use of the bit_reverse[] table by recoding the generation of
 * LFSR1. Finish combining this with the css authentication code. */
#include <stdio.h>
#include <string.h>
#include "css-descramble.h"
typedef unsigned char byte;
/* this function is only used internally when decrypting title key */
static void css_titlekey(byte *key, byte *im, byte invert)
{
    unsigned int lfsr1_lo,lfsr1_hi,lfsr0,combined;
    byte o_lfsr0, o_lfsr1, k[5];
    int i;
    lfsr1_lo = im[0] | 0x100;
    lfsr1_hi = im[1];
    lfsr0 = ((im[4] << 17) | (im[3] << 9) | (im[2] << 1)) + 8 - (im[2]&7);
    lfsr0 = (bit_reverse[lfsr0&0xff]<<24) | (bit_reverse[(lfsr0>>8)&0xff] << 16)
        | (bit_reverse[(lfsr0>>16)&0xff]<<8) | bit_reverse[(lfsr0>>24)&0xff];
    combined = 0;
    for (i = 0; i < 5; ++i) {
        o_lfsr1 = lfsr1_bits0[lfsr1_hi] ^ lfsr1_bits1[lfsr1_lo];
        lfsr1_hi = lfsr1_lo>>1;
        lfsr1_lo = ((lfsr1_lo&1)<<8) ^ o_lfsr1;
        o_lfsr1 = bit_reverse[o_lfsr1];
        o_lfsr0 = ((((((lfsr0>>8)^lfsr0)>>1)^lfsr0)>>3)^lfsr0)>>7);
        lfsr0 = (lfsr0>>8)|(o_lfsr0<<24);
        k[i] = combined & 0xff;
        combined += (o_lfsr0 ^ invert) + o_lfsr1;
        combined >>= 8;
    }
    key[4]=k[4]^csstabl[key[4]]^key[3];
    key[3]=k[3]^csstabl[key[3]]^key[2];
    key[2]=k[2]^csstabl[key[2]]^key[1];
    key[1]=k[1]^csstabl[key[1]]^key[0];
    key[0]=k[0]^csstabl[key[0]]^key[4];
    key[4]=k[4]^csstabl[key[4]]^key[3];
    key[3]=k[3]^csstabl[key[3]]^key[2];
    key[2]=k[2]^csstabl[key[2]]^key[1];
    key[1]=k[1]^csstabl[key[1]]^key[0];
}
/* this function decrypts a title key with the specified disk key
 * tkey: the unobfuscated title key (XORed with BusKey)
 * dkey: the unobfuscated disk key (XORed with BusKey)
 * 2048 bytes in length (though only 5 bytes are needed, see below)
 * pkey: array of pointers to player keys and disk key offsets
 * use the result returned in tkey with css_descramble */
int css_decrypttitlekey(byte *tkey, byte *dkey, struct playkey **pkey)
{
    byte test[5], pretkey[5];
    int i = 0;
    for (; *pkey; ++pkey, ++i) {
        memcpy(pretkey, dkey + (*pkey)->offset, 5);
        css_titlekey(pretkey, (*pkey)->key, 0);
        memcpy(test, dkey, 5); css_titlekey(test, pretkey, 0);
        if (memcmp(test, pretkey, 5) == 0) {
            fprintf(stderr, "Using Key %d\n", i+1);
            break;
        }
    }
    if (!*pkey) {
        fprintf(stderr, "Shit - Need Key %d\n", i+1);
        return 0;
    }
    css_titlekey(tkey, pretkey, 0xff);
    return 1;
}
/* this function does the actual descrambling
 * sec: encrypted sector (2048 bytes)
 * key: decrypted title key obtained from css_decrypttitlekey */
void css_descramble(byte *sec,byte *key)
{
    unsigned int lfsr1_lo,lfsr1_hi,lfsr0,combined;
    unsigned char o_lfsr0, o_lfsr1;
    unsigned char *end = sec + 0x800;
#define SALTED(i) (key[i] ^ sec[0x54 + (i)])
    lfsr1_lo = SALTED(0) | 0x100;
    lfsr1_hi = SALTED(1);
    lfsr0 = ((SALTED(4) << 17) | (SALTED(3) << 9) | (SALTED(2) << 1)) + 8 - (SALTED(2)&7);
    lfsr0 = (bit_reverse[lfsr0&0xff]<<24) | (bit_reverse[(lfsr0>>8)&0xff] << 16)
        | (bit_reverse[(lfsr0>>16)&0xff]<<8) | bit_reverse[(lfsr0>>24)&0xff];
    sec+=0x80;
    combined = 0;
    while (sec != end) {
        o_lfsr1 = lfsr1_bits0[lfsr1_hi] ^ lfsr1_bits1[lfsr1_lo];
        lfsr1_hi = lfsr1_lo>>1;
        lfsr1_lo = ((lfsr1_lo&1)<<8) ^ o_lfsr1;
        o_lfsr1 = bit_reverse[o_lfsr1];
        /*o_lfsr0 = (lfsr0>>7)^lfsr0>>10)^lfsr0>>11)^lfsr0>>19);*/
        o_lfsr0 = ((((((lfsr0>>8)^lfsr0)>>1)^lfsr0)>>3)^lfsr0)>>7);
        lfsr0 = (lfsr0>>8)|(o_lfsr0<<24);
        combined += o_lfsr0 + (byte)~o_lfsr1;
        *sec++ = csstabl[*sec] ^ (combined&0xff);
        combined >>= 8;
    }
}

```