Fault attack on the DVB Common Scrambling Algorithm

Kai Wirt

Technical University Darmstadt
Department of Computer Science
Darmstadt, Germany
`wirt@informatik.tu-darmstadt.de`

Abstract. The Common Scrambling Algorithm (CSA) is used to encrypt streams of video data in the Digital Video Broadcasting (DVB) system. The algorithm uses a combination of a stream and a block cipher, apparently for a larger security margin. However these two algorithms share a common key.
In this paper we present a fault attack on the block cipher which can be launched without regarding the stream cipher part. This attack allows us to reconstruct the common key and thus breaks the complete Algorithm.

Keywords: block cipher, cryptanalysis, fault attack, dvb, paytv

1 Introduction

The DVB Common Scrambling Algorithm is used to secure MPEG-2 transport streams. These are used for example for digitally transmitted Pay-TV in Europe. The algorithm was specified by ETSI and adopted by the DVB consortium in May 1994. However the exact origin and date of the design is unclear. Interestingly, licensees were not allowed to implement the algorithm in software and it was only available under a Non-Disclosure Agreement from an ETSI custodian. As was pointed out, this was due to "security reasons". Only very little information like an ETSI Technical Report [Eur96] and patent applications [Bew98], [WAJ98] were available to the public until 2002. In the fall of 2002 a Windows program called FreeDec which implemented the CSA in software was released and quickly reverse–engineered. The results were published on a web site [Pse03] and details on the algorithm became available to the public.

For keying the CSA, so called *control words* are used. These control words are generated from encrypted control messages contained in the DVB transport stream by a *conditional access mechanism*. Examples for these mechanisms are Irdeto, Betacrypt, Nagravision, Cryptoworks and many others. They vary between broadcasters and are usually implemented on a smart card which is required to view encrypted pay tv transmissions.

The actual key for the CSA is called *common key* and is usually changed every 10–120 seconds. The great relevance of CSA lies in the fact, that every encrypted digital Pay-TV transmission in Europe is secured using CSA. A practical break of CSA would thus affect all broadcasters which would have to exchange the hardware used to decrypt the transport streams.

The scrambling algorithm is a combination of two cryptographic primitives: a 64-bit block cipher and a stream cipher which both are keyed with the same common key. Thus a key recovery attack on one of the two primitives would break the complete algorithm.

In this paper we present a fault attack on the block cipher part which allows the recovery of the key.

The rest of this paper is organized as follows. In Section 2 we present the notation used in this paper, section 3 gives a short overview over side-channel attacks and sections 4 and 5 describe CSA resp. the block cipher part. Our Attack is presented in section 6 and final remarks are given in section 7. Tables and figures are combined in an appendix.

2 Definitions

In the rest of this paper we use the following notation:

K	the common key. A 64 bit key used for both the stream and the block cipher
k_i	denotes the i-th bit of K
K^E	denotes the running key which is derived through the key schedule of the block cipher
s_i	denotes the value held in register i
s_i^j	is the value held in register i at round j
p_i, c_i	denote a plain text resp. cipher text byte
$\overline{x}$	is the faulted value x

3 Side-Channel attacks

Conventional attacks try to find weaknesses in a cipher construction itself. There are various methods to do so, like observing the distribution of ciphertexts or attacking the structure of a cipher with algebraic methods. In contrast, side channel attacks are used to find weaknesses in an actual implementation of a cipher system. They are more powerful than conventional attacks, because of the fact, that the attacker can get additional information by observing side channels like the time required to encrypt certain plaintexts or the power usage of the encryption device.

One certain type of side channel attacks are so called fault attacks, where the attacker introduces errors in the encryption or decryption process. The attacker then gains informations on the Key by observing the difference between the actual and the faulty result. There are various results showing, that these attacks are very powerful and feasible like [BDJ97] and [ABF$^+$02].

Fault attacks are often combined with observations on other side channels, because the faults have to be introduced at specific points in the encryption/ decryption process or at a specific register value. To simplify things one specifies what values can be affected when by the attacker.

This type of side channel attacks was first applied to symmetric crypto systems by Eli Biham and Adi Shamir in [BS97]. Our attack is a variant using a slightly different setting, than in [BS97] and [BDJ97]. In the setting we investigate in this paper, the attacker is capable of changing the value of a specific register to a random value in one specific round. However, we will show, that if the attacker is only able to introduce a random error, where the exact error location is evenly distributed over the whole decryption process (i.e. the setting used by Boneh et. al) our attack still works.

One possibility to inject such errors is to do it by laser. It is possible to target at a specific part of the device performing the cryptographic operations and thus affect for example a certain register. We believe that changing the value stored in such a register to a random value is possible. Moreover we believe that using short flashes of the laser and precise equipment it is possible to do so in a certain round. We therefor believe, that the presented attack is an actual threat to the common scrambling algorithm. An overview and further references on how to realize fault attacks can be found in [BS03] where the first fault attack on the Advanced Encryption Standard is given.

4 Overview over CSA

The common scrambling algorithm can be seen as a cascade of two different cryptographic primitives, namely a block cipher and a stream cipher. Both ciphers use the same 64-bit key K, which is called the *common key*. In this section we will describe how the block and the stream cipher are combined, whereas the next section is focussing on the block cipher.

In the encryption process a m-byte packet is first divided into blocks (DB_i) of 8 bytes each. It is possible that the length of the packet is not a multiple of 8 bytes. If so, the last block is called *residue*.

The sequence of 8-byte blocks is encrypted in reverse order with the block cipher in CBC mode. The initialization vector is always equal to zero. Note, that the residue is left untouched in this encryption step.

The last output of the chain IB_0 is then used as a nonce for the stream cipher. The first $m - 8$ bytes of keystream generated by the stream cipher are XORed to the encrypted blocks $(IB_i)_{i \geq 1}$ followed by the residue to produce the scrambled blocks SB_i.

Figure 1 on page 8 depicts the descrambling process.

Note, that since we are interested in introducing errors in the decryption process of the block cipher and in comparing the actual decrypted output with the faulty output we can completly ignore the chaining mode and the stream cipher part taking only the decryption process of the last block cipher application into account. For more details on the overall design and an analysis of the stream cipher as well as an overview of properties of the block cipher we refer to [WW04].

5 The DVB CSA block cipher

CSA uses an iterated block cipher that operates bytewise on 64-bit blocks of data. In each round of the cipher the same round transformation is applied to the internal state. We will denote this transformation by ϕ. ϕ takes the 8-byte vector representing the current internal state, along with a single byte of the running key, to produce the next internal state. This round transformation is applied 56 times.

The key schedule Let ρ be the bit permutation on 64-bit strings as defined in table 2 on page 8. The 448-bit running key $K^E = (k_0^E, \dots, k_{447}^E)$ is recursively computed as follows:

$$\begin{aligned} k^E_{0,\dots,63} &= k_{0,\dots,63} \\ k^E_{64i,\dots,64i+63} &= \rho(k^E_{64(i-1),\dots,64i-1}) \oplus \texttt{0x0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i \quad \text{for all } 1 \leq i \leq 6 \end{aligned}$$

where the expression $\texttt{0x0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i\texttt{0}i$ is to be interpreted as a hexadecimal constant.

The round function The round transformation uses two non-linear permutations on the set of all byte values π and π'. These permutations are related by another permutation σ, i.e. $\pi' = \sigma \circ \pi$. The bit permutation σ maps bit 0 to 1, bit 1 to 7, bit 2 to 5, bit 3 to 4, bit 4 to 2, bit 5 to 6, bit 6 to 0 and bit 7 to 3. See table 4 on page 9 for the actual values described by π.

Let $S = (s_0, \dots, s_7)$ be the vector of bytes representing the internal state of the block cipher in an arbitrary round. The function ϕ taking the internal state S from round i to round $i+1$ is given by

$$\begin{aligned} \phi(s_0, \dots, s_7, k) = (&s_1, s_2 \oplus s_0, s_3 \oplus s_0, s_4 \oplus s_0, \\ &s_5, s_6 \oplus \pi'(k \oplus s_7), s_7, s_0 \oplus \pi(k \oplus s_7)) \end{aligned}$$

. The inverse round transformation for the decryption of a message block is then

$$\begin{aligned} \phi^{-1}(s_0, \dots, s_7, k) = (&s_7 \oplus \pi(s_6 \oplus k), s_0, \\ &s_7 \oplus s_1 \oplus \pi(s_6 \oplus k), s_7 \oplus s_2 \oplus \pi(s_6 \oplus k), \\ &s_7 \oplus s_3 \oplus \pi(s_6 \oplus k), s_4, s_5 \oplus \pi'(s_6 \oplus k), s_6) \end{aligned}$$

Encryption/Decryption A plaintext $P = (p_0, \ldots, p_7)$ is encrypted according to

$$\begin{aligned} S^0 &= P \\ S^r &= \phi(S^{r-1}, (k_{8r}, \ldots, k_{8r+7})) \qquad \text{for all } 1 \leq r \leq 56 \\ C &= S^{56} \end{aligned}$$

which yields the ciphertext $C = (c_0, \ldots, c_7)$. For decrypting this ciphertext the inverse round transformation is used and therefor the following operations have to be carried out:

$$\begin{aligned} S^0 &= C \\ S^r &= \phi^{-1}(S^{r-1}, (k_{448-8r}, \ldots, k_{455-8r})) \qquad \text{for all } 1 \leq r \leq 56 \\ P &= S^{56} \end{aligned}$$

6 Fault attack on the block cipher

Our attack consists of two steps which we will describe in this section. The first step is a fault attack on the decryption of the last block from the block cipher part of CSA which yields the last eight round keys i.e. the bits $k^E_{384} \ldots k^E_{447}$. The second step is the reconstruction of the common key $K = k^E_0 \ldots k^E_{63}$ from these keybits.

Note, that since we are only interested in the decryption of the last block from the block cipher part the stream cipher and the chaining mode used with the block cipher are irrelevant as pointed out before.

6.1 Step 1

The attacker starts by introducing a random error in the last round of the decryption process in s^{55}_6 which changes this value to $\overline{s}^{55}_6$. Since these two values appear unchanged in the decrypted plaintext the attacker can calculate

$$\begin{aligned} s^{55}_6 &= s^{56}_7 = p_7 \\ \overline{s}^{55}_6 &= \overline{s}^{56}_7 = \overline{p}_7 \\ g(k^E_{440\ldots447}) := \pi(s^{55}_6 \oplus k^E_{440\ldots447}) \oplus \pi(\overline{s}^{55}_6 \oplus k^E_{440\ldots447}) &= s^{56}_0 \oplus \overline{s}^{56}_0 = p_0 \oplus \overline{p}_0 \end{aligned}$$

from the faulted and the actual output.

Now we verify for every possible round key k' if $g(k') = g(k^E_{440\ldots447})$. Table 1 on the next page shows, how many possible round keys are expected to fulfill this equation. As we can see, we can expect that the number of possible round keys is approximately two for every introduced error. Therefor if we repeat the attack for two or three different errors, the round key can be uniquely determined.

After recovery of the round keys for the rounds $i \dots 56$ the attacker introduces an error at round $i-1$ of the decryption process and uses the known round keys to perform i rounds of the encryption process with the plaintext and the faulted plaintext. Doing so, the attacker gets the values

$$\begin{aligned} s_6^{i-1} &= s_7^i \\ \overline{s}_6^{i-1} &= \overline{s}_7^i \\ g(k^E_{8(i-1)} \dots k^E_{8(i-1)+7}) &:= \pi(s_6^{i-1} \oplus k^E_{8(i-1)} \dots k^E_{8(i-1)+7}) \\ &\quad \oplus \pi(\overline{s}_6^{i-1} \oplus k^E_{8(i-1)} \dots k^E_{8(i-1)+7}) = s_0^i \oplus \overline{s}_0^i \end{aligned}$$

He can thus retrieve the keybits $k^E_{8(i-1)} \dots k^E_{8(i-1)+7}$ as pointed out above and therefor iteratively recover the required 8 round keys.

To perform this basic version of our attack, the attacker has to introduce approximately two errors per round key, that sums up to a total of 16 errors. Additionally to uniquely determine one round key the attacker has to evaluate g(k') for all 256 different values of k' for every introduced error. Therefor the overall complexity is 16 error introductions and $8 \cdot 2 \cdot 256 = 4096$ evaluations of g.

Another possibility to recover the round keys is, that the attacker takes every possible key retrieved through the equation $g(k') = g(k^E_{8(i-1)} \cdots k^E_{8(i-1)+7})$ into account. With this method, the attacker does not have to repeat the error introduction. From table 1 we conclude, that the attack is only little more expensive. The wrong round keys can then be either discovered in the second step of the attack, or by testing all calculated common keys for the correct one.

In this version the number of required error introductions decreases to 8. However the attacker now has to evalute the g-function approximately $\sum_{i=0}^{7} 2^i \cdot 256 = 65280$ times which leaves him with 256 possible keys.

Table 1. Probability for the number of round keys for the attack

Possible number of keys	0	1	2	3	4	5	6	7	8	> 8
Probability	0.61	0.00	0.31	0.00	0.07	0.00	0.01	0.00	0.00	0.00

6.2 Step 2

The second step of the attack is the recovery of the common key from the key bits $k^E_{384} \dots k^E_{447}$. This is a straightforward task, because of the simple key schedule. The attacker can reconstruct the unknown bits according to

$$k^E_{64(i-1),\dots,64i-1} = \rho^{-1}(k^E_{64i,\dots,64i+63} \oplus 0x0i0i0i0i0i0i0i0i) \qquad \text{for all } 6 \geq i \geq 1$$

where ρ^{-1} is the inverse key bit permutation which is given in Table 3 on page 9 and the value $0x0i0i0i0i0i0i0i0i$ has to be interpreted as a hexadecimal constant.

The common key K is then given by

$$K = k^E_{0..63}$$

Since the stream and the block cipher parts of CSA share the common key this attack breaks the complete CSA - Cipher.

6.3 Improvements

The presented attack allows a time-memory tradeoff. It is possible to calculate a table which contains all the possible round keys for every combination of $s_6, \overline{s}_6$ and $g(k)$. This table uses approximately $2^8 \cdot 2^8 \cdot 2^8 \cdot 2 = 2^{25}$ bytes.

Using this improvement the attack requires only one table lookup per introduced error, resp. per possible round key in the above scenarios.

One additional possibility is, that the adversary does not calculate all 8 round keys. He can also retrieve only some of the last round keys and then perform an exhaustive search on the missing bits. Since 64 bits are enough to reconstruct the common key the costs of an exhaustive search can be reduced to $2^{64-8\cdot j}$, where j is the number of round keys calculated. Clearly, this variant requires $2 \cdot j$ introduced errors and $j \cdot 2 \cdot 256$ evaluations of the g-function in the basic setting.

6.4 Evenly distributed errors

In the case that the attacker is not able to introduce errors at a specific register at a certain round, but only an error evenly distributed over the whole decryption process, i.e. the error can affect either register at either round, the presented attack still works. This is due to the fact, that the attacker can determine if the error has affected the desired value by comparing the values s^i_7 and $\overline{s}^i_7$. If these values are not equal and $s^i_1 = \overline{s}^i_1, s^i_5 = \overline{s}^i_5$ and $s^i_2 \oplus \overline{s}^i_2 = s^i_3 \oplus \overline{s}^i_3 = s^i_4 \oplus \overline{s}^i_4 = s^i_0 \oplus \overline{s}^i_0$ the correct register and the correct round have been modified.

Assuming that the errors are evenly distributed this should occur every $56 \cdot 8$ tries. Therefor the costs for the attack in terms of the number of introduced errors only increase by a constant factor.

One further improvement would be, that the attacker records all the faulted outputs, even if the wrong round and/or register have been altered. Before introducing an error targeting the next round key, the attacker then checks if one of the recorded values is a modification of the correct register in this round. With this method the number of required faults can be decreased.

7 Conclusion

In this paper we presented a fault attack on the DVB common scrambling algorithm. Although the overall design, especially the combination of the stream and the block cipher makes simple attacks difficult [WW04] it is possible to easily break the cipher using a fault attack. This again proves, that it is important to include countermeasures against fault attacks like the verification of the result of the encryption respectively decryption process in an implementation of a cryptographic system. Additional countermeasures against the presented attack should include different keys for the stream and the block cipher part, a nonlinear key schedule and modifications on the round function of the block cipher to make the recovery of the round key more difficult.

A Appendix

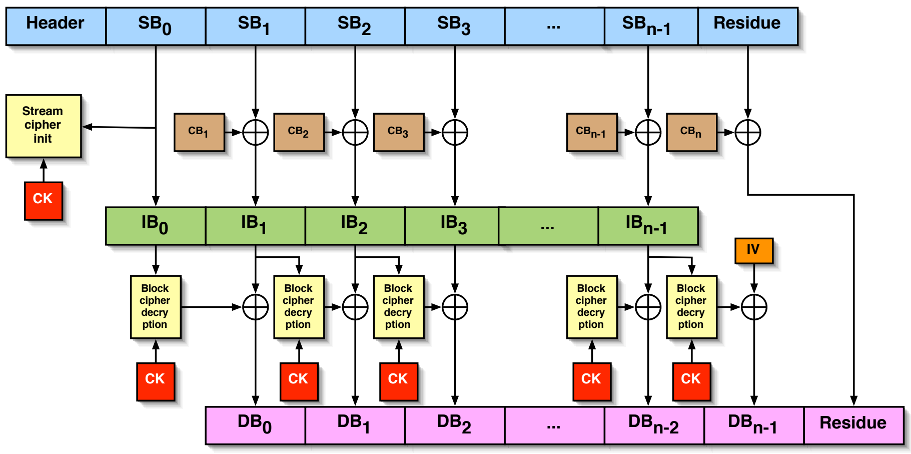

Fig. 1. Combination of block- and stream cipher

Table 2. Key bit permutation

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	17	35	8	6	41	48	28	20	27	53	61	49	18	32	58	63
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\rho(i)$	23	19	36	38	1	52	26	0	33	3	12	13	56	39	25	40
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\rho(i)$	50	34	51	11	21	47	29	57	44	30	7	24	22	46	60	16
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\rho(i)$	59	4	55	42	10	5	9	43	31	62	45	14	2	37	15	54

Table 3. Inverse key bit permutation

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	23	20	60	25	49	53	3	42	2	54	52	35	26	27	59	62
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\rho(i)$	47	0	12	17	7	36	44	16	43	30	22	8	6	38	41	56
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\rho(i)$	13	24	33	1	18	61	19	29	31	4	51	55	40	58	45	37
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\rho(i)$	5	11	32	34	21	9	63	50	28	39	14	48	46	10	57	15

Table 4. *S-Box of the block cipher. Output arranged row-wise; lower nibble on horizonal, upper on vertical*

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x00	0x3A	0xEA	0x68	0xFE	0x33	0xE9	0x88	0x1A	0x83	0xCF	0xE1	0x7F	0xBA	0xE2	0x38	0x12
0x01	0xE8	0x27	0x61	0x95	0x0C	0x36	0xE5	0x70	0xA2	0x06	0x82	0x7C	0x17	0xA3	0x26	0x49
0x02	0xBE	0x7A	0x6D	0x47	0xC1	0x51	0x8F	0xF3	0xCC	0x5B	0x67	0xBD	0xCD	0x18	0x08	0xC9
0x03	0xFF	0x69	0xEF	0x03	0x4E	0x48	0x4A	0x84	0x3F	0xB4	0x10	0x04	0xDC	0xF5	0x5C	0xC6
0x04	0x16	0xAB	0xAC	0x4C	0xF1	0x6A	0x2F	0x3C	0x3B	0xD4	0xD5	0x94	0xD0	0xC4	0x63	0x62
0x05	0x71	0xA1	0xF9	0x4F	0x2E	0xAA	0xC5	0x56	0xE3	0x39	0x93	0xCE	0x65	0x64	0xE4	0x58
0x06	0x6C	0x19	0x42	0x79	0xDD	0xEE	0x96	0xF6	0x8A	0xEC	0x1E	0x85	0x53	0x45	0xDE	0xBB
0x07	0x7E	0x0A	0x9A	0x13	0x2A	0x9D	0xC2	0x5E	0x5A	0x1F	0x32	0x35	0x9C	0xA8	0x73	0x30
0x08	0x29	0x3D	0xE7	0x92	0x87	0x1B	0x2B	0x4B	0xA5	0x57	0x97	0x40	0x15	0xE6	0xBC	0x0E
0x09	0xEB	0xC3	0x34	0x2D	0xB8	0x44	0x25	0xA4	0x1C	0xC7	0x23	0xED	0x90	0x6E	0x50	0x00
0x0A	0x99	0x9E	0x4D	0xD9	0xDA	0x8D	0x6F	0x5F	0x3E	0xD7	0x21	0x74	0x86	0xDF	0x6B	0x05
0x0B	0x8E	0x5D	0x37	0x11	0xD2	0x28	0x75	0xD6	0xA7	0x77	0x24	0xBF	0xF0	0xB0	0x02	0xB7
0x0C	0xF8	0xFC	0x81	0x09	0xB1	0x01	0x76	0x91	0x7D	0x0F	0xC8	0xA0	0xF2	0xCB	0x78	0x60
0x0D	0xD1	0xF7	0xE0	0xB5	0x98	0x22	0xB3	0x20	0x1D	0xA6	0xDB	0x7B	0x59	0x9F	0xAE	0x31
0x0E	0xFB	0xD3	0xB6	0xCA	0x43	0x72	0x07	0xF4	0xD8	0x41	0x14	0x55	0x0D	0x54	0x8B	0xB9
0x0F	0xAD	0x46	0x0B	0xAF	0x80	0x52	0x2C	0xFA	0x8C	0x89	0x66	0xFD	0xB2	0xA9	0x9B	0xC0

References

[ABF$^+$02] Christian Aumueller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In B. Kaliski, editor, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer-Verlag, 2002.

[BDJ97] Dan Boneh, Richard A. DeMillo, and Richard J.Lipton. On the importance of checking cryptographic protocols for faults. In W. Furny, editor, *Advances in Cryptology – Eurocrypt 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[Bew98] Simon Bewick. Descrambling DVB data according to ETSI common scrambling specification. UK Patent Applications GB2322994A / GB2322995A , 1998.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In B. Kaliski, editor, *Advances in Cryptology – Crypto 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 1997.

[BS03] Johannes Bloemer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (aes). In R. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag, 2003.

[Eur96] European Telecommunications Standards Institute. ETSI Technical Report 289: Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems, 1996.

[Pse03] Pseudononymous authors. CSA – known facts and speculations, 2003. `http://csa.irde.to`.

[WAJ98] Davies Donald Watts, Rix Simon Paul Ashley, and Kuehn Gideon Jacobus. System and apparatus for blockwise encryption and decryption of data. US Patent Application US5799089 , 1998.

[WW04] Ralf-Phillip Weinmann and Kai Wirt. Analysis of the dvb common scrambling algorithm. In *Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, CMS 2004. Proceedings.* Kluwer Academic Publishers, 2004.