Acta Cybernetica **16** (2003) 315-326.

Two Content Protection Schemes for Digital Items

Paula Steinby*

Abstract

Modern techniques make digital articles easy to copy and manipulate. *Content protection systems* aim at protecting the rights of producers and distributors. These mostly rely on data encryption, digital watermarking, and special-purpose devices. In this paper, we describe two content protection schemes, both of which make use of tamper-resistant devices and device-dependent decryption keys. One of the schemes uses a modified El Gamal system, in the other one we combine watermarking with encryption.

1 Introduction

Consider a scheme where a digital article is distributed over an insecure channel. During the transmission, the data may be subjected to eavesdropping and transformations. The combination of digital data and modern techniques to handle it brings along some controversial possibilities. Producing identical or manipulated copies of a digitized item is easy, and devices and programs for this purpose are commonly available.

For the parties using the transmission channel, there is a need for *privacy* and *content authentication* (i.e. capability to detect any data manipulation). Owner of an item may require *copyright protection* and further, a possibility for *traitor tracing*, maybe even for *copy* and/or *use control* of the item.

Content protection systems have been designed to protect media producers and distributors. The existing tools are limited: data encryption, digital watermarking, tamper-resistant and special-purpose devices. Encryption contributes to the privacy of the parties as well as makes the data useless for those without means to decrypt it. Watermarking enables one to recognize the copyright owner, or even distinguish between each copy of the data. A unique label in every copy provides for traitor tracing. This type of watermarking is usually referred to as *fingerprinting*. Cryptographic protection gives privacy for the transmission, but there lies a fundamental weakness in it. Namely, one must remove the encryption at some point to reproduce the item, thus leaving the item without any protection whatsoever.

*Turku Centre for Computer Science, 20014 Turun yliopisto, Finland, email: `pauste@utu.fi`

Special-purpose devices may offer a solution to this problem. These are devices with some special features, designed in view of a certain operating system. In this work, we describe two content protection schemes, which both make use of special-purpose devices, and device-dependent decryption keys. In Chapter 2 we describe a scheme with a modified El Gamal system, where the device can recognize if the input is supposed to be given in encrypted form, and refuses to process such data if given in plaintext form. In Chapter 3 we sketch a scheme to combine encryption, watermarking and compression of the data. In both schemes, we assume the device to have a secret key which is not known to anybody outside the device.

Digital watermarking is the so far best technique to protect an item after decryption. As the ultimate goal of content protection (i.e. making producing illegal copies impossible) remains unachievable, digital watermarking introduces methods to make producing, distributing and using illegal copies of some data unattractive: difficult, risky or unprofitable. We use digital watermarking for both schemes discussed in this paper, in order to bring security even in the case where the security provided by encryption and/or the device failed.

2 A scheme with modified El Gamal system

In this section, we sketch a method to protect copyrighted digital items using techniques based on public-key cryptography and a tamper-resistant device $\mathcal{D}$. The items to be protected can be visual, aural, etc. We assume that a public-key interface is used: each $\mathcal{D}$ is equipped with a public-key pair $(t_{\mathcal{D}}, s_{\mathcal{D}})$. The private key $s_{\mathcal{D}}$ is unknown even to the owner of $\mathcal{D}$. (It is a common practice that private keys are generated within smart cards such that no other unit will ever learn them.)

Our scheme has the following features:

- The data is delivered to buyers in encrypted form. The encryption is the same for all buyers. This is convenient, because then it is enough for the merchant to perform a single encryption on the data, and then make it freely available. We denote the (symmetric) encryption/decryption key by K.
- A tamper-resistant special-purpose device $\mathcal{D}$ is needed to reproduce the item. The respective device-dependent key is needed for $\mathcal{D}$ to be able to compute the decryption key K. Hence, this key is different for all buyers.
- Even if K was revealed, legal unhacked devices could not exploit it.

Consider giving up the last feature in the list. Then the key delivery could be realized by encrypting the decryption key K with the public device key $t_{\mathcal{D}}$, and sending it to $\mathcal{D}$. But if the decryption key was revealed, then unauthorized device-dependent keys could be easily computed, and thus legal devices would be compatible with hacked documents. Preventing this seems highly desirable.

In the following, we present the notation and the cryptographic primitives that will be used in our protocol.

- The tamper-resistant devices use an El Gamal type public key encryption system. The domain parameters common for all are p and g, where p is a large prime (1024 bits), and g is a generator modulo p. The private key of $\mathcal{D}$ is $s_\mathcal{D}$ $(0 < s_\mathcal{D} < p-1)$, and the corresponding public key is $t_\mathcal{D} = g^{s_\mathcal{D}} \pmod p$.
- M (the Merchant) has an item for sale. We denote the digital representation of the item by I. Prior to delivery, I is encrypted using a symmetric encryption function E with key K. We denote $enc(I) = E(I, K)$.
- B (the Buyer) wants to buy the item, and he has the device $\mathcal{D}$ with the key pair $(s_\mathcal{D}, t_\mathcal{D})$ to reproduce it from I.
- h is a cryptographic hash function with output length equal to the key length $|K|$.

For encryption, M could use some fast stream cipher. If, say, RC4 with key length 160 was used, then SHA-1 can be chosen for h with a 160 bit output.

The Protocol

The protocol proceeds as follows. Step 0, where M encrypts her data, is preliminary. Steps 1 and 2 constitute the purchase phase, and in step 3 B's device $\mathcal{D}$ decrypts and reproduces the data.

0. M selects a random $\alpha \in [1, p-1]$ and computes $K = h(g^\alpha)$. Then M computes $enc(I) = E(I, K)$, which is the data set for delivery.
1. B sends M a request for I together with his device public key $t_\mathcal{D}$.
2. M computes $r = t_\mathcal{D}^\alpha \pmod p$ and sends B the pair $\beta = (x, y)$, where $x = g^\alpha \cdot t_\mathcal{D}^r \pmod p$ and $y = g^r \pmod p$, together with $enc(I)$.
3. B inputs (x, y) and $enc(I)$ to his device $\mathcal{D}$. $\mathcal{D}$ computes $K' = x \cdot (y^s)^{-1} \pmod p$, $r' = K'^s \pmod p$. Then it checks whether $y \doteq g^{r'} \pmod p$. If this is the case, then $\mathcal{D}$ computes $K = h(K')$ and $I = D(enc(I), K)$ and reproduces I.

The protocol is a modification of El Gamal system. The difference is that instead of picking a random r we choose $r = t_\mathcal{D}^\alpha$, thus tying the value to the device $\mathcal{D}$ through its public key $t_\mathcal{D}$. If the protocol is properly performed, then $\mathcal{D}$ obtains the correct key K in step 3. This is verified by observing that

$$K' = x \cdot (y^s)^{-1} = (g^\alpha \cdot t_\mathcal{D}^r) \cdot g^{-rs} = g^\alpha \cdot g^{rs} g^{-rs} = g^\alpha \pmod p,$$

and hence $K = h(K')$. It follows that

$$r' = K'^s = g^{\alpha s} = t_\mathcal{D}^\alpha = r,$$

and hence the check in step 3 is always successful if r is of the right form.

Let us weigh a legal buyers chances to determine K after the purchase (without hacking $\mathcal{D}$). B knows the public key $t_{\mathcal{D}}$, and a pair (x, y) where $x = g^{\alpha} \cdot t_{\mathcal{D}}^{r} \pmod{p}$ and $y = g^{r} \pmod{p}$. Clearly, finding K is equivalent to finding g^{α}. Thus the task would be to compute $t_{\mathcal{D}}^{r} = g^{rs}$, given $t_{\mathcal{D}} = g^{s}$ and $y = g^{r}$. This is the famous Diffie-Hellman problem (DHP), for which no efficient algorithm is known. DHP is believed to be equivalent to the discrete logarithm problem (the equivalence has a partial proof, see [1]). The fact that r is of special form, $r = g^{\alpha s}$, does not seem to help, but one could as well select $r = \text{hash}(g^{\alpha s})$ to be on the safe side. Then also the check in step 3 changes to $y = g^{\text{hash}(r')} \pmod{p}$.

To be able to produce any device-dependent keys needed to decrypt and reproduce *enc*(I), one must be able to compute r for a given t. For this purpose, one must know either α, or the respective s, since $r = t^{\alpha} = (x(y^{s})^{-1})^{s}$. Note that the problem does not become any easier even if a hacker has discovered the encryption key $K = g^{\alpha}$; there is still the discrete logarithm problem to be solved for α. As we assumed that the knowledge of s is not available outside $\mathcal{D}$, we conclude that it is M alone who can make *enc*(I) compatible with $\mathcal{D}$.

Drawbacks and improvements

In most cases, it would be useful if it was possible to display some unencrypted items with $\mathcal{D}$ as well. However, we want to be able to distinguish between a document, which was originally delivered in plaintext form, and another document which was purchased in encrypted form and later decrypted. In particular, the decryption D(*enc*(I), K) of *enc*(I) should be distinguishable from any originally unprotected piece of data. Then, even if a hacker was able to decrypt *enc*(I) (i.e. the key K was somehow revealed), this decrypted version would not be too useful because it would be rejected by the device $\mathcal{D}$.

One solution is to embed an 'information bit' b into I before encryption, thus labeling I as a protected document. For instance, if $b = 1$, then any $\mathcal{D}$ would refuse to process the data when input in the plaintext form. The bit b can be embedded in I using some robust watermarking scheme, so that b cannot be removed or its value changed without also destroying the document (see f.ex. [2], [3], [6]). The device $\mathcal{D}$ then checks the value of the watermark in I, and decides whether to reproduce I or not on the grounds of the value of b.

We must require that any key K is authorized by some trusted third party T. Otherwise, if a hacker H can access I which is decrypted but unreproducable with any legal device because of the watermark, he can easily sidestep the hindrance caused by b. Namely, re-encrypting I will do the trick: H can choose the encryption key, take the position of M and thus compute any device-dependent key β.

To prevent this possibility, M includes T's signature for the encryption key K to the device-dependent key β in Step 2 of the protocol. In other words, β becomes a triple (x, y, z), where x and y are as before, and $z = \text{sig}_{T}(K)$. When $\mathcal{D}$ gets β

as input, it checks the validity of z before using K. Naturally, it must be assumed that hackers cannot obtain T's signatures for their own keys.

There is still another major weakness in the system. Suppose that hacker H has got hold of a key K used for some I, and that some user B has acquired the same I. Thus, B has received the respective decryption key $\beta = (x, y, z)$. Now H and B can collude: any time the hacker is able to hack some item J, then he can re-encrypt it with K. Consequently, B can use the same β to decrypt J, without having to purchase a legal copy.

The obvious solution is to bind each key K to a specific item I. We propose a few methods. The simplest one is to set $z = \mathrm{sig}_T(K \mid\mid I)$. The drawback is that then $\mathcal{D}$ has to read all of I before it can determine whether z is valid. A more practical solution would be to divide I into blocks I_0, I_1, ...,I_n, and encrypt each with a different key K_0, K_1, ...,K_n. K_0 is the original key K, and each K_i is a function of K_{i-1} and a hash of I_{i-1}. In this case we have $z = \mathrm{sig}_T(K_0 \mid\mid I_0)$, and $\mathcal{D}$ can decide z's validity right after reading I_0.

A variant would be to set $z = \mathrm{sig}_T(K \mid\mid w)$, where w is a watermark embedded in I prior to encryption. w could contain any kind of information on I, the purchase etc. The information bit b could be included in w as well. Depending on the placement of w in I, again $\mathcal{D}$ must have read at least some of I before it can verify z.

The primary aim of the proposed content protection system is to prevent hackers from getting hold of unencrypted data items, and - if failing in this - secondary to minimize the usability of illegally decrypted data. The scheme does not have traitor tracing feature. This means that should we come short of both the goals, and illegal copies of I were made and distributed, there would be no way to find who is to blame.

A way to add traceability would be to make all the legal copies look different, i.e. uniquely fingerprint them. The devices $\mathcal{D}$ could be equipped with an additional watermarking module and each I would be labeled before putting it out. Each device would have an unique watermarking pattern, and hence each copy of I would be different and distinguishable. The obvious weakness of this solution is that it relays quite heavily on the tamper-resistance of the device. One could argue, that if somebody can hack $\mathcal{D}$ to obtain K and I, he would probably be able to pass by the watermarking module as well.

3 Encryption with Watermarking

In this chapter, we will describe a purchase protocol combining encryption with watermarking. Data encryption adds to the privacy of the parties, watermarking enables copyright protection and traitor tracing. We choose to use a watermarking scheme which is compatible with the compression procedure, since it is customary to compress any data prior to sending it over a transmission channel. Next we will discuss watermarking and combining it with compression, after which the scheme with encryption and watermarking is presented. We assume that the subject of the purchase I is an image. A watermark, a bit sequence of length N, is denoted by w.

Combining watermarking with compression

In general terms, watermarking an image I means encoding the bits of a watermark w into I in some imperceptible way. A usual practice is to divide I in blocks of 8×8 pixels, and (pseudorandomly) choose the blocks in which the watermark will be embedded. Values of certain coefficients of I will be manipulated, and their absolute or relative values will then indicate the values of encoded bits of w.

Images are usually expressed by giving the gray-scale value(s) of each pixel. However, to achieve greater robustness and minimization of the computation time, watermarking is often performed in some transform domain instead of the spatial one. Namely, it is easier to predict the effects of compression (or some other manipulation) on the watermark if we work in the same domain as the manipulating algorithm.

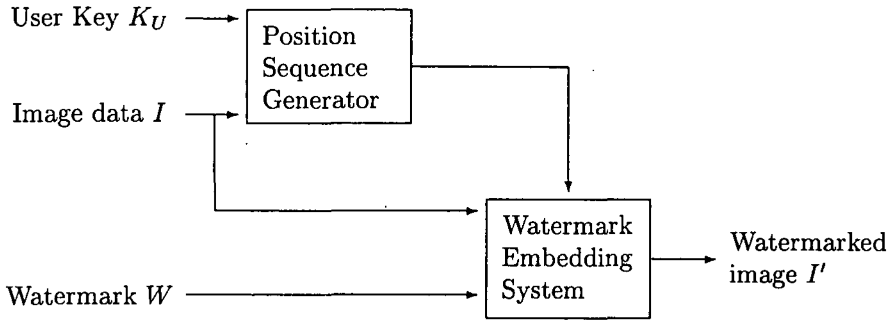
Compression algorithms make use of different transformations to separate the data into parts of different importance with respect to visual quality. *Discrete Cosine Transformation* (DCT) is an orthogonal transformation exploited by e.g. JPEG and MPEG algorithms. Its different basis vectors capture different features present in the input data. The effect is that the role of low frequency coefficients is emphasized, whereas most of the high frequency coefficients are small, and will be rounded off to zero during the compression. Therefore, placing the watermark in the low frequency DCT coefficients greatly adds to its robustness against compression. (For a throughout introduction on current watermarking schemes, see Chapter 6 in [3], or Chapter 8 in [2] on robust watermarking in general. A detailed description of the baseline JPEG can be found in [5].)

As changes in low frequency components easily become perceptual, various perceptual models, i.e. models imitating human visual system, are exploited in watermarking. One can use these models to compute so-called masking constraints, upper boundaries for the amounts a certain coefficient can be changed without causing visual effects. For more on the subject, see Chapter 7 in [2].

Many of the masking functions give the boundaries in the spatial domain instead of the frequency one. Often the problem has been solved by first embedding the watermark in the frequency domain and then cutting the visible changes in the spatial domain. The watermark remains imperceptible, but suffers in robustness. The framework by Pereira & al. in [4], however, enables strong watermarking in the frequency domain without violating the constraints in the spatial domain.

3.1 Combining watermarking with encryption

Whichever watermark embedding system or masking function is used, we can assume the outline of the procedure to be as follows:



The Position Sequence Generator is used to pseudorandomly select the pixel blocks of the image in which the watermark is placed. Hereafter, we use the word "image" as referring to a single 8×8 block, and "embedding a watermark" means encoding a single bit of the watermark in the block. This is natural, since embedding a longer watermark in a bigger image means just repeating this procedure for many enough blocks. The following notation is adopted:

I	=	the DCT-coefs of an image block
I'	=	the DCT-coefs of the image after watermarking
w	=	the watermark
E	=	encryption coefficients
D	=	decryption coefficients.

For simplicity, we assume that all the variants above are real valued vectors of length 64, although most of the entries are zero for w, E and D. By $+$ we denote component-wise addition of two vectors.

Whichever the actual watermarking embedding system, *watermarking* means making imperceptible changes in some low frequency DCT coefficients of the image: $I' = I + w$. We note that encryption is realizable in the fashion of watermarking, by making *perceptible* changes in the coefficients: $enc(I) = I + E$. Here E's non-zero entries are placed in the low frequency DCT coefficients, and they are large in magnitude compared with w.

Naturally, decryption reverses the effects of encryption in a straightforward way: $I = enc(I) + D$ where $D = -E$. However, the idea of the following protocol is to combine watermarking and encryption through 'imperfect decryption', that is by setting $D = w - E$. Then

$$\begin{aligned} enc(I) + D &= enc(I) + (w - E) \\ &= I + E - E + w \\ &= I + w \\ &= I'. \end{aligned}$$

Clearly, decryption strips off most but not all of encryption, leaving I watermarked. Further, if w is unique, then so is I'.

Let again Merchant M and Buyer B be the parties of a purchase protocol. M has image I for sale, which he encrypts prior to setting it for distribution. B has a device $\mathcal{D}$ to display the data. During the protocol, M delivers B a unique decryption vector D. As comparison between two decryption vectors D and D' gives away a lot of information on the respective watermarks w and w', any two buyers of the same item could collude and easily destroy the watermarks, unless the decryption vectors were somehow protected. We solve the problem by adding a *device mask* as follows.

Connected to every device, there is a unique device key K_{dev} which determines a mask *Dev*. *Dev* is an integer vector, which the device will automatically subtract from the DCT-coefficients of any data prior to reproducing it. Therefore M adds the respective *Dev* to each decryption vector D:

$$enc(I) + D = I + E + (w + Dev - E) = I' + Dev.$$

It is important that the explicit value of K_{dev} remains unknown to everybody except for M, because the presence of a secret *Dev* in D makes comparisons between different decryption keys useless. However we assume that B has an index number k, with which B can enable M to find out the actual K_{dev}.

The main features of the protocol are as follows:

- *Purchase:* B sends the index k to M for computing K_{dev}. M returns B a unique decryption key $K_{D,B}$. Applying $K_{D,B}$ to $enc(I)$ using the device $\mathcal{D}$, B receives a copy of I with a unique watermark w_B.

- *Tracing:* Suppose B illegally redistributes his copy of I. He can be traced on the basis of the watermark w_B, which can be extracted only from the copies originating from his version of I.

- One encryption of I can be distributed to all buyers, but each decryption key is bound to a certain buyer with a certain device. The unique watermarking is forced to be done along the decryption.

The Protocol

We will adopt the following notation:

Dev	=	the mask removed from any input data by $\mathcal{D}$ on the basis of the key K_{dev}.
w_B	=	a unique watermark embedded in B's copy of I.
K_E	=	the encryption key, on basis of which the encryption vector E is computed.
$K_{D,B}$	=	the decryption key, from which the decryption vector D_B for buyer B is achieved.

The protocol consists of a preliminary step (step 0), the purchase phase (steps 1 to 3), and step 4 where B's device decrypts and reproduces the data.

0. M encrypts image I with a secret, symmetric key K_E. Encrypted image $enc(I)$ is set for distribution.
1. B gives M the index k for computing the key K_{dev}.
2. M computes K_{dev} on the basis of given k, chooses a watermark w_B for B, and computes a unique decryption key $K_{D,B}$ s.t. $D = Dev + w - E$.
3. M returns $K_{D,B}$ to B.
4. B applies $K_{D,B}$ together with the device key K_{dev} to $enc(K)$.

In the last step,

$$\begin{aligned} enc(I) \rightarrow enc(I) + D &= I + E - E + Dev + W \\ &= I' + Dev, \end{aligned}$$

and further

$$I' + Dev \rightarrow I' + Dev - Dev = I'.$$

Hence the result of $\mathcal{D}$'s computations is the watermarked image I'.

We have not specified the correspondences $K_E \sim E$, $K_D \sim D$, or $K_{dev} \sim Dev$. Use of the keys is necessary, since the actual vectors D and Dev are too long and too many to be transmitted as such (even though they mostly consist of zeros). A mapping to pack the information is needed. Possible solutions are many, as an example we give one.

We have thought of D and Dev as 64-dimensional integer vectors. Let us present a vector as a concatenation of the binary presentations of its entries, and let N be an integer such that for every entry i in D or Dev, $|i| \leq |N|$. The length of the binary presentation is then $64 \cdot \log_2(2N)$. For $N = 2^{15}$ this equals 1024. We can establish a one-to-one correspondence between 1024-long binary vectors and the elements of the group $\mathbb{Z}_p^*$, where $|p| = 1024$. Therefore, each vector D or Dev can be presented as an element of $\mathbb{Z}_p^*$.

Note that most of the entries of the vectors are zeros, as it is enough to mask/encrypt about five to twenty most significant of them. Thus, we can cut the extra zeros by setting for example $D, Dev \in [-N, N]^{20}$. Then, for $N = 2^{15}$, $20 \cdot \log_2(2N) = 320$ and thus the vectors can be expressed as elements of $\mathbb{Z}_p^*$, where $p = |320|$ only.

In the previous scheme we assumed there is a mapping $k \rightarrow K_{dev}$, which remained unknown to B but could be found out by M. In this case, it could be f.ex. a permutation on $\mathbb{Z}_p^*$. In the following chapter, we will re-examine the concepts of and relations between k, K_{dev} and Dev.

3.2 On the device key K_{dev}

Consider the following scheme: buyers B and B' with devices $\mathcal{D}$ and $\mathcal{D}'$ resp., both purchase an encrypted item $enc(I) = I + E$ from the merchant M. The following communication takes place:

$$\begin{array}{lrcl} M \to B: & K_{D,B} \sim D_B & = & -E + Dev_{\mathcal{D}} + w_B \\ M \to B': & K_{D,B'} \sim D_{B'} & = & -E + Dev_{\mathcal{D}'} + w_{B'} \end{array}$$

Here is a chance for B and B' to collude. Subtracting one decryption vector from the other, they learn $S = Dev_{\mathcal{D}} - Dev_{\mathcal{D}'} + w_B - w_{B'}$. Here $w_B - w_{B'}$ is small, thus $S \approx Dev_{\mathcal{D}} - Dev_{\mathcal{D}'}$. Now, if B buys another image $enc(J) = J + E_J$, then B' can use the corresponding decryption vector $D_B^J = -E_J + Dev_{\mathcal{D}} + w'_B$ by computing

$$\tilde{D}_{B'}^J = D_B^J - S = -E_J + Dev_{\mathcal{D}} + \epsilon,$$

where $\epsilon = w'_B + w_{B'} - w_B$ is a small error. Decryption of $enc(J)$ with the new vector $\tilde{D}_{B'}^J$ using the device $\mathcal{D}'$ yields $J' \to J + Dev_{\mathcal{D}} + \epsilon \to J + \epsilon$. Thus, B' can use his own device to reproduce B's copy of J with only small distraction.

The above scheme suggests that the device mask Dev should not be fixed, but different for each I. In our model, Dev is deterministically computed from a device key K_{dev} (see the end of the previous chapter). Thus, what we need is a method to generate keys K_{dev}. As the computation $K_{dev} \to Dev$ is reversible, K_{dev} must remain unknown to B.

Relying on the tamper-resistance of $\mathcal{D}$, the keys could be generated within the device by some function $f^n(k) = K_{dev}^n$, for $n = 1, 2, \ldots$ etc. The merchant would be able to compute Dev if he was given the pair (k, n) instead of the index k only. However, the system with indices and secret generating functions seems somewhat impractical, because duplex communication between B and $\mathcal{D}$ is needed, as well as an active third party with the knowledge of $k \sim K_{dev}$ correspondences and the functions f.

To avoid these difficulties we take a new starting point: allowing M to take part in the generation of K_{dev}. If M is able to compute K_{dev} on his own, then the role of index k shrinks into tying K_{dev} to $\mathcal{D}$. If M can actually *decide* the value of the key (and thereby of the mask) used in decryption, then M can as well give the value of the decryption key and the mask together. In other words, M can provide B with a key K, which corresponds with the vector $D_B - Dev$, instead of giving $K_{D,B}$ and K_{dev} separately.

Let us discuss options of carrying out the above scenario. Let M provide $\mathcal{D}$ with a seed d to generate K_{dev}, as a function of both I and k, for example. M generates d from I, and computes $f(k, d) = K_{dev} \sim Dev$. If M sends d to B together with the decryption key (step 3), then $\mathcal{D}$ can compute K_{dev} too. The problem is that so can B, unless the function f is kept secret from B (but it has to be available to $\mathcal{D}$, which in turn again would complicate the system).

Function f is not needed, if M decides the value of K_{dev} and sends it to B as such. However, if K_{dev} has the value of an element in $\mathbb{Z}_p^*$, then the correspondence

$\mathbb{Z}_p^* \sim \{0,1\}^{|p|}$ between the key and the mask must remain unknown to B (but accessible to $\mathcal{D}$). If $\mathcal{D}$ possessed a public key pair $(s_\mathcal{D}, t_\mathcal{D})$, then M could protect K_{dev} from B by encrypting it with $t_\mathcal{D}$ before handing it out. $\mathcal{D}$ would still be able to find out K_{dev}, since it has access to $s_\mathcal{D}$.

The scheme with $\mathcal{D}$ possessing a public key pair (s_D, t_D) where s_D is accessible to $\mathcal{D}$ only (c.f. the scheme in Chapter 2) seems useful. We can use the Diffie-Hellman protocol to generate the key K_{dev} as follows. M creates an ephemeral key pair (s_M, t_M), and performs the D-H protocol to obtain $K_{dev} = t_D^{s_M}$. Then he can further compute Dev, and compute the decryption vector $D_B = -E + Dev + w_B$. Now M sends B both $K_{D,B}$ and t_M. Given these, $\mathcal{D}$ can decrypt $enc(I)$, since $enc(I) + D_B = (E + I) + (-E + Dev + w_B) = I + Dev + w_B = I' + Dev$, where $Dev = t_M^{s_\mathcal{D}}$. On the other hand, on basis of the given information, B cannot learn and remove Dev, as he does not know s_D which is needed to compute K_{dev}.

Assume K_{dev} is generated as above. M wants to give B a key K such that $K \sim D_B - Dev = -E + w_B$ (we assume that he can easily compute the target value K once he knows $D_B - Dev$). To give $D_B - Dev$ with a single key, we can proceed as follows.

1. M computes the value of $K \sim Dev + D_B$.
2. M generates a random public-key pair (s_M, t_M), and computes $K_{dev} = t_D^{s_M}$.
3. M computes the difference $\Delta = K - K_{dev}$ and sends t_M and Δ to B.
4. $\mathcal{D}$ computes $K_{dev} = t_M^{s_\mathcal{D}}$, and further $K = \Delta - K_{dev}$.

As B knows only that Δ is the difference between K and K_{dev}, he cannot find out either of these values because he does not know $s_\mathcal{D}$. The key K is computed in each end of the transmission channel, but not transmitted at all.

The above method can be applied even if K_{dev} was generated in some other manner, as long as M can find out the target value $K \sim D_B - Dev$ on his own. Then the difference between a random key $t_D^{s_M}$ and the target is computed, and t_D and the corrective key (Δ above) are given to B.

References

[1] B. den Boer, Diffie-Hellman is as Strong as Discrete Log for Certain Primes, *Proceedings of CRYPTO'88*, LNCS 403, Springer-Verlag, 1988, pp. 530-539.

[2] I. Cox, M. Miller and J. Bloom, *Digital Watermarking*, Academic Press, San Francisco 2002.

[3] S. Katzenbeisser and F. Petitcolas (ed.), *Information Hiding Techniques for Steganography and Digital Watermarking*, Artech House, London 2000.

[4] S. Pereira and T. Pun, Optimal Transform Domain Watermark Embedding Via Linear Programming. *Signal Processing* 81, No. 6 2001, pp. 1251-1260.

[5] K. Wallace, The JPEG still picture compression standard. *Communications of the ACM* 34, No. 4 1991, pp. 30-40.

[6] J. Zhao and E. Koch, Embedding Robust Labels Into Images For Copyright Protection. *Proceedings of the International Congress on Intellectual Property Rights for Specialized Information, Knowledge and New Technologies*, Vienna 1995.