On the Difficulty of Software Key Escrow

Lars R. Knudsen[1] and Torben P. Pedersen[2]

[1] Katholieke Universiteit Leuven, Belgium, email: knudsen@esat.kuleuven.ac.be
[2] Cryptomathic, Denmark, email: tpp@cryptomathic.aau.dk

Abstract. At Eurocrypt'95, Desmedt suggested a scheme which allows individuals to encrypt in such a way that the receiver can be traced by an authority having additional information. This paper shows that the proposed scheme does not have the required properties, by devising three non-specified protocols misleading the authority. We also discuss how to repair Desmedt's scheme, such that our attacks are no longer possible. However, by allowing slightly more general, but absolutely realistic attacks also this improved system can be broken. In fact, we argue that software key escrow as proposed by Desmedt will be very hard to implement as it requires that the distributed public key can only be used in few, well-defined systems. Furthermore, even if this is achieved, most applications to key distribution can be broken.

1 Introduction

In key escrow systems, such as Clipper [5], it is necessary to be able to identify ciphertexts sent to a person whose messages are to be read by the authorities (given a court order, of course). The necessity of such identification is discussed in [4]. In Clipper the identification is enforced by adding a field, LEAF, to the ciphertext. If this field is missing the decryption device will refuse to decrypt. Thus this technique depends on the fact that this device is in a tamper resistant unit, such that decryption cannot be enforced.

At Eurocrypt'95, Desmedt suggested a key escrow scheme not depending on tamper resistant devices which allows individuals to encrypt information in such a way that the recipient (i.e. the person able to recover the clear text) can be traced by an authority having additional information [1]. According to [1] the investigation of such software key escrow systems has also been initiated by NIST.

Key escrow systems can only be argued secure in situations where the participants have not had the possibility to distribute other (secret) keys among themselves. This is a necessary assumption, because otherwise they could have used these keys instead of those distributed by the authority.

However, without physical protection such as that provided by tamper resistant smart cards no practical key escrow system can avoid that some users use a publicly described protocol different from that devised by the authority. We will say that such a *non-specified protocol* succeeds if, first, the users obtain the same level of security as in the specified protocol, and second, the receiver can

U. Maurer (Ed.): Advances in Cryptology - EUROCRYPT '96, LNCS 1070, pp. 237-244, 1996.

decrypt the ciphertext correctly, but the authority cannot identify the receiver (either because the identification fails, or because a wrong user is identified).

In this paper we consider the system proposed in [1] and devise three non-specified protocols, by which users can communicate secretly and mislead the authority at the same time. By the first protocol it is possible to send a message to two or more collaborating receivers, either of who can then decipher the message. If the authority tries to identify the receiver an "innocent" individual (different from the collaborating receivers) may be identified. The success of this protocol depends on the ability of two collaborating receivers to communicate privately during key generation. Our second protocol does not require this, and enables any registered users to communicate secretly and at the same time mislead the authority. Our third and simpler protocol is applicable in the case where the escrow system is used for key distribution.

This paper is organised as follows. First, Section 2 discusses possible attacks on software key escrow. This is not a complete definition of such systems, but is meant to provide a general basis for our work. In Section 3 Desmedt's proposed software key escrow system is described briefly, and in Section 4 we give three attacks on Desmedt's scheme and suggest a redesign, which prevents the second and third attacks. However, the redesigned system can be broken by general attacks which additionally shows that a secure software key escrow system will in general be very hard to construct. This is discussed in Section 5.

2 Software Key Escrow

Software key escrow is only defined very informally in [1] by mentioning some of the properties, that such systems must have. It is out of the scope of this paper to give a complete definition (see [4] for a discussion of the properties of escrow systems), but before presenting our attacks the major components and properties of software key escrow are described.

A key escrow system involves a number of users, say $P_1, P_2, \ldots, P_n$, and an authority, A, which should be able to trace the recipient of encrypted messages (and subsequently decrypt the message, if applicable). The system consists of a protocol (or algorithm) for key generation, and algorithms for encrypting, decrypting and tracing as described below. All components must be efficient (i.e., run in polynomial time in a security parameter).

- Key generation: This is a protocol which results in each P_i getting a pair of public and secret keys (denoted (p_i, s_i)) and A obtaining some auxiliary information, aux.
- An encryption algorithm, E, which on input a message from a suitable defined message space, $\mathcal{M}$, and a public key produces a ciphertext.
- A decryption algorithm, D, which given a ciphertext and a secret key produces the corresponding clear text.
- An algorithm, ID, which on input a ciphertext, some public information and aux returns $i \in \{1, 2, \ldots, n\}$. Intuitively, i is the index of the person able to decrypt the ciphertext.

For this system to work properly, it must be required that if the keys are generated as prescribed, then

$$\forall i \in \{1,\ldots,n\}, M \in \mathcal{M}:$$
$$D(E(M,p_i),s_i) = M \ \wedge \ ID(E(M,p_i),(p_1,\ldots,p_n),aux) = i.$$

We next discuss some security aspects of such systems. In an attack, a sender S is trying to send an encrypted message which can be decrypted by a collusion of cooperating receivers, $R_1,\ldots,R_k$ (note that Desmedt also allows receivers to conspire [1, Footnote 10]).

We make the restriction that S may not have sent or received any message over any private channels prior to the attack. This is quite restrictive, but as mentioned in the introduction, key escrow is only possible if S has not had any private communication with $R_1,\ldots,R_k$. However, restricting the possible behaviour of the attacker does not make the attack weaker.

Let pub_inf_S denote all information which S has received prior to the attack and let pub_inf_i denote all the public information, which R_i has received for $i = 1,2,\ldots,k$. Finally, let pub_inf denote all information, which has been sent prior to the attack (by any participant) including the public keys. We assume that A has this maximal amount of information.

A generic attack runs as follows. Given a message, $M \in \mathcal{M}$, pub_inf_S and the public keys of $R_1,\ldots,R_k$, S computes a number of ciphertexts $c_1,\ldots,c_l$. Based on these, $(pub\_inf_i)_{i=1,2,\ldots,k}$ and their secret keys, $R_1,\ldots,R_k$ compute a message, $M' \in \mathcal{M}$. The attack is successful if

1. $M' = M$;
2. It is not easier to find M given $c_1,\ldots,c_l$ and $(pub\_inf_j)_{j=1,\ldots,k}$ than if M had just been encrypted as $E(M,p_j)$ (i.e., as in the specified protocol) for some j for which P_j is among $R_1,\ldots,R_k$.
3. A is not able to identify any of the receivers. In other words, for all $i = 1,2,\ldots,l$ ID on input c_i, aux and the public string $(pub\_inf, c_1,\ldots,c_l)$ either fails or outputs a number not identifying any of the receivers. In the first case A will discover the fraud, and in the second case A will be totally mislead.

A generic attack as described above can be executed in several ways. Some possibilities are:

- One receiver. This means that $k = 1$.
- Many receivers: A distinction can be made whether the receivers cooperate using a secret channel or only public discussions. Since, S is not allowed to use a private channel it could be argued that the same should hold for the receivers. However, in our opinion a strong key escrow system should also be able to cope with receivers using private communication internally, since we are looking at the transfer of a message from S to the group of receivers.
- Usage of public keys. We distinguish whether the public key is only used as input to the prescribed encryption algorithm (which may be possible) or it is used in other systems as well. In the latter case the attack can be prevented

using physically protected devices, whereas this may not help in the former. Note, however, that if a different encryption method is used A knows it as part of *pub_inf*.

As mentioned initially other attacks are conceivable, but in this paper we only consider attacks, which can be described in these terms.

3 The Proposed Solution

The scheme proposed in [1] is based on the ElGamal public key scheme (see [3]). First determine m such that at most $n \leq 2^m$ individuals can participate. Let $p, q_1, \ldots, q_m$ be large, different primes such that each q_i divides $p-1$, and let Q denote the product $\prod_{i=1}^{m} q_i$.[3] Furthermore, let g be an element in $\mathbb{Z}_p^*$ of order Q.

The authority selects these numbers together with a personal public number g_j for the j'th individual. If $\mathbf{e}_j = (e_1, \ldots, e_m) \in \{0,1\}^m \setminus \{0\}^m$ uniquely identifies the j'th participant, then

$$g_j = g^{\frac{Q}{\prod_{e_i=1} q_i}}.$$

Thus the order of g_j is $\prod_{e_i=1} q_i$, and this is different for different j's.

The j'th participant will have a secret key s_j and a public key (g_j, y_j) such that $y_j = g_j^{s_j}$. An encryption of $M \in \mathbb{Z}_p^*$ under this public key is a pair

$$(c_1, c_2) = (g_j^r, M \times y_j^r),$$

where $r \in \mathbb{Z}_{p-1}^*$ is chosen uniformly at random. A pair (c_1, c_2) can be decrypted as

$$M = c_2 / c_1^{s_j}.$$

The authority can trace the owner of the corresponding public key since c_1 has order $\prod_{e_i=1} q_i$, unless $\gcd(r, p-1) > 1$ in which case c_1 might be in a smaller subgroup. However, without knowing the factorisation of $p-1$ it seems hard to utilise this property in attacks against the system.

The scheme is used only to exchange a common session key, therefore the sender should choose a uniformly random $M \in \mathbb{Z}_p^*$. Once M has been obtained, both sender and receiver hash M to obtain a session key [1].

In [1] it has not been suggested how to generate the user identifier vectors $\mathbf{e}_j$. Desmedt does note, however, that first, there is no need for the authority to reveal the vectors or how they are computed, and second, it might be better to let the Hamming weight of all vectors $\mathbf{e}_j$ be identical.

[3] [1] suggests that each q_i is 320 bits long. Thus, as also noted by Desmedt, the scheme will be quite slow in practice for a moderate number of users.

4 Problems with the Solution

In the following we will first give a method, by which it is possible to send a message to two (or more) collaborating receivers, who can then decipher the message. If the authority tries to identify the receiver, a user different from the collaborating receivers may be identified or the identification fails (depending on the setup of the identity vectors). This attack requires that the two receivers select the same secret key. The second attack involves only one receiver, but requires two ciphertexts. The third attack requires only one receiver and one ciphertext, and works in the case where the escrow system is used for key distribution only.

4.1 Attack Involving Conspiring Receivers

Consider a scenario in which three participants cooperate, and denote these by S, P_i and P_j corresponding to one sender and two receivers. Let the public keys of P_i and P_j be (g_i, y_i) and (g_j, y_j), respectively.

Below it will be shown that S can send an encryption of a message in such a way that both P_i and P_j can decipher it, but if the authority tries to identify the recipient it will not obtain the identity of any of these three parties.

Protocol 1 *Using a private channel during key setup, P_i and P_j select $s_i = s_j = s$. P_i and P_j publish a message saying that they have chosen the same secret key. S encrypts $M \in \mathbb{Z}_p^*$ as*

$$(c_1, c_2) = ((g_i g_j)^r, M \times (y_i y_j)^r),$$

where $r \in \mathbb{Z}_{p-1}^*$ is chosen at random. This corresponds to encryption under the public key $(g_i g_j, y_i y_j)$. P_i (and P_j) can decipher the message as

$$M' = c_2 / c_1^{s_i}.$$

Here c_1 is in a subgroup identifying neither P_i nor P_j. Also, the protocol is easily extended to the cases where more than two receivers choose the same secret keys.

Whether c_1 identifies a registered user depends on how the user identifier vectors $\mathbf{e}_j$ are constructed. However, A will not be able to identify P_i or P_j. By choosing the identity vectors properly, it can, however, be ensured that c_1 will not encode a registered user.

Note, that the authority has no way to decide (let alone prove) whether two receivers select the same secret key (a publicly broadcast message does not serve as a proof).

4.2 Attacks Involving one Receiver

Consider a scenario in which a sender S and a receiver P_i try to attack the system using the following published protocol. Let (g_i, y_i) denote the public key of P_i and let (g_k, y_k) be the public key of some registered user $P_k \neq P_i$.

Protocol 2 *Initially P_i chooses a random element, $h \in \mathbb{Z}_p^*$ and publishes it. S encrypts $M \in \mathbb{Z}_p^*$ as follows. First, S sends to P_i*

$$C = (c_1, c_2) = (g_k^{r_1}, M \times y_i^r)$$

where $r, r_1 \in \mathbb{Z}_{p-1}^*$ are chosen at random. Next, S sends to P_i

$$C^* = (c_3, c_4) = (g_k^{r_2}, h g_i^r)$$

where r is as in c_2 and $r_2 \in \mathbb{Z}_{p-1}^*$ is chosen at random. Clearly, P_i (only) can decipher the message as

$$M' = c_2/(c_4/h)^{s_i}.$$

Here both c_1 and c_3 are in a subgroup identifying the registered user P_k. The purpose of h is to prevent that c_4 is in a small subgroup, which would identify P_i.

Note, that the authority even with the knowledge of the non-specified protocol has no way of deciding whether C and C^* contain two messages to P_k or one message to P_i.

For the application to key distribution, which is the typical situation [1], there is a simpler protocol.

Protocol 3 *Initially P_i chooses a random element, $h \in \mathbb{Z}_p^*$ and publishes it. S computes the session key, $M = g_k^{r_1}/y_i^r$, and sends to P_i*

$$C = (c_1', c_2') = (c_2, c_1) = (M \times y_i^r, h g_i^r)$$

where $r, r_1 \in \mathbb{Z}_{p-1}^*$ are chosen at random. P_i deciphers the message as

$$M' = c_2/(c_1/h)^{s_i}.$$

Again the purpose of h is to hide the attack. The authority has no way of deciding whether C contains a message to P_k using the specified protocol or contains a message to P_i using the non-specified Protocol 3, since the first component of the cipher text, $M y_i^r = g_k^{r_1}$, identifies P_k as the recipient.

Note that the subgroup generated by g_i is by far large enough to encode all possible session keys (Desmedt suggests that g_i has order approximately 2^{320}, which should be compared with a, say, 128 bits session key).

The two attacks above exploit that the first and second part of the ciphertexts in the ElGamal system can be separated without destroying the ability for the receiver to decrypt, and the attacks can be prevented if both parts of the ciphertext are in the same subgroup. We do not know how to achieve this in general, but for the application to key distribution, it can be done by forcing M to be in the subgroup $< g_i >$ if the receiver is P_i. This can for example be done by choosing random $r_2 \in \mathbb{Z}_{p-1}^*$ and setting $M = g_i^{r_2}$. In that way both parts of the ciphertext belong to the same receiver dependent subgroup.

However, as discussed below, there are practical problems with this solution as well.

5 General Problems

In the attacks described above the public key was used only in the intended crypto systems. However, in the setup used in [1] it is possible to use the public keys in different crypto systems. One simple example is to use the Diffie-Hellman key exchange protocol [2]. Assume that a list of generators g_k has been broadcasted. Then any two users in the escrow system can use the generators and their own secret keys to exchange a new session key using the Diffie-Hellman protocol with a different generator each time.[4]

Another possibility is to replace c_1 by $g_i^r g_j$, where P_i publishes the index j. Here, the authority might discover the fraud (depending on the choice of user identity vectors).

Many such variations are possible, but the point we want to make is that ID must be able to cope with all of these and we expect it will be hard to come up with a method for doing that. The range of possible variations clearly depends on the public-key pairs and not on the specified encryption method. This is one problem with software key escrow.

Next, even if it was possible to construct a software key escrow system handling all variations of the prescribed encryption system, A may not be able to exploit this. If the system is used for key distribution, the authorities may not in practice be able to find the actual session key being used. Consider the case where two users, P_i and P_j, are going to use session keys $K_1, K_2, \ldots, K_m$ over a period of time. Using the escrowed public key system, they exchange a key k_l (which of course the authorities may be able to find). Then they compute the l'th session key as

$$K_1 = \mathcal{H}(k_1)$$
$$K_l = \mathcal{H}(K_{l-1}, k_l) \qquad l = 2, 3, \ldots$$

where e.g. $\mathcal{H}$ is a public one-way hash function. Of course the method for doing this must be published (i.e., the authorities know it), but the authority will only be able to find K_l if it knows all previous keys. In other words, the authorities must tape all key exchanges!

6 Conclusion

In this paper we have shown that the scheme proposed by Desmedt does not have the required properties. We devised three non-specified protocols misleading the authority. We showed how to repair Desmedt's scheme, such that our attacks are no longer possible, but by allowing slightly more general attacks also this improved system was broken. We are convinced that the software key escrow as proposed by Desmedt will be very hard to implement as it requires that the

[4] Note, that a user, P_i, can raise any generator g_k to his secret key by choosing $c_2 \in \mathbb{Z}_p^*$ at random and asking his device to decrypt the cipher text (g_k, c_2). If this returns the message, M, then the required result can be obtained as $c_2/M \bmod p$.

distributed public keys can only be used in few, well-defined systems. In general, we showed how most key escrow applications to key distribution can be broken.

References

1. Y. Desmedt. Securing traceability of ciphertexts – towards a secure software key escrow system. In L.C. Guillou and J.-J. Quisquater, editors, *Advances in Cryptology - EUROCRYPT'95, LNCS 921*, pages 147–157. Springer Verlag, 1995.
2. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, 1976.
3. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. on Information Theory*, IT-31:469–472, 1985.
4. Y. Frankel and M. Yung. Escrow Encryption Systems Visited: Attacks, Analysis and Design. In *Advances in Cryptology - proceedings of CRYPTO 95*, volume 963 of *Lecture Notes in Computer Science*, pages 222–235. Springer-Verlag, 1995.
5. A proposed federal information processing standard for an escrowed encryption standard (EES). Federal register, July 30, 1993.