A fair certification protocol

Chris J. Mitchell and Konstantinos Rantos
Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
Tel: +44 1784 443423
C.Mitchell@rhbnc.ac.uk, K.Rantos@dcs.rhbnc.ac.uk

ABSTRACT

In this paper a 'fair' key generation and certification protocol for Diffie-Hellman keys is proposed, which is intended for use in cases where neither User nor CA are trusted to choose the User's key on their own. This protocol also ensures that key agreement mechanism 1 in ISO/IEC 11770-3 [2] provides 'fair key agreement' [4].

Keywords

Key agreement, commitment, certification.

1. INTRODUCTION

In the multi-part international standard ISO/IEC 11770, a number of key establishment techniques are described. Key establishment, as defined in ISO/IEC 11770-3 [2], is "*the process of making available a shared secret key to one or more entities*", and it can be subdivided into key transport and key agreement. Key agreement is "*the process of establishing a shared secret key between entities in such a way that neither of them can predetermine the value of that key*" [2]. This definition implies that the shared secret is derived as a function of the information contributed by or associated with all the communicating parties such that none of them can predetermine the value of the key [3, §12.2].

Key agreement mechanisms are used in environments where the communicating parties, who may not trust one another, wish to be sure that a session key used to protect communications between them is derived so that neither of the communicating parties can predetermine part or all of its value. As briefly discussed in [4], the mechanisms described in ISO/IEC 11770-3 [2], clause 6, and 11770-2 [1], clauses 5.5 and 5.6, do not provide 'fair key agreement', as they do not prevent one of the communicating entities from choosing part of the shared secret key.

The basic idea behind most key agreement protocols, and certainly all the protocols described in ISO/IEC 11770, is that both parties provide a 'key component', and the two components are combined in some way to give the key. The method used to combine the components is typically a one-way function. As mentioned in ISO/IEC 11770-3 [2], certain checks, depending on the particular key agreement mechanism and/or cryptographic functions used, should also be enforced to prevent the use of weak values (key components).

However, in the mechanisms in the above standard, neither the use of a one-way 'combiner' function, nor these checks can prevent one entity gaining an advantage over the other. Suppose, as is the case with most such schemes, one entity (*A* say) sends its component to the other entity (*B* say) before *B* sends its component back to *A*. There is then nothing to stop *B* from working on the key component received from *A* prior to choosing its own component, allowing *B* to choose part of the shared secret key. Specifically, 'if *B* is prepared to perform approximately 2^s computations of the one-way function used to combine key components prior to sending a response to *A*, then *B* will be able to choose *s* bits of the shared secret key' [4]. The computation of the combinations has to be performed within the limited space of time that *B* has prior to sending a response back to *A*. Yet, if a fast hash function, such as SHA-1, is used to combine components, *B* may be able to test as many as $10^4 - 10^5$ key components in a second or so, allowing him to choose as many as 16 bits of the shared secret key.

Before proceeding we consider why allowing one of the two entities to choose a few bits of the shared key might be a threat. Suppose entity *B* has agreed to allow party *C* to have access to his keys for key recovery purposes, but *B* does not wish to let *A* know. Moreover, although this could be achieved by having *B* pass a copy of every key to *C*, this is potentially costly in communications and storage, and *B* and *C* wish to find an alternative. Suppose that the keys agreed between *A* and *B* are 64 bits long. Then, using essentially the same technique as described in [4], *B* may be in a position to choose every key shared with *A* such that the last 16 bits are a fixed function (known only by *B* and *C*) of the first 48 bits. When *C* wishes to recover a key, *C* needs only test at most 2^{48} possibilities for the key (e.g. using a known plaintext/ciphertext pair).

2. THE *COMMITMENTS* SOLUTION

To avoid the above problem, the use of *commitments* is proposed in [4]. The notion of *commitments* in cryptographic protocols is a well-established one, particularly in the context of zero-knowledge protocols (see, for example, [3]). By a commitment here we mean the disclosure of a value by an entity which binds that entity to a related value, without revealing that related value (in some contexts the disclosed value is referred to as a *witness*). The main idea behind using commitments to make key agreement schemes fair is to ensure that both parties choose their own key component before seeing the other party's component. This is

achieved by requiring one entity, say *B*, to hash its key component using a one-way hash function, and to send the resulting hash-code as the commitment to the other entity, say *A*, before *A* sends its own key component to *B*. Assuming that the key component contains sufficiently many bits, *A* cannot calculate *B*'s key component before choosing its own. Therefore, *A* has to generate and send its own key component without seeing the exact value of *B*'s component. After that *B* can pass its key component back to *A*, who hashes the value and checks whether the two hashed values match.

Most of the standardized key agreement mechanisms with joint key control described in ISO/IEC 11770-2 [1] and 11770-3 [2] can easily be adapted, using *commitments*, to provide 'fair key agreement'. However, of the seven key agreement methods specified in ISO/IEC 11770-3, the use of the commitment-based solution only applies to four of them; it does not work for the mechanisms involving use of pre-established key agreement key pairs (key agreement mechanisms 1–3 in ISO/IEC 11770-3). The main reason is that if one of the communicating parties has a public key agreement key certified by a CA, the other party can work on it for a long period of time before choosing its key component.

The purpose of this paper is to consider key agreement mechanism 1 in ISO/IEC 11770-3 and provide a solution to the 'fair key agreement' problem for this scheme. The proposed solution could more generally serve as a 'fair certification protocol' for Diffie-Hellman keys where the User does not choose his private key, but neither is the private key released to the CA. The mechanism could be deployed in environments where neither the CA nor the User trust each other to choose the User's key.

3. KEY AGREEMENT USING PUBLIC KEY CRYPTOGRAPHY

In the key agreement mechanisms described in ISO/IEC 11770-3 [2], both communicating parties contribute to the shared secret key, which is computed as a one-way function of the key components that the parties have chosen. The requirements for use of the mechanisms are given in ISO/IEC 11770-3 clauses 5 and 6. Most importantly, entities A and B using one of these protocols must have agreed on a function $F: H \times G \to G$, with the following properties.

1. F satisfies the commutativity condition $F(h_A, F(h_B, g)) = F(h_B, F(h_A, g))$.
2. It is computationally intractable to find $F(h_1, F(h_2, g))$ from $F(h_1, g)$, $F(h_2, g)$ and g. This implies that $F(\cdot, g)$ is a one-way function.

Also *A* and *B* must share an element g in G, which may be publicly known, and *A* and *B* must be able to efficiently compute values $F(h, g)$ and generate random elements in H. One 'obvious' candidate for F is to choose a large prime p, put $G = Z_p$, put $H = Z_p^*$, let g be a primitive element modulo p, and define

$$F(h, g) = g^h \bmod p$$

The seven key agreement mechanisms specified in ISO/IEC 11770-3 [2] can be divided into three classes.

- In one scheme (mechanism 1) the shared secret key is generated as a function of the two parties' pre-established key agreement keys. According to mechanism 1 two entities *A* and *B* non-interactively establish a shared secret key using their key agreement key pairs. Use of the mechanism requires each entity X to have a private key agreement key h_X in H and a public key agreement key $p_X = F(h_X, g)$, and both entities to have an authenticated copy of each other's public key agreement key [2, clause 6.1]. The mechanism involves the following steps:
  1. *A* computes, using its own private key agreement key h_A and *B*'s public key agreement key p_B, the shared secret key as $K_{AB} = F(h_A, p_B)$.
  2. *B* computes, using its own private key agreement key h_B and *A*'s public key agreement key p_A, the shared secret key as $K_{AB} = F(h_B, p_A)$.
- In two schemes (mechanisms 2 and 3) the shared secret key is generated as a function of one party's pre-established key agreement key, and the other party's dynamically generated component.
- In the other four schemes (mechanisms 4–7) the shared secret key is computed as a function of two dynamically generated components, one for each party.

As already discussed, the 'commitments' solution only applies to the third class of mechanisms. The fairness problem arises in the first class of mechanisms because one party may select his/her key agreement key pair so as to choose part of the key shared with another specified entity. Suppose entity *A* gets and publishes its public key agreement key. When *B* chooses its key agreement key it can ensure that the key established between *A* and *B* has certain properties. Of course, *B* has no control over keys established between himself and other entities, so the problem is restricted in scope. Nevertheless, *B* potentially has a long time in which to work on *A*'s key before choosing its own, and in this respect the problem is worse than for mechanisms 4–7. We now propose a solution to the first class of mechanisms, based on the idea of preventing a user choosing his/her key pair, whilst preserving the secrecy of the user's private key.

Finally note that we do not have a solution to the problem for the second class of mechanisms. This case appears particularly intractable, and if the lack of 'fairness' is a major problem then a mechanism from one of the other two classes should be used.

4. A FAIR CERTIFICATION PROTOCOL

We present a protocol between an entity and a CA which provides the entity with a private key and a certified public key, where the user cannot choose his/her private key but also no other entity (including the CA) knows the private key. Use of the proposed certification and key derivation mechanism requires the User and CA to share a secure channel and have agreed a modulus (a large prime number p), an element g of large multiplicative order q modulo p, and a collision-resistant hash-function h. The values p, g and h would typically be shared by a large domain of users, and could be distributed as part of an implementation of the scheme. Alternatively, the agreement of these values could be done using one of the mechanisms proposed in [3, §13.4]. The proposed protocol consists of the following steps.

- The User chooses a private key component *x*, computes $h(g^x)$, and sends it to CA.

$$U \xrightarrow{h(g^x)} CA$$

- CA chooses a second private key component *y*, and also computes $y^{-1} \bmod q$. The CA sends *y* to the User.

$$U \xleftarrow{y} CA$$

- The User computes its private key as $xy \bmod q$ and sends its public key g^{xy} to the CA.

$$U \xrightarrow{g^{xy}} CA$$

- The CA computes $h((g^{xy})^{y^{-1}})$ and checks that the result equals the value sent by the User in the first message. If the check is successful CA accepts g^{xy} as the public key of the User, certifies it, and returns the certificate to the User.

$$U \xleftarrow{Cert_{CA}(g^{xy})} CA$$

It is simple to verify that the User cannot choose the private key *xy*, and also that the CA does not know *xy*. Use of this protocol for the establishment and certification of all the users' key agreement keys ensures that a User cannot influence a key established with another User. In other words, the protocol prevents the User choosing his key agreement key to have certain specific properties [4]. Thus, in particular, *B* will not be able to choose part of the shared secret key established between *A* and himself, even if *B* has a long time to work on A's public key agreement key. However, although the User cannot control the generation of his key agreement key, the User's privacy is protected as neither the CA nor any other entity get knowledge of the generated private key agreement key. It is also clear that, through the use of 'commitments', the CA cannot choose part of the final key. This mechanism, as mentioned earlier, could more generally serve as a fair certification protocol for Diffie-Hellman keys in cases where neither User nor CA is trusted to choose the User's private key 'on their own'. As long as one party's contribution is random, the resulting key will be 'good'.

The value of *g* will typically be fixed for a particular application. If the multiplicative order of *g* (*q* say, where $q|(p-1)$) is non-prime, as would be the case if *g* were chosen to be primitive, then the User can have an influence on the value of their private key, albeit at the cost of choosing a rather 'weak' key. To see how this might arise, suppose *r* is a small prime dividing *q*. The user now chooses *x* so that g^x has order *r*. This is easily achieved by choosing a random value *z* $(0 < z < r)$ and putting $x = zq/r$. Whatever the CA chooses as the value *y*, the final private key will be one of a set of *r* possible values.

To avoid this pathological case it is necessary to choose *q* to be prime, and also for the CA to check that $g^x \neq 1$.

5. CONCLUSIONS

In this paper a new fair certification protocol was described, which enables a User to be provided with a certificate for a Diffie-Hellman public key such that the User does not choose his/her private key, but neither is this private key known to anyone other than the User. This protocol provides a solution to the fair key agreement problem for one of the three standardised mechanisms in ISO/IEC 11770-3 to which the commitment based solution does not apply.

6. REFERENCES

[1] International Organization for Standardization, Genève, Switzerland. ISO/IEC 11770-2, Information technology–Security techniques–Key Management–Part 2: Mechanisms using symmetric techniques, 1996.

[2] International Organization for Standardization, Genève, Switzerland. ISO/IEC 11770-3, Information technology–Security techniques–Key Management–Part 3: Mechanisms using asymmetric techniques, 1999 (to be published).

[3] Menezes, A.J., van Oorschot, P.C., and Vanstone, S.A. Handbook of Applied Cryptography. CRC Press, Boca Raton, 1997.

[4] Mitchell, C.J., Ward, M., and Wilson, P. Key control in key agreement protocols. Electronics Letters, **34**:980-981, 1998.