On One-Pass Authenticated Key Establishment Schemes

Kaisa Nyberg
Prinz Eugen-Straße 18/6, A-1040 Vienna, Austria
Email: `nyberg@ict.tuwien.ac.at`

21 April 1995[1]

Abstract

It is observed that time-variance is necessary in one-pass key establishment schemes. A weakness originally present in the Nyberg-Rueppel scheme can be removed by integration of a time-variant parameter to it. The recently proposed one-pass French scheme is shown to have a similar weakness. It is unknown whether it can be repaired. A new time variant one-pass key establishment scheme, whose security is not dependent on the basic Diffie-Hellman problem, is presented.

1 Introduction

For authenticated key establishment schemes three general requirements can be identified. The scheme must provide secrecy, authenticity and uniqueness of the established keys. In this contribution we understand the requirement for authenticity in the following sense: at the end of the execution of the scheme each participant knows that only the intended other participant can be in the possession of the established secret key. The requirement for uniqueness is generally taken to mean that the knowledge of some number of previously established keys cannot be used to violate secrecy and authenticity of any further keys. In other words, the key establishment scheme must resist known key attacks.

In this contribution authenticated one-pass key establishment schemes are considered. In such a scheme the initiator computes a session key and a related message, the key token, to be sent to the intended receiver using the receiver's public key and the sender's secret key. From the received key token the receiver computes the session key, which is the same as the one computed by the sender, using the sender's public key and the receiver's secret key.

[1] To Rainer Rueppel, on the occasion of his 40th anniversary.

One-pass key establishment schemes are useful for applications where the communication is *de facto* non-interactive. This is the case if the session key is used for authentication or encryption of a message which is either sent to the receiver for instant reception, like in electronic mail applications, or put in a secure file accessible to intended recipients at some later time.

In one-pass key establishment schemes particular precautions has to be taken to prevent from replay of old keys and corresponding key-tokens. This fraud becomes possible if only the sender is responsible for the refreshness of the keys. Since the receiver has no means to create refreshness, the only possibility is to use some time-variant parameter, which the sender cannot choose and whose validity can be verified by the receiver.

The first purpose of this contribution is to see what kind of possibilities there are to integrate time-variance to some previously proposed one-pass schemes based on discrete exponentiation. Secondly, we point out a weakness in a French one-pass scheme. There does not seem to be any natural ways to repair it either by integrating time-variance or by other means. Thirdly, we present a new time-variant one-pass key agreement scheme, which offers certain advantages over the other schemes and is essentially different. While the security of all other schemes discussed in this paper relies on the difficulty of solving a single Diffie-Hellman problem, the breaking of even one instance of the new scheme gives solution to this basic Diffie-Hellman problem.

2 Notation

The schemes discussed in this paper, with the exception of the French scheme, are defined in the ordinary Diffie-Hellman setting. Let p be a prime, q a divisor of $p-1$ and $g \in \mathbb{Z}_p$ an element of multiplicative order q. Each user A of the scheme has a secret key $x_A \in \mathbb{Z}_q$ and the corresponding public key of A is $y_A = g^{x_A} \bmod p$.

The security of most of the key establishment schemes in this setting is based on the assumed difficulty of the *basic Diffie-Hellman problem:* given g, y_A and y_B, find $y_{AB} = g^{x_A x_B} \bmod p$. We call y_{AB} the Diffie-Hellman key of A and B.

3 The First Scheme

A simple key agreement scheme can be based on an one-way collision-resistant hash-function h. Let y_{AB} be the Diffie-Hellman key of two users A and B. The key-token of the sender A contains just the user identifier of A and the valid value t of the time-variant parameter. Then parties A and B compute the key $K_{AB}(t) = h(y_{AB}, t)$. The security of this scheme relies on the difficulty of the basic Diffie-Hellman problem. In this sense, the schemes to be discussed in Sections 4,5 and 7, are not any better.

4 The Agnew-Mullin-Vanstone Scheme

In [1] proposed an one-pass authenticated key agreement scheme, where the participants compute the secret key from

$$K_{AB} = y_{AB}^k \bmod p$$

where k is a seed generated randomly by the sender and sent to the receiver encrypted using some public key encryption mechanism. As proposed in [1] it is natural to choose the ElGamal encryption system [3] for this purpose. There are several ways to integrate time-variance to this scheme. For example, if a hash-function is available, the sender computes the hash-value of the concatenation of k and the current value of the time-variant parameter and sends this hash-code along the encrypted k.

The security of this scheme also depends on the basic Diffie-Hellman problem. It is clear that anybody who is in possession of y_{AB} can impersonate A to B.

5 Integrating Time-Variance to the Nyberg- Rueppel Scheme

The following modification of the one-pass key agreement scheme [8] is presented in [7]. The general setting is as described in Section 2. In addition the users have agreed on a common hash-function h with values in $\mathbb{Z}_q$. Let the current value of the time-variant parameter be t. Then the sender A

- generates two integers K and k in $\mathbb{Z}_q$ randomly and secretly;
- computes $r = g^{K-k} \bmod p$;
- computes $r' = h(r, t)$;
- computes $s = k - x_A r' \bmod q$;
- sends (r, s) to the receiver B;
- computes $K_{AB} = y_B^K \bmod p$.

The receiver B of the token (r, s)

- computes $r' = h(r, t)$;
- computes $g^K = g^s y_A^{r'} r \bmod p$;
- computes $K_{AB} = (g^K)^{x_B} \bmod p$.

Note that the key token (r, s) is a Nyberg-Rueppel signature of A of the message g^K giving message recovery. From the expression of the key

$$K_{AB} = y_B^{k+s} y_{AB}^{r'} \bmod p$$

we see that impersonation of A to B becomes possible with the solution y_{AB} of the basic Diffie-Hellman problem.

In the original time-invariant scheme [8] it was possible to replay an r corresponding to a known key K, and by replacing the corresponding s by $s + u$ to establish a new key $K' = K y_B^u \bmod p$ with the receiver. Note that this fraud is not applicable to the corresponding two-pass scheme. A very similar known key attack can be launched against a recently proposed identity based one-pass key agreement scheme we discuss next.

6 The French Scheme

This identity-based public key agreement scheme was proposed be the French national body to the ISO/IEC SC27 [6]. Let n be a product of two primes, which are only known to an authority who provides the public keys for the users. The public system parameters constitute of n and two other integers e and g. Let A be a user of the system with identity I_A, an integer, and a secret key x_A, also an integer. Then the authority computes the public key of A

$$y_A = s_A g^{x_A} \bmod n,$$

where s_A is an integer such that

$$s_A^e = I_A^{-1} \bmod n.$$

The proposed one-pass scheme is as follows. The sender A

- generates an integer k randomly and secretly;
- computes $r = s_A g^k \bmod n$;
- sends r to the receiver B;
- computes the key $K_{AB} = (y_B^e I_B)^k = g^{ekx_B} \bmod n$.

When receiving r from A the receiver B computes the key from $K_{AB} = (r^e I_A)^{x_B} = g^{ekx_B} \bmod n$.

If an outsider C obtains a solution K_{AB} of even one instance of this key agreement between A and B, then C can impersonate A to B by sending $r' = rg^u \bmod n$ to B with any u. Then C can compute the key computed by B from r', since

$$((r')^e I_A)^{x_B} = K_{AB} g^{eux_B} = K_{AB} (y_B^e I_B)^u \bmod n.$$

It remains an open problem whether this scheme can be repaired to become resistant against this fraud, and against the replay attack specifically.

7 The Horster-Michels-Petersen Scheme

Recently Horster, Michels and Petersen presented in [5] a scheme, which they claim to be an improvement of the authenticated encryption scheme of Nyberg and Rueppel [9], since it requires lower communication costs. However, the Horster-Michels-Petersen scheme cannot really be considered to provide authenticated encryption in the same sense as the Nyberg-Rueppel scheme, since it may fail to offer the receiver B of the message any means of proving the origin A of the message to a third party without revealing the Diffie-Hellman key y_{AB}, which will ruin the system.

But the Horster-Michels-Petersen scheme can be useful for applications which do not provide non-repudiation, like for a secure transport of a symmetric session key. Also it offers natural possibilities for integration of time-variance. The general setting is as described in Section 2. In addition, the scheme employs an one-way function h with values in $\mathbb{Z}_p$.

Given a message $m \in \mathbb{Z}_p$ with sufficient redundancy, the sender A

- generates $k \in \mathbb{Z}_p$ secretly and randomly;
- computes $r = h(y_B^k)^{-1} m \bmod p$;
- computes $r' = r \bmod q$;
- computes $s = k - x_a r' \bmod p$;
- sends (r, s) to the receiver B.

When receiving (r, s) from A the receiver B

- computes $r' = r \bmod q$;
- recovers the message $m = h(y_B^s y_A^{r' x_B}) r \bmod p$.

When used for key transport, we propose to choose h to be a collision resistant hash function, and hash the items concatenated with the current value of the time-variant parameter.

Similarily, as for other schemes discussed above, we can see that the security of this key transport scheme depends on the secrecy of the Diffie-Hellman key y_{AB} as shown in [5]. Anybody in possession of y_{AB} can compute the redundant message (session key) $m = h(y_B^s y_{AB}^{r'}) r \bmod p$ from the transmitted token (r, s). If the application, where the session key is used, does not require the key to be chosen by A, which normally is the case, then this scheme does not offer any improvement over the previous schemes.

The particular feature, that only the intended receiver is able to verify the origin of the key token, can be developed further to design a new one-pass authenticated scheme with many advantages over the previous schemes.

8 A New Scheme

Let the key agreement setting be as described in Section 2 and let us assume that the users share a collision-resistant one-way hash function h with values in $\mathbb{Z}_q$. Then to establish a session key with B, when the current value of the time-variant parameter is t, the sender A

- generates $k \in \mathbb{Z}_p$ randomly and secretly;
- computes $r = g^k \bmod p$;
- computes the key $K_{AB} = y_B^k \bmod p$;
- computes $r' = h(K_{AB}, t)$;
- computes $s = k - x_A r' \bmod q$; and
- sends (r, s) to B.

When receiving (r, s) from A the receiver B

- computes the key $K_{AB} = r^{x_B} \bmod p$;
- computes $r' = h(K_{AB}, t) \bmod p$; and
- verifies the equality $r = g^s y_A^{r'} \bmod p$.

This scheme has some advantages over the previously discussed schemes. First, only one random number generation is needed. Secondly, the pair (K_{AB}, s) can be considered as the signature of A for the message t using Schnorr's signature scheme [10] with the generator element y_B. Hence the receiver B can explicitely verify the authenticity of the key token. Thirdly, breaking one instance of the scheme, that is, obtaining knowledge of one K_{AB} and the corresponding key-token (r, s) gives the solution y_{AB} of the basic Diffie-Hellman problem from the equation

$$K_{AB} = y_B^s y_{AB}^{h(K_{AB},t)} \bmod p.$$

On the other hand, it seems that the knowledge of y_{AB} is not sufficient to compute K_{AB} from (r, s). Impersonation of A in this scheme is not possible even to B, who can, by revealing K_{AB}, prove to any third party, first that $K_{AB} = r^{x_B} \bmod p$, and then that A has created the key token (r, s). Hence the security of this one-pass scheme is based on the underlying signature scheme.

Note that Schnorr's signatures can be replaced by other digital signatures that are defined in the setting of Section 2. For example, this scheme could be based on the DSS [4].

References

[1] G. B. Agnew, B. C. Mullin and S. A. Vanstone, *An interactive data exchange protocol based on discrete exponentiation*, Advances in Cryptology – Proceedings of Eurocrypt'88, Lecture Notes in Computer Science, Springer-Verlag (1989).

[2] W. Diffie and M. Hellman, *New Directions in Cryptography*, IEEE Trans. Inform. Theory, IT-22(6), November 1976, pp. 644-654

[3] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. Inform. Theory, IT-31(4), July 1985, pp. 469-472.

[4] *Digital Signature Standard (DSS)*, FIPS PUB 186, 1994 May 19.

[5] P. Horster, M. Michels and H. Petersen, *Authenticated encryption schemes with low communication costs*, Electronics Letters, September 1994.

[6] *Summary of voting on Letter Ballot N 866 "2nd ISO/IEC CD 11770-3, Key Management - Part 3: Mechanisms using asymmetric techniques"*, ISO/IEC JTC 1/SC 27 N 932, 1994-10-10.

[7] *3rd ISO/IEC CD 11770-3, Key Management - Part 3: Mechanisms using asymmetric techniques*, ISO/IEC JTC 1/SC 27 N 974, 1995-01-03.

[8] K. Nyberg and R. A. Rueppel, *A New Signature Scheme Based on the DSA Giving Message Recovery*, 1st ACM Conference on Computer and Communications Security, Nov 3-5, 1993, Fairfax, Virginia.

[9] K. Nyberg and R. A. Rueppel, *Message recovery for signature schemes based on the discrete logarithm problem*, to appear in Advances in Cryptology - Proceedings of Eurocrypt'94, Lecture Notes in Computer Science, Springer-Verlag.

[10] C. P. Schnorr, *Efficient signature generation by smart cards*, J. Cryptology, 4, 1991, pp. 161-174.

Server-Aided Secret Key Exchange

Chae Hoon Lim and Pil Joong Lee
Department of Electrical Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, 790-784, KOREA

Abstract

Diffie-Hellman-type key exchange protocols require modular exponentiation operations, which are time-consuming to be performed on a weak power device such as a smart card. In this paper, we consider the problem of accelerating smart card computation of shared secrets. We first propose and analyze two protocols for authenticated key exchange (two-pass and three-pass protocols) and then present their server-aided versions. The proposed server-aided key exchange(SAKE) protocols only require a small amount of additional communication but substantially reduce the smart card's computational load. For example, in one of the proposed methods, the smart card can compute a shared secret in about 50 multiplications on average while achieving the cryptanalytic complexity of more than 2^{60} operations. This performance may be further improved by increasing the communication complexity.

1 Introduction

One of the elementary prerequisites for secure communications is to establish a shared secret between two communicating partners. Such a shared secret key can be used to provide secrecy or integrity of a transmitted message by enciphering the message or generating MAC (message authentication code) with symmetric cryptosystems such as DES.

Secure key exchange needs authentication of communicating partners to prevent impersonation. Key exchange protocols with implicit authentication may fail under some sophisticated attacking scenarios (e.g., see [1]). Thus it would be preferable to provide explicit authentication, whenever possible, at the cost of some more computation and communication. It is quite easy to achieve authenticated key exchange with a slight modification of authentication protocols. By the end of a successful run of such protocols, two involved communicants will end up in possession of a shared secret.

Another threat to key exchange protocols may result from disclosed session keys [2,3]. Lost or leaked session keys may be exploited to compute a session key in a future session simply by observing the conversation or by impersonating a legitimate party [2,4,5]. Thus session keys should not be related to each other so that possession of old session keys does not make it any easier to compute another session key. In practical implementations, this kind of attack can be prevented in most cases by computing a session key as one-way function of the shared secret (e.g., as a hash-value of the shared secret), but still necessitates special precautions under some circumstances.

Once security is assured, it is the next objective in designing a key exchange protocol to enhance its efficiency as possible as one can. However, most key exchange protocols based on the discrete logarithm problem require at least one modular exponentiation, which is time-consuming to be performed on a weak power device such as a smart card. This motivated us to devise a method for accelerating the key exchange procedure on smart cards. Since the smart card only communicates with the outside world through a powerful terminal (such as PC or workstation), it may perform the required secret computation by borrowing the computing power of the (untrusted) powerful terminal without revealing its secret information.

This kind of client-server based computation model, called the server-aided secret computation, has been studied by many researchers in recent years, mainly focused on the RSA signature generation [6-11]. On the other hand, the present authors recently have developed efficient protocols for smart card verification of identity proofs and signatures with the aid of the prover/signer [12]. Compared to the server-aided secret computation protocol, the developed sever-aided public verification protocol requires much less communication and thus more practical for smart card implementations.

In this paper we consider another important, yet untouched application area of server-aided secret computation, i.e. the problem of speeding up secret key establishment on a smart card with the aid of a powerful terminal. Two protocols are proposed for Diffie-Hellman-based authenticated key exchange and server-aided versions of these protocols are then presented. It doesn't matter with which the smart card tries to agree a secret key. The communicating partner may also be another smart card. Then each smart card can interact with its own terminal only during the server-aided computation.

The rest of this paper is organized as follows. Section 2 introduces some conventions used in this paper and describes some exponentiation algorithms for evaluating multiple exponential terms. Section 3 describes two protocols for authenticated key exchange, one with two moves and the other with three moves. In Section 4 we present server-aided versions of the proposed key exchange protocols. Finally we conclude in Section 5.

2 Preliminaries

All the protocols presented in this paper will be illustrated using Schnorr's scheme [13], but they can be constructed with any signature scheme based on the discrete logarithm problem. Throughout this paper, we will use the following conventions.

Let p and q be two large public primes such that q divides $p-1$ and g be an element of order q in $\mathbf{Z}_p$. We denote the bit-length of p (q, resp.) by m (n, resp.) (i.e., $|p| = m, |q| = n$). The reference size of m and n for our illustration will be taken as 512 and 160 respectively, as in the digital signature standard (DSS) [14]. Let (s_A, v_A) be the secret and public key pair of user A, where $v_A = g^{-s_A} \bmod p$ with $s_A \in \mathbf{Z}_q$. We assume that precomputation of random powers to the fixed base g is performed in advance and thus does not take time during the protocol execution. Multiplication will always denote multiplication mod p and multiplication mod q will be neglected when counting the number of multiplications.

We next briefly describe methods for evaluating $w = \prod_{i=1}^{N} g_i^{x_i} \bmod p$ with $|x_i| = t$, which will be needed for the performance analysis of the proposed SAKE protocols. Here we assume that the most significant bit of an exponent is always one for completeness, though their effect on the performance is negligible. For $N = 2$, one can compute $w = g_1^{x_1} g_2^{x_2} \bmod p$

using the binary method, known as the square-and-multiply algorithm, where a frequently used value $g_1 g_2 \bmod p$ is first computed and stored for later use. With this method, w can be computed in $1.75t - 0.75$ multiplications on average ($2t-1$ in the worst case). Similarly, one can compute $w = g_1^{x_1} g_2^{x_2} g_3^{x_3} \bmod p$ in about $1.875t + 2.125$ multiplications on average ($2t+2$ in the worst case) using four precomputed values (all possible combinations of g_i's).

As the number of exponential terms increases, this method becomes inefficient in both computation time and storage usage. For large values of N, there exists a more efficient way to evaluate w. For this, we arrange the N exponential terms into groups consisting of almost equal terms, prepare products of all possible combinations of base elements in each group and then apply the square-and-multiply algorithm [15]. As an example, we explain this method for $N = 5$. That is, we want to compute

$$w = \prod_{i=1}^{5} g_i^{x_i} = (g_5^{x_5} g_4^{x_4} g_3^{x_3}) \cdot (g_2^{x_2} g_1^{x_1}) \bmod p.$$

Let $(x_{it} \cdots x_{i2} x_{i1})_2$ be the binary representation of x_i where $x_{ij} \in \{0,1\}$. We first compute and store the following values

$$G[1][I] = g_5^{i_5} g_4^{i_4} g_3^{i_3} \bmod p, \quad G[0][J] = g_2^{i_2} g_1^{i_1} \bmod p,$$

where $I = (i_5 i_4 i_3)_2$ and $J = (i_2 i_1)_2$ with $i_j \in \{0,1\}$. This precomputation requires 5 multiplications. Using this precomputed table, we execute the following algorithm.

```
$w := G[1][7]G[0][3] \bmod p$;
for i := t-1 to 1 step -1
    $w := w^2 \bmod p$;
    $I := (x_{5i} x_{4i} x_{3i})_2$;  $J := (x_{2i} x_{1i})_2$;
    $w := wG[1][I]G[0][J] \bmod p$;
return($w$);
```

The above algorithm can computes w in $2.625(t+1)$ multiplications on average, including the number of multiplications required for precomputation. It also requires a temporary storage for 10 precomputed values. The performance of the proposed server-aided protocols will be evaluated based on the described exponentiation methods.

3 Protocols for Authenticated Key Exchange

3.1 One/Two-pass protocol

Let h be a one-way hash function producing l-bit digests and f be a permutation on a set of l-bit numbers. The parameter l determines the security level of the protocol. Suppose that users A and B want to establish an authenticated session key. Then the two users conduct the following protocol.

1) User A randomly picks R_A and r_A in $\mathbf{Z}_q$ and computes $K_A = v_B^{R_A} \bmod p$, $x_A = g^{-r_A} \bmod p$, $e_A = h(K_A \oplus x_A, T_A)$ and $y_A = r_A - R_A + s_A e_A \bmod q$, where $\oplus$ denotes bitwise exclusive-or and T_A a timestamp. Then A sends $\{x_A, e_A, y_A, T_A\}$ to user B.

2) User B first checks the timeliness of T_A with respect to local clock. Then B computes $K_B = (g^{y_A} v_A^{e_A} x_A)^{s_B} \bmod p$ and checks that $e_A = h(K_B \oplus x_A, T_A)$. If the check is successful, B computes $e_B = h(K_B, f(T_A))$ and sends it back to A.

3) User A checks that $e_B = h(K_A, f(T_A))$.

At the end of successful completion, A and B will obtain the authenticated secret key, $K_{AB} = K_A = K_B$, from which a session key can be derived for use in a subsequent secure communication. For example, the session key can be computed as $K_{AB} \bmod q$. For one-way communications, step 3) can be omitted. Then authentication of A to B becomes implicit. The protocol is quite efficient. A (B, resp.) only needs to perform about $1.5(n-1)$ ($1.875n + 2.125$, resp.) multiplications on average and the number of communication bits is $m + n + 3l$ (excluding public key related information).

The hash function h may be replaced by other simpler functions. An example is to compute e_A as $K_{A1} \oplus x_{A1} + K_{A2} \oplus T_A$, where K_{Ai} denotes the i-th l-bit block of K_A, and e_B as $e_B = K_{B3} \oplus T_A + K_{B4}$. In this case, the timestamp T_A needs not be transmitted since it can be recovered from e_A. Note also that e_A and e_B release almost no information on the secret K_{AB}.

The above protocol is designed so that it can resist any conceivable attack. The replay attack can be detected by the use of timestamp. It also resists known-key attacks due to the use of two distinct random numbers for authentication and key exchange(see [5]). Explicit authentication will prevent any impersonation attack. The most promising way to impersonate user A would be to find a pair $\{e_A, y_A\}$ satisfying $e_A = h(v_B^{y_A + r_A} \bmod p \oplus v_A^{-e_A} g^{r_A} \bmod p, T_A)$ for random r_A and some T_A containing future time information. This requires about 2^l operations and can be made infeasible, for example, by taking $l = 64$. Note that if K_A is not involved in the computation of e_A, then an imposter may pass the check of step 2) though he cannot compute the shared secret.

To the authors' opinion, it seems unlikely that the whole secret key K_{AB} would be disclosed in most applications, since the session key is usually much less than the modulus size and thus it can be obtained by applying some one-way functions (e.g., one-way hash functions) or it can be simply computed as $K_{AB} \bmod q$ (of course, more care should be taken under some circumstances such as negotiation of contracts, see [1,3]). If this is the case, the above protocol can be further simplified as follows.

1) User A randomly picks r_A in $\mathbf{Z}_q$ and computes $K_A = v_B^{-r_A} \bmod p$, $e_A = h(K_A, T_A)$ and $y_A = r_A + s_A e_A \bmod q$. Then A sends $\{e_A, y_A, T_A\}$ to user B.

2) After verifying the timeliness of T_A, user B computes $K_B = (g^{y_A} v_A^{e_A})^{s_B} \bmod p$ and checks that $e_A = h(K_B, T_A)$. If the check succeeds, B computes $e_B = h(K_B, f(T_A))$ and sends it back to A.

3) User A checks that $e_B = h(K_A, f(T_A))$.

This variant achieves somewhat better efficiency in computation and communication, and particularly has additional advantage in view of security. To impersonate user A, an attacker has to find $\{e_A, y_A, T_A\}$ such that $e_A = h((g^{y_A} v_A^{e_A})^{s_B} \bmod p, T_A)$. However, solving this equation is impossible, irrespective of the size of e_A, as far as the discrete logarithm problem is intractable. Thus the size of e_A and e_B may be further reduced (say, of 40 bits).

3.2 Three-pass protocol

We now present a three-pass variant of the protocol described above. In this protocol, the Schnorr identification scheme is applied to user A while user B still employs the signature variant. Consequently, the use of timestamp is unnecessary. In general, using nonces (random numbers) is regarded as a better way to assure freshness of conversations and thus the resulting protocol will be more robust against replay attacks (see [16-18] for various ways for freshness assurance and their comparison). Note that K_{Ai} denotes the i-th l-bit block of K_A.

1) User A randomly picks R_A and r_A in $\mathbf{Z}_q$ and computes $x_A = g^{R_A} \bmod p$ and $e_A = (g^{r_A} \bmod p) \bmod 2^L$. Then A sends $\{x_A, e_A\}$ to user B.

2) User B randomly picks r_B in $\mathbf{Z}_q$ and computes $K_B = x_A^{r_B} \bmod p$, $e_B = K_{B1} \oplus K_{B2}$ and $y_B = r_B + s_B e_B \bmod q$. Then B sends $\{y_B, e_B\}$ to user A.

3) User A computes $K_A = (g^{y_B} v_B^{e_B})^{R_A} \bmod p$ and checks that $e_B = K_{A1} \oplus K_{A2}$. If the check succeeds, A computes $y_A = r_A + s_A e_B \bmod q$ and sends it back to user B.

4) User B checks that $e_A = (g^{y_A} v_A^{e_B} \bmod p) \bmod 2^L$.

The soundness of Schnorr's identification scheme guarantees that an attacker cannot succeed in impersonating user A with probability better than by guessing e_B jointly determined by A and B. It will suffice to take l around 40. Small values of L may reduce the security level as noted in [19], but $L = 128$ would be sufficient. As mentioned before, masquerading as user B is impossible unless the discrete logarithm problem is easy. The above protocol also uses two distinct random numbers for authentication and secret key exchange to avoid the known-key attack. As before, after successful completion of the protocol, both users can compute a session key as $K_{AB} \bmod q$ where $K_{AB} = K_A = K_B$.

4 Server-Aided Computation of Shared Secrets

This section deals with a method for speeding up the smart card computation of shared secrets in the key exchange protocols described before. Let user A be the smart card and user B be any party with which the smart card wants to share a secret session key. For the sake of convenience, let us call the two-pass protocol as 2PP and the three-pass protocol as 3PP. The smart card has to compute

$$\begin{aligned} K_A &= v_B^{R_A} \bmod p \text{ in 2PP and} \\ K_A &= (g^{y_B} v_B^{e_B})^{R_A} \bmod p \text{ in 3PP.} \end{aligned}$$

Direct computation with the square-and-multiply algorithm requires about 238.5 multiplications in 2PP and 279.25 multiplications in 3PP, on average (for $n = |q| = 160$). These amounts of computation are clearly too much for smart cards under current technology.

However, since the smart card only communicates with the outside world through a powerful terminal, it can borrow the computing power of the terminal to reduce the required number of multiplications. This will of course somewhat increase the communication complexity. As is clear from the above key computation equations, such a server-aided key exchange(SAKE) protocol requires somewhat different approach from that for RSA computation. That is, in addition to protecting the involved secret R_A, the SAKE protocol also has to resist the impersonation attack which can be mounted during the server-aided protocol. This is a major difference from the server-aided RSA computation protocol.

4.1 Basic SAKE Protocols

Let us first consider the server-aided computation of $K_A = v_B^{R_A} \bmod p$ in 2PP. Before begining communication with user B, the smart card (user A) conducts the following protocol with the server.

SAKE Protocol for 2PP

1) The smart card randomly picks $k_i \in [0, 2^t)$ $(0 \le i \le N)$ and $u_i \in \mathbf{Z}_q$ $(1 \le i \le N)$ such that
$$R_A - k_0 = \sum_{i=1}^{N} k_i u_i \bmod q. \tag{1}$$
Then the smart card sends $\{u_i\}$ to the server.

2) The server computes $w_i = v_B^{u_i} \bmod p$ for each i and sends $\{w_i\}$ back to the smart card.

3) The smart card then computes K_A as
$$K_A = v_B^{k_0} \prod_{i=1}^{N} w_i^{k_i} \bmod p. \tag{2}$$

We first examine two security aspects of the above protocol, the probability of impersonation and the attacking complexity for finding the secret R_A, and then analyze its performance.

On-line Attack for Impersonation : Suppose that the server tries to impersonate user B by manipulating its transmissions. As can be seen from (2), this is possible only if it could replace v_B with any value whose logarithm it knows, say $\beta = g^{R_S} \bmod p$ with random $R_S \in \mathbf{Z}_q$. The only way to achieve this is to guess k_0 and one other k_i for some i (say, $i = 1$) and then return w_i's such that $w_1 = \beta^{u_1 + k_1^{-1} k_0} v_B^{-k_1^{-1} k_0} \bmod p$ and $w_i = \beta^{u_i} \bmod p$ for $i = 2, \ldots, N$. Then the server will be able to share the session key $K_A = (g^{R_A})^{R_S} \bmod p$ with the smart card since it can compute g^{R_A} as $g^{y_A} v_A^{e_A} x_A \bmod p$. However, this attack can be successful only with probability 2^{-2t}. Practically, a guessing probability of 10^{-9} (i.e., $t = 15$) would be sufficient in most applications.

Off-line Attack for Finding R_A : The server (any eavesdropper, indeed) may try to find R_A itself from (1). Equation (1) can be rearranged as
$$R_A - k_0 - \sum_{i=1}^{N/2} k_i u_i = \sum_{i=N/2+1}^{N} k_i u_i \bmod q, \tag{3}$$

where "/" denotes integer division. Then one can obtain, by raising g to each side of (3), the equation

$$g^{R_A} \cdot g^{-k_0} \cdot \prod_{i=1}^{N/2} w_i^{k_i} = \prod_{i=N/2+1}^{N} w_i^{k_i} \bmod q, \tag{4}$$

Now the server can exhaustively search for k_i's based on (4) and then compute R_A from (3). That is, the server enumerates both sides of (4) for all possible values of $k_i \in [0, 2^t)$ and searches for a common value. The enumeration requires about $K = 2^{(1+N/2)t}$ multiplications and sorting and merging the enumerated sets requires about $K \log_2 K$ comparisons and the storage of order K.

Suppose that we want to achieve $K > 2^{60}$, which results in the actual computational complexity of about 2^{66} operations. Then we have to choose the parameter t such that $(1 + N/2)t > 60$ for given N. It will be preferable to choose N even, as can be seen from (4). For example, we get $t = 60, 30, 20$ for $N = 1, 2, 4$ respectively. Note that knowledge of R_A only disclose the secret K_A of that session but does not affect the secret key s_A of user A. This is also true for 3PP.

Performance : Now let us consider the performance of the basic SAKE protocol. First note that N cannot be taken large for practicality since the smart card has to evaluate $N + 1$ small powers. We only consider three values of N, i.e. $N = 1, 2$, and 4. To compute the shared secret K_A using (2), we directly apply the square-and-multiply algorithm for $N = 1$ and $N = 2$. For $N = 4$, we use the method in [15] where the five exponential terms are divided into two groups consisting of two and three exponential terms and then the square-and-multiply algorithm is applied (see section 2).

The following table summarizes the performance of the protocol. The number of multiplications required of the smart card is evaluated for the average case only. The storage column only shows the number of n-bit storage for temporarily storing the precomputed values required for exponentiation. The figures in the rightmost two columns are computed for a security level of 2^{60}. Finally, note that we are using m and n as the bit-lengths of p and q respectively.

N	Mul	Storage	Commun	t	Mul
1	$1.75t - 0.75$	3	$m + n$	60	104.25
2	$1.875t + 2.125$	7	$2(m + n)$	30	58.38
4	$2.625(t + 1)$	10	$4(m + n)$	20	55.13

Table 1. Performance of the basic SAKE protocol for 2PP

¿From this table, we can see that if the smart card has a scratch-pad memory (RAM) for about ten values of m bits ($N = 2$), it can compute the shared secret K_A in less than 60 multiplications on average. This gives about a four-fold improvement over direct computation by the binary method only using a small amount of communication. The next section will describe enhanced SAKE protocols giving better performance.

Next, we consider the server-aided computation of $K_A = (g^{y_B} v_B^{e_B})^{R_A} \bmod p$ in 3PP. The only difference from the SAKE protocol for 2PP is that the term $g^{y_B R_A} \bmod p$ needs to be computed in addition. We can compute this power almost for free using precomputation. The smart card executes the following protocol with the server after step 2) in 3PP.

SAKE Protocol for 3PP

0) The smart card randomly picks $K \in \mathbf{Z}_q$, computes $z = g^K \bmod p$ and securely stores the pair $\{K, z\}$.

1) The smart card randomly picks $k_i \in [0, 2^t)$ $(0 \le i \le N)$ and $u_i \in \mathbf{Z}_q$ $(1 \le i \le N+1)$ such that

$$\begin{aligned} R_A - k_0 &= \sum_{i=1}^{N} k_i u_i \bmod q, && (5) \\ u_{N+1} &= y_B R_A - K \bmod q. && (6) \end{aligned}$$

Then the smart card sends $\{u_i\}$ to the server.

2) The server computes $w_i = v_B^{u_i} \bmod p$ $(1 \le i \le N)$ and $w_{N+1} = g^{u_{N+1}} \bmod p$ and then sends $\{w_i\}$ back to the smart card.

3) The smart card then computes K_A as

$$K_A = z w_{N+1} v_B^{k_0} \prod_{i=1}^{N} w_i^{k_i} \bmod p. \tag{7}$$

Since step 0) can be carried out ahead of time, the smart card can compute K_A as in the SAKE protocol for 2PP, only by increasing two multiplications due to w_{N+1} and z. Note that u_{N+1} provides no additional information on R_A due to the involvement of another random number K. As for security, we have to mention one more point. For the server to impersonate user B by the real-time attack explained before, it is now sufficient to guess k_0 only, since w_{N+1} is not raised to random power. Thus the success probability for the attack becomes 2^{-t} rather than 2^{-2t}. This should be kept in mind when determining the size of t.

With some increase of computational amount, one can achieve the same imposter pass rate as before. For this, the smart card randomly selects $k_{N+1} \in [0, 2^t)$ additionally and computes u_{N+1} as $u_{N+1} = (y_B R_A - K) k_{N+1}^{-1} \bmod q$. Then K_A can be computed as

$$K_A = z v_B^{k_0} \prod_{i=1}^{N+1} w_i^{k_i} \bmod p. \tag{8}$$

Thus the number of small powers to be evaluated is increased by one. This will somewhat degrade the performance. Note, however, that we need not to take this alternative for the basic SAKE protocol with $N = 2$, which seems most suitable for smart cards in computation time and storage usage, since we already have $t = 30$ for this case.

4.2 Improved SAKE Protocols

We can further reduce the number of multiplications required of the smart card in the basic SAKE protocols by using more communication. We only consider variants of the SAKE protocol for 2PP. The following is a generalized SAKE protocol for 2PP.

1) The smart card randomly picks $k_i \in [0, 2^t)$ $(0 \leq i \leq N)$ and $u_j \in \mathbf{Z}_q$ $(1 \leq j \leq M)$ such that

$$R_A - k_0 = \sum_{i=1}^{N} k_i \left(\sum_{j=1}^{M} f_{ij} u_j \right) \bmod q, \tag{9}$$

where $f_{ij} = \sum_{l=1}^{L} f_{ijl} 2^{l-1}$ with $f_{ijl} \in \{0, 1\}$, L-bit secret integers. Then the smart card sends $\{u_j\}$ to the server.

2) The server computes $w_j = v_B^{u_j} \bmod p$ for each j and sends $\{w_j\}$ back to the smart card.

3) The smart card then computes K_A as

$$K_A = v_B^{k_0} \prod_{i=1}^{N} \left(\prod_{j=1}^{M} w_j^{f_{ij}} \right)^{k_i} \bmod p. \tag{10}$$

The only difference from the basic SAKE protocol is a further decomposition of u_i's. This somewhat increases the communication complexity, but, as will be seen below, considerably reduces the computation time. All three parameters M, N and L are closely related to the performance of the protocol (i.e., the number of multiplications, communication bits and storage required of the smart card). For performance reason, we will choose L-bit numbers f_{ij} sparse. That is, the total weight of NML binary numbers f_{ijl}'s will be limited to a certain value W. We only consider two values of N, $N = 1$ and $N = 2$, since larger values of N result in smaller t's and thus increase the success probability of impersonation.

Computing K_A : Equation (10) can be rewritten as

$$K_A = v_B^{k_0} \prod_{i=1}^{N} \left\{ \prod_{l=1}^{L} \left(\prod_{j=1}^{M} w_j^{f_{ijl}} \right)^{2^{l-1}} \right\}^{k_i} \bmod p. \tag{11}$$

Using this expression, we can compute K_A as follows. The smart card first accumulates $z_{il} = \prod_{j=1}^{M} w_j^{f_{ijl}} \bmod p$ for each i and l. This accumulation requires approximately $W - NL$ multiplications. Next it computes $z_i = \prod_{l=1}^{L} z_{il}^{2^{l-1}} \bmod p$ for each i, which requires at most $2N(L-1)$ multiplications. The total number of multiplications required to compute all z_i's can be shown to be at most $W + N(L-2)$. Now K_A is computed as $K_A = v_B^{k_0} \prod_{i=1}^{N} z_i^{k_i}$ $\bmod p$. Therefore, the total expected number of multiplications required of the smart card is at most

$$1.75t + W + L - 2.75 \text{ for } N = 1 \text{ and} \tag{12}$$
$$1.875t + W + 2L - 1.875 \text{ for } N = 2. \tag{13}$$

Parameter Selection : To determine appropriate values of M, L, W and t for a given N, we solve the following minimization problem.

- Conditions :
  - Either M or W must be chosen small, say less than 10, considering the space limitation of typical smart cards.

- The probability of successful impersonation must be smaller than 10^{-9}. For this, we simply choose t greater than or equal to 15.
- The complexity for finding R_A must be greater than 2^{60} operations. For this, we simply require the number of possible values to be sorted to be greater than 2^{60}.

• Objective :

- Minimize the required number of multiplications given in (12) and (13).

As for the first condition, we would like to mention the following note [11]. During the accumulation of $z_{il} = \prod_{j=1}^{M} w_j^{f_{ijl}} \bmod p$ for each i and l, the server may monitor the duration of the smart card's computation in order to deduce the weights of f_{ijl}'s. Thus, to avoid this kind of implementation-dependent attack, the smart card must either spend the same amount of time on each step or take other measures. We consider two methods of restricting parameter selection to get better performance. The smart card may compute z_{il}'s after receiving all w_j's at a time. This requires M to be chosen small. Alternatively, the smart card may store w_j's with non-zero f_{ijl}'s and then compute z_{il}'s. This requires W to be chosen small. These are the reason why we take either M or W small.

Analyzing Performance : To estimate the attacking complexity for the case of $N = 1$, we have to consider (9) by rewriting it as

$$R_A - k_0 - k_1 \sum_{j=0}^{M_1} f_{1j}u_j = k_1 \sum_{j=M_1+1}^{M} f_{1j}u_j \bmod q, \tag{14}$$

where M_1 is a number less than $M/2$. Then, to achieve the required security level of 2^{60} operations, we can easily see that given M, L, W and $t \geq 15$, the following inequality must be satisfied for all $M_1 < M/2$.

$$\sum_{j=1}^{W}\sum_{k=0}^{j}\left\{2^{2t}\binom{M_1L}{k} + 2^t\binom{(M-M_1)L}{j-k}\right\} > 2^{60} \tag{15}$$

For $N = 2$, we rewrite (9) as $R_A - k_0 - k_1 \sum_{j=1}^{M} f_{1j}u_j = k_2 \sum_{j=1}^{M} f_{2j}u_j \bmod q$ and obtain the following inequality to be satisfied.

$$\sum_{j=2}^{W}\sum_{k=1}^{j-1}\left\{2^{2t}\binom{2ML}{k} + 2^t\binom{2ML}{j-k}\right\} > 2^{60} \tag{16}$$

Now, our objective is to minimize the required number of multiplications given in (12) and (13) under the constraints (15) and (16). Table 2 summarizes some selected parameters for small M's. As can be seen from (12) and (13), there may exist many pairs of (W, L) giving the same performance. We have chosen the smallest L among such pairs. Note that the number of communication bits is determined by M and is given by $M(m + n)$. From this table we can see that with a small increase of communication the required number of multiplications can be substantially reduced compared to the basic SAKE protocol, especially for the case of $N = 1$.

	$N=1$				$N=2$			
M	L	W	t	Mul	L	W	t	Mul
5	14	15	15	52.50	7	13	15	53.25
6	14	13	16	52.25	6	11	16	51.12
7	12	14	15	49.50	5	11	16	49.12
8	10	15	15	48.50	5	11	15	47.25
9	11	12	15	46.50	5	10	15	46.25

Table 2. Selected parameters I for the improved SAKE protocol

If the communication cost is much lower than the computation cost, we may further increase M to get better performance. For example, Table 3 shows some selected parameters for $M = 30$. In this case, we have chosen W small and thus the smart card can receive and store all w_j's with non-zero f_{ijl}'s at a time and then perform the required computation.

	$N=1$			$N=2$		
W	L	t	Mul	L	t	Mul
5	13	24	57.25	3	20	46.62
6	14	18	48.75	2	19	43.75
7	12	15	42.50	3	16	41.12
8	8	15	39.50	2	16	40.12
9	6	15	38.50	2	15	39.25
Common : $M = 30$						

Table 3. Selected parameters II for the improved SAKE protocol

Reducing the Server's Computational Load : We finally consider the number of multiplications required of the server and suggest a means of reducing the server's load. The server has to perform M exponentiations of n-bit exponents. This will require about an average of $(1.75n - 0.75)M$ ($279.25M$ for $n = 160$) multiplications with the binary algorithm. However, since these exponentiations are based on the fixed number v_B, we can apply the precomputation techniques [15,20,21] to reduce the number of multiplications. For example, with one of the methods from [15], this computation can be completed in at most $41M + 228$ multiplications for $n = 160$, where 228 accounts for the number of multiplications needed to prepare the precomputation table. Even for $M = 10$, this gives better than a four-fold speedup.

On the other hand, we can further reduce the server's computational load by using u_j's with a special structure. Note first that u_j's need not be chosen at random. It is sufficient for the weighted numbers $f_{ij}u_j$'s to be distinct. Thus we may choose u_j's as $u_j = 2^{(j-1)L}$ for $j = 1, \cdots, M-1$ and compute the final value u_M from (9). In this case we have to increase M by one since the last value must always have non-zero weight. For example, applying this method for $N = 2$, equation (9) can be expressed as

$$\begin{aligned} R_A - k_0 &= k_1(f_{11} + f_{12}2^L + \cdots + f_{1M}2^{(M-1)L}) \\ &+ k_2(f_{21} + f_{22}2^L + \cdots + f_{2M}2^{(M-1)L} + u_{M+1}). \end{aligned} \quad (17)$$

¿From this equation, u_{M+1} can be computed and transmitted to the server.

Now the server can compute w_j's as $w_2 = v_B^{2^L} \bmod p$, $w_j = w_{j-1}^{2^L} \bmod p$ for $j = 3, \cdots, M-1$ and $w_M = v_B^{u_M} \bmod p$. Note that $w_1 = v_B$. All these computations require $(L-1)M + 1.75n - 0.75$ multiplications on average. For the parameters given in Table 3, L takes at most 14 and even for this worst case the server only needs to perform 669.25 multiplications on average.

5 Conclusion

We can expect that smart cards will be rapidly prevalent in the near future as pocket computers for performing secret cryptographic operations, due to its high security and portability. Smart card systems provide good computing resources easily accessable to a weak power smart card, since the smart card only communicates with the outside world through the powerful terminal. Diffie-Hellman-type key exchange protocols require modular exponentiation operations, which take a considerable time on typical smart cards. In this paper we have proposed two protocols for Diffie-Hellman-based authenticated key exchange and then presented their server-aided versions. The proposed server-aided protocols were shown to substantially speed up the smart card computation of shared secrets only with a small increase of communication. We believe that our protocols will be useful for practical implementations of most key exchange protocols on smart cards.

References

[1] M.Burmester : 'On the risk of opening distributed keys', *Proc. of Crypto'94*, LNCS 839, Springer Verlag, pp.308-317 (1994).

[2] Y.Yacobi : 'A key distribution paradox', *Proc. of Crypto'90*, LNCS 537, Springer Verlag, pp.268-273 (1991).

[3] Y.Desmedt and M.Burmester : 'Towards practical 'proven secure' authenticated key distribution', *Proc. of 1st ACM Conference on Computer and Communications Security*, ACM Press, pp.228-231 (1993).

[4] K.Nyberg and R.Rueppel : 'Weaknesses in some recent key agreement protocols', *Electronics Letters*, 30(1), pp.26-27 (1994).

[5] K.Nyberg and R.Rueppel : 'Message recovery for signature schemes based on the discrete logarithm problem', to appear in *Proc. of Eurocrypt'94* to be published by Springer Verlag.

[6] T.Matsumoto, K.Kato and H.Imai, Speeding up secret computations with insecure auxiliary devices, *Proc. of Crypto'88*, Springer-Verlag, LNCS 403, 497-506 (1990).

[7] J.J.Quisquater and M.De Soete, Speeding up smart card RSA computation with insecure coprocessors, *Proc. Smart Card 2000*, North-Holland, 191-197 (1991).

[8] B.Pfitzmann and M.Waidner, Attacks on protocols for server-aided RSA computation, *Proc. of Eurocrypt'92*, Springer-Verlag, LNCS 658 (1993).

[9] T.Matsumoto, H.Imai, C.S.Laih and S.M.Yen, On verifiable implicit asking protocols for RSA computation, *Proc. of Auscrypt'92*, Springer-Verlag, LNCS 718 (1993).

[10] S.Kawamura and A.Shimbo, Fast server-aided secret computation protocols for modular exponentiation, *IEEE J. Selected Areas in Commun.*, 11(5), 778-784 (1993).

[11] J.Burns and C.J.Mitchell : 'Parameter selection for server-aided RSA computation schemes', *IEEE Trans. Computers*, 43(2), 163-174 (1994).

[12] C.H.Lim and P.J.Lee : 'Server(prover/signer)-aided verification of identity proofs and signatures', to be presented at *Eurocrypt'95*.

[13] C.P.Schnorr, Efficient signature generation by smart cards, *Journal of Cryptology* 4(3) 161-174 (1991).

[14] NIST, Digital signature standard, *FIPS PUB 186* (1994).

[15] C.H.Lim and P.J.Lee, More flexible exponentiation with precomputation, *Advances in Cryptology-Crypto'94*, Springer-Verlag, LNCS 839, 95-107 (1994).

[16] L.Gong : 'Variations on the themes of message freshness and replay', *Proc. of Symposium on Research in Security and Privacy*, IEEE Press, pp.131-136 (1993).

[17] K.Y.Lam and T.Beth : 'Timely authentication in distributed systems', *Proc. ESORICS'92*, Springer Verlag, LNCS 648, pp.293-303 (1992).

[18] K.Y.Lam and D.Gollmann : 'Freshness assurance of authentication protocols', *Proc. ESORICS'92*, Springer Verlag, LNCS 648, pp.261-271 (1992).

[19] M.Girault and J.Stern : 'On the length of cryptographic hash-values used in identification schemes', *Proc. Crypto'94*, Springer Verlag, LNCS 839, pp.202-215 (1994).

[20] E.F.Brickell, D.M.Gordon, K.S.McCurley and D.B.Wilson : 'Fast exponentiation with precomputation', *Advances in Cryptology-Eurocrypt'92*, Springer-Verlag, LNCS 658, pp.200-207 (1993).

[21] P.de Rooij : 'Efficient exponentiation using precomputation and vector addition chains', to appear in *Proc. of Eurocrypt'94* to be published by Springer Verlag.

Some New Key Agreement Protocols Providing Implicit Authentication

Alfred J. Menezes*
Dept. of Discrete and Statistical Sciences
Auburn University, Auburn, AL 36849, USA

Minghua Qu
MÖBIUS Encryption Technologies
200 Matheson Boulevard, West, Suite 103
Mississauga, Ontario, Canada L5R 3L7

Scott A. Vanstone†
Dept. of Combinatorics and Optimization
University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

April 24, 1995

Abstract

There are a number of key agreement protocols in the literature and in use in practice which purport to give mutual implicit authentication. Examples include the Nyberg-Rueppel one-pass protocol [8], and the Matsumoto-Takashima-Imai (MTI) [5], Goss [4] and Yacobi [10] two-pass protocols for key agreement. In this paper we describe new attacks on these systems which demonstrate that implicit authentication is in fact not obtained. In some cases we show how the protocols can be extended to provide the desired property. We also present three new protocols which are very efficient and give mutual implicit authentication; the first two have two passes, while the third is a one-pass protocol.

1 Introduction

Key establishment is the process by which two (or more) parties establish a shared secret key, called the *session key.* The session key is subsequently used to achieve some cryptographic goal, such as privacy. A key establishment protocol is said to provide *implicit key authentication* (or simply *key authentication*) if one party is assured that no other party aside from a specially identified second party may learn the value of the session key. Note that the property of implicit key authentication does not necessarily mean that the first party is assured of the second party actually possessing the session key. A key establishment protocol is said to provide *key confirmation* if one party is assured that a specially identified second party actually has possession of a particular session key. If the implicit key authentication or key confirmation is provided to

*The first author is a consultant to MÖBIUS Encryption Technologies.

†The third author holds the MÖBIUS Chair of Cryptography at St. Jerome's College, University of Waterloo.

both parties involved in the protocol, then the authentication is said to be *mutual*; if provided to only one party, the authentication is *unilateral*. The number of *passes* in a protocol is the number of messages exchanged between the two parties. Broadly speaking, there are two kinds of key establishment protocols: *key transfer* protocols in which a key is created by one party and securely transmitted to the second party, and *key agreement* protocols in which both parties contribute information which jointly establish the shared secret key.

In this paper, we shall only consider key agreement protocols for the asymmetric (public-key) two-party setting. For further discussions on the properties of key agreement protocols, the reader is referred to the survey article by Rueppel and van Oorschot [9] and Chapter 12 of [6].

There are various schemes in the literature which claim to provide implicit key authentication. The purpose of this paper is fourfold:

- To describe new attacks on some implicit key agreement protocols. The first kind of attack is general and applies to many systems. In all the attacks an active adversary E modifies messages exchanged between parties A and B, the result being that B believes he shares a key K with E while A believes she shares the same key K with B; E does not learn the value of K.

  A practical scenario where the attack may be launched successfully is the following (this attack was first described by Diffie, van Oorschot and Wiener [3]). Suppose that B is a bank branch and A is an account holder. Certificates are issued by the bank headquarters and within the certificate is the account information of the holder. Suppose that the protocol for electronic deposit of funds is to exchange a key with a bank branch via a mutually authenticated key agreement. Once B has authenticated the transmitting entity, encrypted funds are deposited to the account number in the certificate. Suppose that no further authentication is done in the encrypted deposit message (which might be the case to save bandwidth). If the attack mentioned above is successfully launched then the deposit will be made to E's account instead of A's account.

- To present modifications of some of these protocols in order to achieve the properties that were desired.

- To present two new two-pass key agreement protocols (Protocol 1 and Protocol 2) which provide mutual implicit key authentication, have low communication and computation overhead, and are *non-interactive* (the message transmitted between the two parties are independent of each other). Other features of the new protocols are that they are *role-symmetric* (the 2 messages transmitted between parties have the same structure), and do not require encryption, hash functions, or timestamping.

- To present a new one-pass key agreement protocol (Protocol 3) which provides mutual implicit key authentication, has low communication and computation overhead, and does not require encryption, hash functions, or timestamping.

The remainder of the paper is organized as follows. Section 2 describes four variants of the Matsumoto-Takashima-Imai (MTI) key agreement protocols, the new attacks on them, and modifications which resist the attack. Section 3 describes the Nyberg-Rueppel one-pass key agreement protocol, and the new attack on it. The new two-pass key agreement protocols are presented in Section 4, while the new one-pass key agreement protocol is presented in Section 5. Section 6 is a general note about a particular problem of replay in one-pass key establishment protocols. Finally, Section 7 makes some concluding remarks.

2 The MTI key agreement protocols

The three protocols described in this section are special cases of the 3 infinite families of key agreement protocols invented by Matsumoto, Takashima and Imai [5]. In particular, protocols MTI/A0, MTI/B0 and MTI/C0 correspond to protocols $A(0)$, $B(0)$, and $C(0)$, respectively, of the original paper [5]. The MTI protocols are variants of the Diffie-Hellman key exchange [2] whose purpose is for parties A and B to establish a secret session key K.

The system parameters for these protocols are a prime number p and a generator α of the multiplicative group $\mathbb{Z}_p^*$; these parameters are fixed and known to all users. Party A has private key a and public key $p_A = \alpha^a \bmod p$. Party B has private key b and public key $p_B = \alpha^b \bmod p$. (In order to simplify the notation, the modulus p will be omitted for the rest of the paper. Also, it is understood that if any protocol or attack uses the quantity $a^{-1} \bmod (p-1)$, $b^{-1} \bmod (p-1)$ or $e^{-1} \bmod (p-1)$, then a, b or e were chosen subject to the additional constraint of being relatively prime to $p-1$.) In all three protocols below, text_A refers to a string of information that identifies party A. If the other party B a priori possesses an authentic copy of A's public key, then text_A merely consists of A's identity, such as her name. Otherwise, text_A will contain A's public-key certificate, issued by a trusted center; B can use his authentic copy of the trusted center's public key to verify A's certificate, hence obtaining an authentic copy of A's public key.

We describe a new attack on the MTI protocols which demonstrate that they do not provide implicit key authentication. In each attack, a third party E wishes to have messages sent from A to B identified as having originated from E herself. To accomplish this, E selects a random integer e, $1 \leq e \leq p-2$, computes $p_E = (p_A)^e = \alpha^{ae}$, and gets this certified as her public key. Notice that E does not know the exponent ae (assuming, of course, that the discrete logarithm problem in $\mathbb{Z}_p^*$ is intractable), although she knows e.

We then present modifications to each of the 3 protocols which foil this new attack thereby achieving the desired property of mutual implicit authentication. In the modified protocols $F(X, Y)$ denotes a cryptographic hash function, such as the NIST Secure Hash Algorithm [7], applied to the string obtained by concatenating X and Y.

2.1 MTI/A0 protocol

The protocol

1. A generates a random integer x, $1 \leq x \leq p-2$, computes α^x, and sends $\{\alpha^x, \text{text}_A\}$ to party B.

2. B generates a random integer y, $1 \leq y \leq p-2$, computes α^y, and sends $\{\alpha^y, \text{text}_B\}$ to party A.

3. A computes $K = (\alpha^y)^a (p_B)^x = \alpha^{ay+bx}$.

4. B computes $K = (\alpha^x)^b (p_A)^y = \alpha^{ay+bx}$.

The new attack

1. E intercepts A's message $\{\alpha^x, \text{text}_A\}$ and replaces it with $\{\alpha^x, \text{text}_E\}$.

2. B sends $\{\alpha^y, \text{text}_B\}$ to E, who then forwards $\{(\alpha^y)^e, \text{text}_B\}$ to A.

3. A computes $K = (\alpha^{ey})^a (p_B)^x = \alpha^{aey+bx}$.

4. B computes $K = (\alpha^x)^b (p_E)^y = \alpha^{aey+bx}$.

A and B now share the key K even though B believes he shares a key with E; E does not learn the value of K.

The modified protocol

1. A generates a random integer x, $1 \leq x \leq p-2$, computes α^x, and sends $\{\alpha^x, \text{text}_A\}$ to party B.

2. B generates a random integer y, $1 \leq y \leq p-2$, and computes α^y, $K = (\alpha^x)^b (p_A)^y = \alpha^{ay+bx}$, and $h = F(\alpha^y, \alpha^{ay+bx})$. B sends $\{\alpha^y, h, \text{text}_B\}$ to party A.

3. A computes $K = (\alpha^y)^a (p_B)^x = \alpha^{ay+bx}$. A also computes $h' = F(\alpha^y, K)$ and verifies that this quantity is equal to h. (If $h \neq h'$ then the protocol terminates with failure.)

2.2 MTI/B0 protocol

The protocol

1. A generates a random integer x, $1 \leq x \leq p-2$, computes $(p_B)^x = \alpha^{bx}$, and sends $\{\alpha^{bx}, \text{text}_A\}$ to party B.

2. B generates a random integer y, $1 \leq y \leq p-2$, computes $(p_A)^y = \alpha^{ay}$, and sends $\{\alpha^{ay}, \text{text}_B\}$ to party A.

3. A computes $K = (\alpha^{ay})^{a^{-1}} \alpha^x = \alpha^{x+y}$.

4. B computes $K = (\alpha^{bx})^{b^{-1}} \alpha^y = \alpha^{x+y}$.

The new attack

1. E replaces A's message $\{\alpha^{bx}, \text{text}_A\}$ with $\{\alpha^{bx}, \text{text}_E\}$.

2. B sends $\{(p_E)^y, \text{text}_B\}$ to E, who then computes $((p_E)^y)^{e^{-1}} = \alpha^{ay}$ and forwards $\{\alpha^{ay}, \text{text}_B\}$ to A.

3. A computes $K = (\alpha^{ay})^{a^{-1}} \alpha^x = \alpha^{x+y}$.

4. B computes $K = (\alpha^{bx})^{b^{-1}} \alpha^y = \alpha^{x+y}$.

A and B now share the key K even though B believes he shares a key with E; E does not learn the value of K.

The modified protocol

1. A generates a random integer x, $1 \leq x \leq p-2$, computes $(p_B)^x = \alpha^{bx}$, and sends $\{\alpha^{bx}, \text{text}_A\}$ to party B.

2. B generates a random integer y, $1 \leq y \leq p-2$, and computes $(p_A)^y = \alpha^{ay}$, $K = (\alpha^{bx})^{b^{-1}} \alpha^y = \alpha^{x+y}$, and $h = F(\alpha^{ay}, \alpha^{x+y})$. B sends $\{\alpha^{ay}, h, \text{text}_B\}$ to A.

3. A computes $K = (\alpha^{ay})^{a^{-1}} \alpha^x = \alpha^{x+y}$. A also computes $h' = F(\alpha^{ay}, K)$ and verifies that this quantity is equal to h. (If $h \neq h'$ then the protocol terminates with failure.)

2.3 MTI/C0 protocol

The protocol

1. A generates a random integer x, $1 \leq x \leq p-2$, computes $(p_B)^x = \alpha^{bx}$, and sends $\{\alpha^{bx}, \text{text}_A\}$ to party B.

2. B generates a random integer y, $1 \leq y \leq p-2$, computes $(p_A)^y = \alpha^{ay}$, and sends $\{\alpha^{ay}, \text{text}_B\}$ to party A.

3. A computes $K = (\alpha^{ay})^{a^{-1}x} = \alpha^{xy}$.

4. B computes $K = (\alpha^{bx})^{b^{-1}y} = \alpha^{xy}$.

The new attack

1. E replaces A's message $\{\alpha^{bx}, \text{text}_A\}$ with $\{\alpha^{bx}, \text{text}_E\}$.

2. B sends $\{(p_E)^y, \text{text}_B\}$ to E, who then computes $((p_E)^y)^{e^{-1}} = \alpha^{ay}$ and forwards $\{\alpha^{ay}, \text{text}_B\}$ to A.

3. A computes $K = (\alpha^{ay})^{a^{-1}x} = \alpha^{xy}$.

4. B computes $K = (\alpha^{bx})^{b^{-1}y} = \alpha^{xy}$.

A and B now share the key K even though B believes he shares a key with E; E does not learn the value of K.

The modified protocol

1. A generates a random integer x, $1 \leq x \leq p-2$, computes $(p_B)^x = \alpha^{bx}$, and sends $\{\alpha^{bx}, \text{text}_A\}$ to party B.
2. B generates a random integer y, $1 \leq y \leq p-2$, and computes $(p_A)^y = \alpha^{ay}$, $K = (\alpha^{bx})^{b^{-1}y} = \alpha^{xy}$, and $h = F(\alpha^{ay}, \alpha^{xy})$. B sends $\{\alpha^{ay}, h, \text{text}_B\}$ to party A.
3. A computes $K = (\alpha^{ay})^{a^{-1}x} = \alpha^{xy}$. A also computes $h' = F(\alpha^{ay}, K)$ and verifies that this quantity is equal to h. (If $h \neq h'$ then the protocol terminates with failure.)

2.4 Remarks

In each of the three attacks, E does not learn the value of the session key K. The new attack thus fails on the modified protocols because in each case B sends the hash value $F(R, K)$, where R is B's random exponential (α^y, α^{ay}, α^{ay}, in protocols MTI/A0, MTI/B0 and MTI/C0 respectively), thereby binding together the values R and K. E cannot subsequently replace the value R with R^e and compute $F(R^e, K)$ since E does not know K.

Instead of F being a cryptographic hash function, an option available is to choose $F = E_K$, where E is the encryption function of a suitable symmetric-key encryption scheme, and K is the session key established. Another way to foil the new attack is to require that each entity prove to the trusted center that it knows the exponent a corresponding to its public key α^a, before the center issues a certificate. This can be achieved through zero knowledge techniques (for example, see [1]).

We comment that the new attack also compromises each of the protocols in the 3 infinite classes of MTI protocols; the attack can be thwarted in the same manner as was done here for the 3 specific MTI protocols presented. The Goss authenticated key exchange protocol [4] is similar to the MTI/A0 protocol, except that the session key is the bitwise exclusive-or of α^{ay} and α^{bx}; that is $K = \alpha^{ay} \oplus \alpha^{bx}$ instead of being the product of α^{ay} and α^{bx}. Hence the attack on the MTI/A0 protocol and its modification can be extended in a straightforward manner to the case of the Goss protocol. Also, Yacobi's authenticated key exchange protocol [10] is exactly the same as the MTI/A0 protocol, except that α is an element of the group of units $\mathbb{Z}_n^*$, where n is the product of 2 large primes. Again, the attack on the MTI/A0 protocol and its modification can be extended in a straightforward manner to the case of the Goss protocol.

Lastly, we note that the modified protocols offer key confirmation from B to A.

3 The Nyberg-Rueppel one-pass key agreement protocol

3.1 The protocol

The purpose of this protocol is for parties A and B to establish a session key K by only having to transmit one message from A to B. We describe a particular variant of the more general protocol presented in [8].

The system parameters for these protocols are a prime number p and a generator α of the multiplicative group $\mathbb{Z}_p^*$. User A has private key a, $1 \leq a \leq p-2$, and public key $p_A = \alpha^a$. User B has private key b, $1 \leq b \leq p-2$, and public key $p_B = \alpha^b$. The protocol assumes that A a priori has an authentic copy of B's public key; otherwise it is a two-pass protocol.

1. A selects random integers k and t, $1 \leq k, t \leq p-2$.
2. A computes $K = \alpha^t$, $r = (p_B)^t \alpha^{-k}$ and $s = k - ra \bmod (p-1)$, and sends $\{r, s, \text{text}_A\}$ to B.
3. B recovers the value α^k by computing $\alpha^s (p_A)^r$ and then computes the shared secret $K = (r\alpha^k)^{b^{-1}} = \alpha^t$.

3.2 The new attack

We describe a new attack on the Nyberg-Rueppel protocol which demonstrates that it does not provide implicit key authentication. In the attack, a third party E wishes to have messages from A identified as having originated from herself. To accomplish this, E selects a random integer e, $1 \leq e \leq p-2$, computes $p_E = \alpha^e$, and gets this certified as her public key.

1. E intercepts A's message $\{r, s, \text{text}_A\}$ and computes $\alpha^k = \alpha^s (p_A)^r$ and $\alpha^{bt} = r\alpha^k$.
2. E then selects a random integer k', $1 \leq k' \leq p-2$, computes $r' = \alpha^{bt}\alpha^{-k'}$ and $s' = k' - r'e \bmod (p-1)$.
3. E sends $\{r', s', \text{text}_E\}$ to B.
4. B recovers the value $\alpha^{k'}$ by computing $\alpha^{s'} (p_E)^{r'}$ and then computes $K = (r'\alpha^{k'})^{b^{-1}} = \alpha^t$.

A and B now share the key K, even though B believes he shares a key with E; E does not learn the value of K.

3.3 The modified protocol

One way to foil the attack is to modify the protocol by requiring A to also transmit $h = F(p_A, K)$, where F is either a cryptographic hash function or an encryption function of a symmetric-key system with key K. The modified protocol is the following.

1. A selects random integers k and t, $1 \leq k, t \leq p-2$.

2. A computes $K = \alpha^t$, $r = (p_B)^t \alpha^{-k}$, $s = k - ra \bmod (p-1)$, and $h = F(p_A, K)$. A sends $\{r, s, h, \text{text}_A\}$ to B.

3. B recovers the value α^k by computing $\alpha^s (p_A)^r$ and then computes the shared secret $K = (r\alpha^k)^{b^{-1}} = \alpha^t$. B also computes $h' = F(p_A, K)$ and verifies that this quantity is equal to h. (If $h \neq h'$ then the protocol terminates with failure.)

4 The new two-pass key agreement protocols

The purpose of the two-pass protocol presented in this section is for parties A and B to establish a session key K. The protocols differ from the modified key agreement protocols of Section 2 in that they are role-symmetric, non-interactive, and do not require hash functions or encryption.

The system parameters for this protocol are a prime number p and a generator α of the multiplicative group $\mathbb{Z}_p^*$. User A has private key a and public key $p_A = \alpha^a$. User B has private key b and public key $p_B = \alpha^b$.

4.1 Protocol 1

1. A picks a random integer x, $1 \leq x \leq p-2$, and computes $r_A = \alpha^x$ and $s_A = x - r_A a \bmod (p-1)$. A sends $\{r_A, s_A, \text{text}_A\}$ to B.

2. B picks a random integer y, $1 \leq y \leq p-2$, and computes $r_B = \alpha^y$ and $s_B = y - r_B b \bmod (p-1)$. B sends $\{r_B, s_B, \text{text}_B\}$ to A.

3. A computes $\alpha^{s_B} (p_B)^{r_B}$ and verifies that this is equal to r_B. A computes $K = (r_B)^x = \alpha^{xy}$.

4. B computes $\alpha^{s_A} (p_A)^{r_A}$ and verifies that this is equal to r_A. B computes $K = (r_A)^y = \alpha^{xy}$.

Discussion

Protocol 1 is essentially a Diffie-Hellman key exchange with s_A serving as a signature for A's exponential, and s_B serving as a signature for B's exponential. One drawback of Protocol 1 is that it does not offer *perfect forward secrecy* [3, page 113]. That is, if an adversary learns the long-term private key a of party A, then the adversary can deduce all of A's past session keys. The property of perfect forward secrecy can be achieved by modifying Protocol 1 in the following way. In step 1, A also sends α^{x_1} to B, where x_1 is a random integer. Similarly, in step 2, B also sends α^{y_1} to A, where y_1 is a random integer. A and B now compute the key $K = \alpha^{xy} \oplus \alpha^{x_1 y_1}$.

Another drawback of Protocol 1 is that if an adversary learns the private information x of A, then the adversary can deduce the long-term private key a of party A from the equation $s_A = x - r_A a \bmod (p-1)$. This drawback can easily be overcome in practice since a well designed implementation of the protocol should prevent this private information from being disclosed.

The next protocol addresses the two drawbacks of Protocol 1.

4.2 Protocol 2

1. A picks a random integer x, $1 \le x \le p-2$, and computes $(p_B)^x$, α^x and $s_A = x + a(p_B)^x \bmod (p-1)$. A sends $\{\alpha^x, s_A, \text{text}_A\}$ to B.

2. B picks a random integer y, $1 \le y \le p-2$, and computes $(p_A)^y$, α^y and $s_B = y + b(p_A)^y \bmod (p-1)$. B sends $\{\alpha^y, s_B, \text{text}_B\}$ to A.

3. A computes $(\alpha^y)^a$ and verifies that $\alpha^{s_B}(p_B)^{-\alpha^{ay}} = \alpha^y$. A then computes $K = \alpha^{ay}(p_B)^x$.

4. B computes $(\alpha^x)^b$ and verifies that $\alpha^{s_A}(p_A)^{-\alpha^{bx}} = \alpha^x$. B then computes $K = \alpha^{bx}(p_A)^y$.

Discussion

Protocol 2 may be viewed as a modification of the MTI/A0 protocol (the key K established is the same in both protocols) where s_A serves as A's signature on her exponential α^x, and s_B serves as B's signature on his exponential α^y. The signatures have the property that s_A can only be verified by B, while s_B can only be verified by A.

Protocol 2 improves upon Protocol 1 in the sense that it offers perfect forward secrecy. While it is still the case that disclosure of private information x allows an adversary to learn the private key a, this will not be a problem in practice because A can destroy x as soon as she uses it in step 1 of the protocol. By contrast, in Protocol 1, having picked x in step 1, A has to securely store x until she uses it in step 3.

If A does not have an authenticated copy of B's public key then B has to transmit a certified copy of his key to B at the beginning of the protocol. In this case, Protocol 2 is a three-pass protocol.

5 The new one-pass key agreement protocol

The purpose of this protocol is for parties A and B to establish a session key K by only having to transmit one message from A to B. The protocol assumes that A has a priori an authentic copy of B's public key; otherwise it is a two-pass protocol.

5.1 Protocol 3

1. A picks a random integer x, $1 \le x \le p-2$, and computes $r_A = \alpha^x$ and $s_A = x - r_A a \bmod (p-1)$. A compute $K = (p_B)^x$ and sends $\{r_A, s_A, \text{text}_A\}$ to B.

2. B computes $\alpha^{s_A}(p_A)^{r_A}$ and verifies that this quantity is equal to r_A. B computes $K = (r_A)^b$.

Protocol 3 can be viewed as an extension of the ElGamal variant of the Diffie-Hellman key exchange (see [9]), with s_A serving as A's signature on her exponential α^x.

Protocol 3 has the same drawbacks as Protocol 1. It's advantage over the Nyberg-Rueppel one-pass protocol is that it does not require a hash function or encryption.

6 A note on one-pass key establishment protocols

All one-pass key establishment protocols (i.e., both key agreement and key transport protocols) have the following problem of replay. Suppose that a one-pass key establishment protocol is used to transmit a session key K from A to B as well as some text encrypted with the session key K. Suppose that E records the transmission from A to B. If E can at a later time gain access to B's decryption machine (but not the internal contents of the machine, such as B's private key), then, by replaying the transmission to the machine, E can recover the original text. (In this scenario, E does not learn the session key K.)

This replay attack can be foiled by usual methods, such as the use of timestamps. There are, however, some practical situations when B has limited computational resources, in which case the following modification may be more suitable.

At the beginning of each session, B transmits a random bit string k in the clear to A. The session key that is used to encrypt the text is then $k \oplus K$. Since a different string k is selected before each key agreement, the replay attack described above will fail.

7 Concluding remarks

All protocols discussed in this paper have been described in the setting of the multiplicative group $\mathbb{Z}_p^*$. However, they can all be easily modified to work in any finite group in which the discrete logarithm problem appears intractable. Suitable choices include the multiplicative group of a finite field (in particular the finite field $GF(2^n)$), subgroups of $\mathbb{Z}_n^*$ where n is a composite integer, subgroups of $\mathbb{Z}_p^*$ of order q, and the group of points on an elliptic curve defined over a finite field.

All protocols discussed in this paper can also be modified in a straightforward way to handle the situation when each user picks their own system parameters p and α (or analogous parameters if a group other than $\mathbb{Z}_p^*$ is used).

Acknowledgements

The authors would like to thank Paul van Oorschot for many helpful comments and for providing several pointers to the literature.

References

[1] D. Chaum, J.-H. Evertse and J. van de Graaf, "An improved protocol for demonstrating possession of discrete logarithms and some generalizations", Advances in Cryptology – Eurocrypt '87, Proceedings, Lecture Notes in Computer Science, **304**, Springer-Verlag, 1988, 127-141.

[2] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, **22** (1976), 644-654.

[3] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, **2** (1992), 107-125.

[4] K.C. Goss, "Cryptographic method and apparatus for public key exchange with authentication", U.S. patent 4,956,865, September 11, 1990.

[5] T. Matsumoto, Y. Takashima and H. Imai, "On seeking smart public-key distribution systems", *The Transactions of the IECE of Japan*, **E69** (1986), 99-106.

[6] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, to appear.

[7] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS Publication 180, May 1993.

[8] K. Nyberg and R. Rueppel, "Message recovery for signature schemes based on the discrete logarithm problem", *Designs, Codes and Cryptography*, to appear.

[9] R. Rueppel and P. van Oorschot, "Modern key agreement techniques", *Computer Communications*, **17** (1994), 458-465.

[10] Y. Yacobi, "A key distribution paradox", Advances in Cryptology – Crypto '90, Proceedings, Lecture Notes in Computer Science, **537**, Springer-Verlag, 1991, 268-273.