Computers & Security Vol. 17, No.3, pp. 253-264, 1998
© 1998 Elsevier Science Limited
All rights reserved. Printed in Great Britain
0167-4048/98 $19.00

Realisation of a minimum-knowledge identification and signature scheme

T. Coffey and T. Newe

Department of Electronic & Computer Engineering University of Limerick Ireland

A minimum-knowledge scheme allows a claimant to prove its identity to a verifier without disclosing any secret information. Minimum-knowledge schemes, incorporating identity verification, signature generation and verification, are generally based on interactive proofs. The Ohta-Okamoto minimum-knowledge identification and signature scheme is characterised by a good balance between the claimant's storage requirements and the time to perform a verification. This makes it particularly suitable for use with minimum storage devices such as smart cards.

This paper presents a realisation of an Ohta-Okamoto based minimum knowledge and signature scheme, ranging over identity verification, signature generation and verification. The modular arithmetic functions, such as: multiplication, division, exponentiation and multiplicative inverse, as well as prime number generation, pseudo random number generation and hashing function are detailed. An analysis of the realised scheme is presented, including a comparison with the Fiat-Shamir identification scheme.

Keywords: minimum-knowledge, Ohta-Okamoto, Fiat-Shamir, identity verification, digital signatures, multiple precision, modular arithmetic, prime numbers, random numbers, hashing algorithm, smart cards.

Introduction

Minimum-knowledge identification schemes provide a means of gaining confidence in a prover or claimant. Generally these schemes involve a series of random challenges made by a verifier to a claimant. The claimant computes a response to each challenge using its private key. By examining the claimant's response, the verifier is able to establish, with a high level of confidence, that the claimant is indeed in possession of the necessary secret information, although no secret information is exchanged during this sequence of events.

Identification schemes are only useful against external threats when both the claimant and verifier co-operate [1]. It is possible that a verifier could forge a credible transcript of an imaginary communication with a claimant, by carefully choosing both the questions and answers in a dialogue. With signature schemes, however, only real communication with a claimant could generate a credible transcript, as the signature for a message is generated entirely by the claimant and not by dialogue with the verifier.

A number of minimum-knowledge identification/signature schemes currently exist, such as: *Fiat-Shamir[1], Ohta-Okamoto[2], Guillou-Quisquater[3], Schnorr[4], Micali-Shamir[5], Beth[6], and Stern[7].* These schemes vary in terms of the number of challenge-response cycles, the complexity of the modular calculations, and the amount of secret information that must be stored. The Ohta-Okamoto scheme, although relatively slow in comparison with the Fiat-Shamir scheme, is particularly suitable for use with minimum storage devices such as smart cards, since it minimises the claimant's storage requirements and the amount of data exchanged during a verification.

2. The Ohta-Okamoto Scheme

2.1 Introduction

The Ohta-Okamoto identification scheme is based on the difficulty of extracting the L^{th} roots *mod n*, where the factors of n are unknown. If L is large ($\geq$ 30 bits) and $n = p * q$: where p and q are private large prime numbers ($\geq$ 256 bits), then extracting modular roots is as difficult as factoring the public modulus n, to obtain p and q [8]. The public modulus n ($\geq$ 512 bits) is considered to be beyond the capabilities of current factoring algorithms [9,10,11,12]. However, as factoring algorithms improve it may be necessary to increase the size of n up to say a 1024 bit number.

The Ohta-Okamoto scheme requires the existence of a trusted centre, such as a government, university, bank, etc., to generate and publish a public modulus n and a public integer L. Parameters n and L are common to all principles in the domain of the trusted centre. The claimant generates and publishes its public key I, where $I^{-1} = S^{L}$ $(mod\ n)$, and S is the claimant's secret random integer, $S \in Z_n$ where Z_n denotes $\{0 \ldots n\text{-}1\}$.

2.2 Identity Verification

Verification of a claimant's identity by a verifier in the Ohta-Okamoto scheme, is achieved by executing the following four steps t times. If the value of L is sufficiently large ($\geq$ 30 bits) then $t = 1$.

1. The claimant generates a random number $R \in Z_n$, and sends $X = R^L$ (mod n) to the verifier.

2. The verifier responds with a random number $E \in Z_L$.

3. The claimant sends $Y = R * S^E$ (mod n) to the verifier.

4. The verifier checks $Y^L * I^E \equiv X$ (mod n).

The verifier only accepts the claimant's proof of identity if step 4 holds true.

2.3 Signature Scheme

In the Ohta-Okamoto scheme a digitally signed message contains a triplet M, E, Y, where M is a message, $E = f(M, X) \in Z_L$, f is a public one-way hash function, and $Y = R * S^E$ (mod n). Signing a message requires the availability of public integer L, a random number R ($1 < R < n$), and the claimant's secret S.

2.4 Security of Ohta-Okamoto Scheme

The security level metric, generally represented as 2^{-x}, refers to the probability of success of forging an identity or a signature. For example, in a system with a security level of 2^{-30}, a forger has a probability of success of $0.93 * 10^{-9}$, or 1 chance in every 10^9 identification procedural attempts. In the Ohta-Okamoto scheme the security level is represented as $L^{-(t*k)}$, where t is the number of cycles required and k is the number of secrets which must be stored. If L is chosen to be large enough ($\geq 2^{30}$) then the value of the parameter $t = k = 1$, is applicable.

3. Realising an Ohta-Okamoto based minimum-knowledge scheme

3.1 Role of Central Authority

The proposed identity verification and signature scheme is designed to be used in applications involving multiple claimants and verifiers. To facilitate the implementation of the scheme in this environment, we have designated the trusted central authority as being responsible for generating the following information: (i) n - the public system-wide modulus ($n \geq$ 512 bits), (ii) L - the public system-wide integer ($L \geq$ 30 bits), (iii) S - the claimant's secret key ($S \in Zn$), (iv) I_c - the claimant's public key, wher $I_c^{-1} = S^L$ (mod n) and (v) a new parameter c ($\geq$ 16 bits) which acts as the claimant's identification number. The verifier uses c to retrieve the claimant's public key I_c.

Each claimant's smart card (or host) is personalised using parameters n, L, S. Each verifier has a copy of parameters n and L as well as access to each claimant's public key I_c, either from a key database located in the

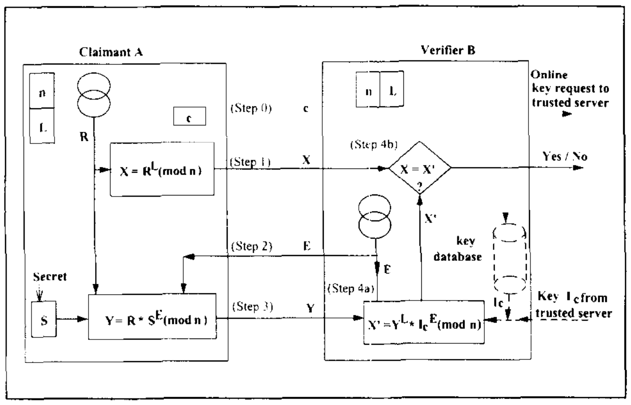

Fig.1: Claimant Identification Procedure

verifier (for off-line operation), or from a trusted on-line server (see fig. 1).

3.2 Claimant Identification procedure

The identification procedure of a claimant by a verifier is outlined in fig. 1. In step 0 of the procedure, the claimant transmits its identification number c to the verifier. The claimant's public key I_c is retrieved from a database using parameter c. The public key may be retrieved off-line using a local database or on-line using a trusted server. Steps 1-4 of the identification procedure are the same as described in section 2.2 above.

On verification of $Y^L \star I_C^E \equiv X \pmod{n}$ in step 4 (a & b), the verifier establishes that the claimant possesses secret S, since :

$$Y = R \star S^E \pmod{n}$$

$$Y^L = R^L \star S^{LE} \pmod{n}$$

$$= X \star S^{LE} \pmod{n} \{ \text{ since } X = R^L \pmod{n} \}$$

$$= X \star I_c^{-E} \pmod{n} \{ \text{ since } I_c^{-1} = S^L \pmod{n} \}$$

$$Y^L \star I_c^E = X \pmod{n}$$

3.3 Signature Scheme

Fig. 2 details the operation of signing a message, using the Ohta-Okamoto signature scheme. The triplet (M, E, Y) is computed and sent as the signed message to the verifier. Fig. 3 outlines a signature verification procedure for such a signed message. The procedure requires the triplet: message M, hash value E, and Y, where $Y = R \star S^E \pmod{n}$. The verification ascertains that $f(M, Y^L \star I_c^E \pmod{n})) \equiv f(M,X)$, as follows:

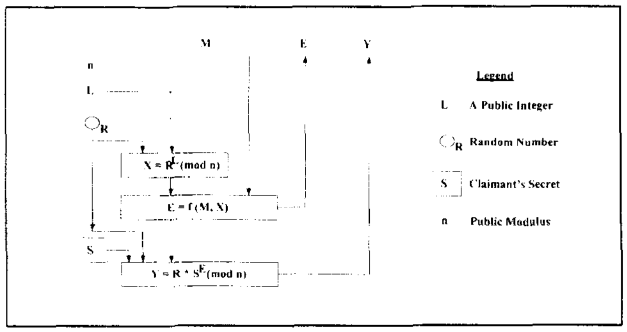

Fig. 2: Signing a Message

$$f(M, Y^L \star I_c^E \pmod{n})) = f(M, R^L \star S^{LE} \star I_c^E \pmod{n})) \quad \{\text{Since } Y = R \star S^E \pmod{n}\}$$

$$= f(M, R^L \star I_c^{-E} \star I_c^E \pmod{n})) \quad \{\text{Since } I_c^{-1} = S^L \pmod{n} \}$$

$$= f(M, R^L \pmod{n}))$$

$$= f(M, X)$$

$$= E'$$

Thus, if E = E' the signed message has been verified.

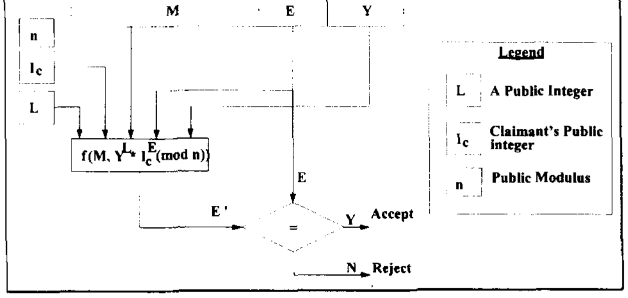

Fig. 3: Digital Signature Verification

4. Multiple Precision Algorithms

In this section the multiple precision (MP) functions required to implement the minimum-knowledge scheme presented in section 3 are discussed and pseudo code algorithms are given. These MP functions form the core of the minimum-knowledge scheme and dictate the scheme's operating efficiency. The algorithms described include: modular multiplication, modular exponentiation, modular inverse, prime number generation, random number generation, and a hashing algorithm.

4.1 Multiplication of MP numbers

In a high level language such as C or Pascal, MP numbers are represented using arrays of integers. Multiplication of MP numbers can be performed using a series of multiplications and additions of array locations.

Consider the following $a = b \star c$, where b and c have the values 70248_{10} and 131363_{10} respectively.

Array a contains a maximum number of locations equal to the sum of locations in b and c. The multiplication takes place as follows:

a[0] = $4712 \star 291 = 60472_{10}$ with a carry into a[1] of 20_{10}

a[1] = $(1 \star 291) + 20 + (4712 \star 2) = 9735_{10}$ with no carry

a[2] = $1 \star 2 = 2_{10}$ with no carry.

a[3] = $0 = 0_{10}$

Fig. 4 presents a pseudo code algorithm to perform $a = b \star c$, using arrays of long integers.

4.2 Division of MP numbers

The division of MP numbers can be handled simply by an estimate and correction algorithm.

Fig. 5 describes an algorithm which estimates each of the quotient digits, by dividing the high-order digit of what remains of the dividend, by the high order digit of the divisor plus 1. The estimate is corrected if what remains after subtracting a shifted multiple of the divisor is too large, and the process repeats until a number which is smaller than the divisor remains.

Fig. 6 contains an MP division example where the dividend and the divisor are 16278049 and 3672 respectively. The remainder of the division (i.e. 73) represents the result of a MOD operation and the Quotient (i.e. 4433) represents the result of a DIV operation.

4.3 Modular Multiplication of MP numbers

Modular multiplication, which is represented by: $a=b\star c$ (*mod d*), can be effected by combining the MP division and multiplication algorithms. This operation is performed by first multiplying b by c, dividing by d, and the remainder is then assigned to a.

	b	c	$a = b \star c$
Decimal	70248_{10}	131363_{10}	9227989024_{10}
Hexadecimal	11268_{16}	20123_{16}	$22607EC38_{16}$
$[0]_{16}, [1]_{16} \cdots [n]_{16}$	1268, 1	0123, 2	EC38, 2607, 2, 0
$[0]_{10}, [1]_{10} \cdots [n]_{10}$	4712, 1	291, 2	60472, 9735, 2, 0

Table 1: Representation of MP numbers

```
bLocations = number of array locations in b.
cLocations = number of array locations in c.
a = long int array with (bLocations+cLocations) locations.
t = long int array with 2 locations.
carry = long int = 0.
for (i = 0;  i < bLocations; i = i + 1)
{       for (j = 0; j < cLocations; j = j + 1)
        {       t = b[i] * c[j];
                if ( a[i+j] = a[i+j] + carry ) < carry
                        then carry = 1;
                        else  carry = 0;
                if ( a[i+j] = a[i+j] + t[0] ) < t[0]
                        then carry = carry + 1
                carry = carry + t[1];
        }
        a[i + cLocations] = a[i + cLocations] + carry;
}
```

Fig. 4: Multiplication of MP numbers

```
cLocations = number of array locations in c (dividend) and a.
dLocations = number of array locations in d (divisor) and b.
t , Quo_est = long int variable.
Normalise Divisor and Dividend.
t = d[dlocations - 1] + 1.
for (i = (cLocations - dlocations); i >= 0; i = i - 1)
{       if (t < MAX_DIGIT)                   // Estimate Quotient digit.
                then Quo_est = c[i] / t;     // Normal division of long integers.
                else Quo_est  = c[i];
        while (estimate too small)           // Test estimate.
                Quo_est = Quo_est + 1;       // Correct estimate.
        a[i] = Quo_est;                      // Result of DIV operation.
}
b = c - (d * a);                             // Get remainder for result of MOD operation.
```

Fig. 5: Division of MP numbers

```
          4433        ⇐   Quotient
3672  √16278049
         0000         ⇐   estimate: 1/4    =0
         16278049
         14688        ⇐   estimate: 16/4          =4
          1590049         (correct)
          11016       ⇐   estimate: 15/4   =3
           488449         (too large)
            3672      ⇐   corrected estimate:     =4
           121249
           11016      ⇐   estimate: 12/4          =3
            11089         (correct)
             7344     ⇐   estimate: 11/4   =2
             3745         (too large)
             3672     ⇐   corrected estimate:     =3
               73     ⇐   remainder
```

Fig. 6: Division using the estimate and correct method.

4.4 Modular exponentiation of MP number

Fig. 7 presents a fast exponentiation algorithm for computing $b^c (mod\ d)$, using the modular multiplication operation described in section 4.3. The algorithm shifts the exponent left by two bits, then checks the two most significant bits. These operations are repeated until the exponent is equal to zero. This enables a pre-calculated number to be used for b, b^2, or b^3 (mod d). This pre-calculation enables the algorithm to operate at greater speed when used with large numbers. The speed of operation is approximately doubled, compared to algorithms which operate on single bits.

The modular calculations ensure that all interim results (and the final result) are restricted to the size of the modulus.

4.5 Modular inverse of MP numbers

A number $a \in \{0, n\text{-}1\}$, may have a unique inverse $x \in \{0, n\text{-}1\}$ such that ax (mod n) $\equiv 1$. In general $a^{-1} = x$ (mod n) has a unique solution if a and n are relatively prime. If a and n are not relatively prime, then $a^{-1} = x$ (mod n) has no solution. If n is a prime number, then every number from 1 to n - 1 is relatively prime to n and has exactly one inverse modulo n in that range [13].

Fig. 8 presents the extended Euclidean algorithm to compute the inverse, x, of a number a modulo n. This computes x, where ax (mod n) = 1.

4.6 Prime number generation

The security of the Ohta-Okamoto scheme depends on using carefully selected primes p and q. If the public modulus n is 512 bits ($n = p * q$), then p and q should be large primes of approximately 256 bits each. A number p (p = 1,2,3,4...) is prime if its only divisors are ±1 and ±p, otherwise it is composite.

The simplest method for testing to see if a number is prime is to divide it by all the prime numbers less than the square-root of the number [13]. However, this approach is not feasible because of the number of computations required due to the size of numbers

```
First calculate $b^2$ mod $d$, and $b^3$ mod $d$ using the modular multiplication algorithm.
cLocations = number of locations in c array
for (i = cLocations - 1; i >= 0; i = i + 1)
{       for (j = 0; j < (number of bits in c[i] - number of leading zeroes); j = j + 2)
        {       if (2 MSBs of c[i] = 1)
                        then compute a = a * b (mod d);
                if (2 MSBs of c[i] = 2)
                        then compute a = a * $b^2$ (mod d);
                if (2 MSBs of c[i] = 3)
                        then compute a = a * $b^3$ (mod d);
                remove 2 MSBs of c[i] so that the next 2 bits are now the MSBs;
        }
}
```

Fig. 7: Modular Exponentiation

used by the Ohta-Okamoto algorithm. Another test to determine if a number is prime, is based on Wilson's theorem [16], which states that: $(p-1)! \equiv -1 (\mathrm{mod}\ p)$, if p is prime, where $(p-1)! = 2 * 3 * 4 \ldots (p-1)$. In all other cases, $(n-1)! \equiv 0(\mathrm{mod}\ n)$, (except $n=4$). A test based on Wilson's theorem is also infeasible for large numbers due to the number of multiplications required to compute $(p-1)!$.

The two methods described above will determine with absolute certainty whether a number is prime or composite. Other approaches based on "probabilistic" algorithms include: Solovay-Strassen [15], Lehmann[16], Rabin-Miller[17] and Adleman-Pomerance-Rumley[18].

The prime number generation procedure presented in fig.9 incorporates the Rabin-Miller primality test

```
NN_ModInv ( a, x, n )
{       $g_0$ = n; $g_1$ = a;
        $u_0$ = 1; $v_1$ = 0;
        i = 1;
        while ($g_i$ != 0) do
        {       y = $g_{i-1}$ div $g_i$;
                $g_{i+1}$ = $g_{i-1}$ - y * $g_i$;
                $v_{i+1}$ = $v_{i-1}$ - y * $v_i$;
        }
        if ($v_{i-1}$ >= 0)
                then x = $v_{i-1}$;
                else x = $v_{i-1}$ + n;          /* negate if result is negative */
}
```

Fig. 8: Extended Euclidean algorithm to perform modular inverses

1. Generate a 256 bit random number p to test for primality.
2. Set the high order bit and low-order bit to 1. (The high order bit ensures that the prime is of the required length, and the low order bit ensures that it is odd.)
3. Check that p is not divisible by any of the following small primes: 3, 5, 7, 11 and so on. All primes less than 256 are used in this test.

- *Perform steps 4 to 10 (Rabin-Miller test) five times.*

4. Calculate b, where b is the largest power of 2 that divides p - 1. Then calculate m, such that $p = 1 + 2^b \star m$.
5. Choose a random number a from the interval (1 to p-1).
6. Set $j = 0$ and set $z = a^m \bmod p$.
7. If $z = 1$, or if $z = p-1$, then p passes the test and may be prime.
8. If $j > 0$ and $z = 1$, then p is not prime.
9. Set $j = j + 1$. If $j < b$ and $z \neq p - 1$, set $z = z^2 \bmod p$ and go back to step 8. If $z = p - 1$, then p passes the test and may be prime.
10. If $j = b$ and $z \neq p - 1$, then p is not prime.

Fig. 9: A prime number generation procedure.

algorithm. Using this algorithm p is almost certainly prime since there is only one chance in 2^{50} that p is composite.

Step 3 of the above procedure eliminates 80% of non prime numbers before the Rabin-Miller algorithm is activated. This greatly speeds up the process of prime searching. If the tests in step 3 are extended to include all primes up to 2000, then 85% of all 256-bit non primes would be eliminated. This percentage is calculated as follows: $(100 - (1.12/\ln n))\%$, where n is the upper test limit.

4.7 Pseudo-Random Number Generator

As part of the identification and signature schemes, random numbers R and E are required to be generated by the claimant and verifier respectively.

In generating R, a random string of 256 bits, the pseudo-random number generator from the 'C' language, system clock values, timer port values and the MD5 hashing algorithm can be used. A number (i.e. E) less than 256 bits can be generated by using randomly selected bits from the larger 256-bit number.

4.8 MD5 Hashing Algorithm

The MD5 hashing algorithm is a one-way function, which takes an input of arbitrary length and produces a unique, 128-bit, digest. The function is collision-free, so that it is not possible to create the same output from two different inputs. The MD5 algorithm has a number of strengths [19,20], such as: (i) it is collision free, (ii) it is suitable for high speed software implementations, (iii) it is optimised for microprocessor architectures, (iv) it is simple to implement.

The MD5 hashing algorithm computes the message digest by repeated application, of a compression function, to successive blocks of the message. The message is first padded so that its length is a multiple of 512 bit

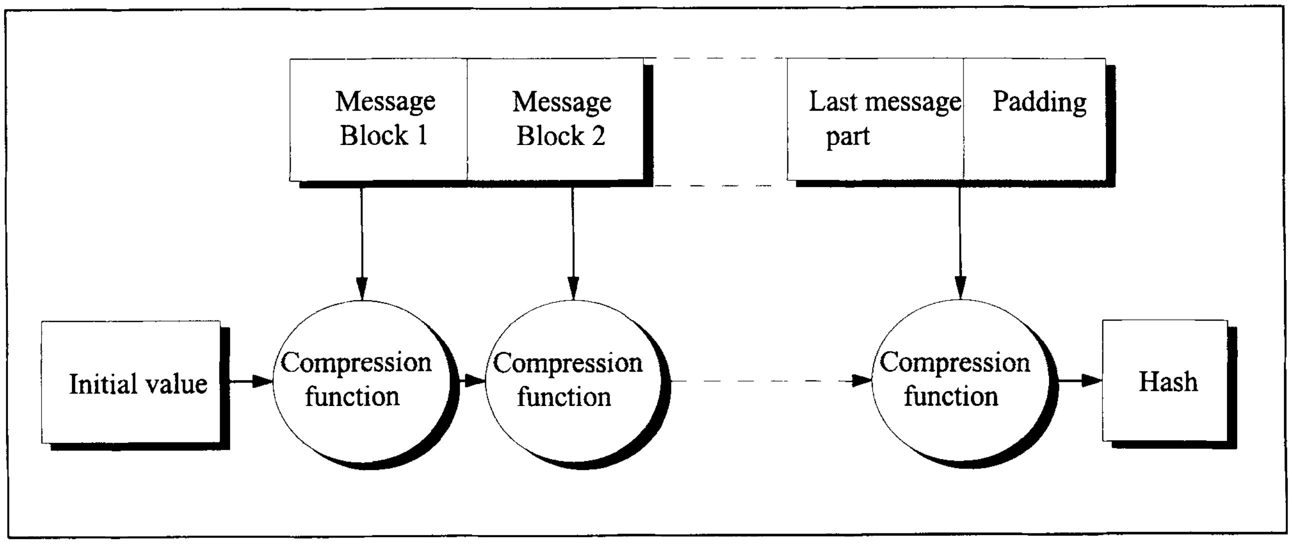

Fig. 10: MD5 Hashing algorithm structure

blocks. The padding consists of a single "1" bit and multiple "0" bits until the message length is congruent to 448 modulo 512. A 64-bit value, representing the message length, is then appended. The entire message length is now an exact multiple of 512 bits.

The compression function takes two inputs, a 128-bit chaining value and a 512-bit message block. It produces a new 128-bit chaining value which is input to the next iteration of the compression function, as seen in fig. 10. The initial chaining value is $0123456789ABCDEFFEDCBA9876543210_{16}$. The output of the MD5 hashing algorithm is a 128-bit value.

5. Analysis of the Realised Minimum-Knowledge Scheme

5.1 Storage requirements

As previously outlined the security level of the scheme is represented as L^{-tk}. If L is large ($\geq$ 30 bits) then t and k can equal 1, indicating that only one secret is required and only one cycle of the protocol is needed. The size of the protocol parameters are as follows: L = 30 or 72 bits for a security of 2^{-30} or 2^{-72} respectively, I = 512 bits, n = 512 bits, S = 512 bits, and c = 16 bits.

Thus the claimant's storage requirement is 134 or 139 bytes for a security level of 2^{-30} or 2^{-72} respectively.

5.2 Calculating modular multiplication requirements

The number of modular operations required during a verification cycle of the realised minimum-knowledge scheme can be obtained by analysing step 4 (a & b) of the verification procedure, in section 3.2. This step verifies $Y^L \star I_c^E \equiv X \pmod{n}$. For this calculation we define l as being the number of bits in E and L. The number of modular multiplications required can be calculated as follows:

for $X \pmod{n} = R^L \pmod{n}$

Number of modular multiplications required = l.

for $Y^L \star I_c^E \pmod{n}$

The computations of Y^L and I_c^E can be combined giving :

$\Rightarrow \quad (Y \star I_c)^E \star Y^{(L-E)} \qquad \{ \text{Since } 0 < E < L \}$

Since both E and L contain a maximum of l bits, the maximum number of modular multiplications (considering only the exponentiation operations) $= 2l$.

Thus, the total number of modular multiplications (M_{OO}) required to verify that $Y^L * I_c^E = X$ (MOD n): $M_{OO} = 2l + l = 3l$. (*eq.*5*a*).

The average number or modular multiplications ($\underline{M}_{OO}$) required:

$$\underline{M}_{OO} = 3l / 2. \qquad (eq.5b).$$

Correspondingly the average number of modular multiplications for the Fiat-Shamir ($\underline{M}_{FS}$) scheme [1]:

$$\underline{M}_{FS} = t(k+2)/2. \qquad (eq.\ 5c).$$

Equations 5*b* and 5*c* assume that the verifier and claimant operate at similar processing speeds. If the processing power of the verifier is significantly greater than that of the claimant (i.e. > 20 times faster), then the processing time of the verifier can be ignored, since it is insignificant compared to the claimant. If we consider only the claimant, the average number of modular multiplications for each scheme is now: $\overline{M}_{OO} = (2l+1)/2$ and $\overline{M}_{FS} = t(k+2)/2$. This gives an significant reduction in the number of modular multiplications required in the realised Ohta-Okamoto scheme, as shown in table 2.

5.3 Time for Identification

The time for performing an identification can be calculated by combining the data-transfer time between claimant and verifier, and the average processing time of the scheme. The data transfer time can be calculated using the standard data transfer operating speed of 9600 bps, as recommended by ISO 7816-3[21].

The processing times can be calculated if it is assumed, for example, that a single modular multiplication takes 0.1 seconds. Using $\overline{M}_{OO}$ and $\overline{M}_{FS}$, the calculated average processing times are given in table 3. Table 4 gives comparative identification times, assuming that the verifier's processing power is significantly greater than that of the claimant.

In practice, if the claimant and verifier execute on high speed processors the identification times are well less than one second.

6. Conclusion

This paper reviewed the Ohta-Okamoto minimum knowledge scheme and presented a realisation of a scheme incorporating peer-to-peer identification, signature generation and validation. The major algorithms required for the scheme's implementation were presented, including modular MP arithmetic functions, hashing algorithm, prime number generation and random number generation.

Analysis of the realised Ohta-Okamoto based scheme revealed the claimant's storage requirements, as well as the time required for identification. This analysis showed that the storage requirements of the realised scheme are significantly less than that of Fiat-Shamir, making it particularly suitable for minimum storage devices such as smart cards. It should be noted that as the security level of the Ohta-Okamoto scheme

	Realised Ohta-Okamoto scheme				Fiat-Shamir scheme			
Security level	Storage-Claimant (bytes)	Data transfer (bytes)	#Mod Muls - Claimant $(2l + 1)/2$	#Mod Muls - Total $3l / 2$	Storage - Claimant (bytes)	Data transfer (bytes)	#Mod Muls - Claimant $t(k+2)/2$	#Mod Muls - total $t(k+2)/2$
2^{-30}	134	134	30.5	45	650	850	20	20
2^{-72}	139	139	72.5	108	850	1242	44	44

Table 2: Comparison of minimum-knowledge schemes

	Realised Ohta-Okamoto scheme			Fiat-Shamir scheme		
Security level	Data transfer time.	Processing time	Total time for ID.	Data transfer time	Processing time	Total time for ID.
2^{-30}	0.1 sec	4.5 sec	4.6 sec	0.7 sec	2 sec	2.7 sec
2^{-72}	0.12 sec	10.8 sec	10.92 sec	1.04 sec	4.4 sec	5.44 sec

Table 3: Identification times for minimum-knowledge schemes.

	Realised Ohta-Okamoto scheme			Fiat-Shamir scheme		
Security level	Data transfer time.	Processing time	Total time for ID.	Data transfer time	Processing time	Total time for ID.
2^{-30}	0.1 sec	3.05 sec	3.15 sec	0.7 sec	2 sec	2.7 sec
2^{-72}	0.12 sec	7.25 sec	7.37 sec	1.04 sec	4.4 sec	5.44 sec

Table 4: Identification times considering claimant processing only.

increases, its storage requirements are relatively unaffected.

The identification speed of the scheme is slower than that of Fiat-Shamir, due to its increased computational complexity. However, if a high speed verifier is used, which is likely to be the case, a considerable improvement is obtained in identification times for the realised scheme.

References

1. Fiat, A. and Shamir, A.: 'How to prove yourself : Practical Solutions to Identification and Signature Problems'. Proceedings Crypto '86, Lecture notes in Computer Science, Vol. 263, Springer Verlag, 1987

2. Ohta, K. and Okamoto, T.: 'A modification of the Fiat-Shamir scheme'. Proceedings Crypto '88, Lecture notes in Computer Science, Vol. 403, Springer Verlag, 1987

3. Guillou, L.C. and Quisquater, J.J.: 'A practical Zero-Knowledge protocol fitted to security microprocessors minimising both transmission and memory'. Proceedings Eurocrypt '88, Lecture notes in Computer Science, Vol. 330, Springer Verlag, 1988

4. Schnorr, C.P.: 'Efficient identification and signatures for smart cards'. Proceedings Crypto '89, Lecture notes in Computer Science, Vol. 435, Springer Verlag, 1990

5. Micali, S. and Shamir, A.: 'An improvement of the Fiat-Shamir identification and signature scheme' Proceedings Crypto '88, Lecture notes in Computer Science, Vol. 403, Springer Verlag, 1987

6. Beth, T.: 'Efficient Zero-Knowledge identification scheme for smart cards'. Proceedings Eurocrypt '88, Lecture notes in Computer Science, Vol. 403, Springer Verlag, 1987

7. Stern, J.: 'An alternative to the Fiat-Shamir protocol'. Proceedings Eurocrypt '89, Lecture notes in Computer Science, Vol. 434, Springer Verlag, 1990

8. Alexi, W., Chor, B., Goldreich, O., and Schnorr, C.P.: 'RSA and Rabin Functions: Certain Parts are as Hard as the Whole'. SIAM Journal on Computing' vol 17, Apr 1988

9. Atkins, D., Graff, M., Lenstra, A.K. and Leyland, P.C.: 'The magic words are squeamish ossifrage'. Advances in Cryptology - Asiacrypt '94, Springer-Verlag, 1995

10. Buhler, J.P., Lenstra, H.W., and Pomerance, C.: 'The development of the number field sieve'. Volume 1554 of Lecture Notes in Computer Science, Springer-Verlag, 1994

11. Buchmann, J., Loho, J., and Zayer, J.: 'An implementation of the general number field sieve'. Advances in Cryptology - Crypto '93, pages 159-166, Springer-Verlag, 1994

12. Dodson, B. and Lenstra, A.K.: 'NFS with four large primes: An explosive experiment'. Advances in Cryptology - Crypto '95, pages 372-385, Springer-Verlag, 1995

13. Knuth, D.: 'The Art of Computer Programming: Vol 2, Seminumerical Algorithms' 1983

14. Ore, O.:'Number Theory and its history' (McGrath Hill, 1948)

15. Solovay, R., Strassen,V.:'A fast Monte-Carlo test for primality'. SIAM Journal on computing, 1978

16. Lehmann, D.J.: 'On Primality Tests'. SIAM Journal on computing, vol 11, May 1982

17. Rabin, M.O.: 'Probabilistic Algorithm for Testing Primality'. Journal of Number Theory, vol 12, Feb 1980

18. Adlemen, L.M., Pomerance, C., Rumley, R.S.: 'On Distinguishing Prime Numbers from Composite Numbers'. Annals of Mathemetics,Vol. 117, No.1, 1983, pp 173-206

19. Rivest, R.:'The MD5 Message-Digest Algorithm: Request For Comments 1321'. MIT Laboratory for Computer Science and RSA Data Security, Inc. April 1992

20. Kaliski, B., Robsham, M.:'Message Authentication with MD5'. RSA Laboratories, http://www.rsa.com/rsalabs/pubs/cryptobytes/spring95/md5.htm, July 1995

21. ISO/IEC 7816-3: 'Identification cards-Integrated circuit(s) cards with contacts-Part 3: Electronic signals and transmission protocols', 1989