Parameter Selection for Server-Aided RSA Computation Schemes

John Burns and Chris J. Mitchell

Abstract— **The security, complexity, and application of two schemes for using an untrusted auxiliary processor to aid smart card RSA signature computations are reviewed, including detailed analysis of possible methods of attack. Guidance is given on practical, secure use of these schemes.**

Index Terms—**Digital signature, modular exponentiation, RSA, server-aided computation, smart card.**

I. Introduction

A. Scope and Purpose

MANY potential applications exist for smart cards capable of performing digital signatures. However, currently available smart cards tend to have very limited processing power, and signature algorithms, such as RSA, are computationally intensive. As a result, performing a single signature operation on a smart card can take far too long (e.g., tens of seconds) for practical applications to be feasible.

In a recent paper by Matsumoto *et al.* [4], two schemes (which we refer to as the *MKI schemes*) for using an untrusted auxiliary processor to speed up smart card RSA signatures were introduced. One main reason for introducing such schemes is that smart card readers could easily be equipped with more sophisticated processors capable of high-speed multiprecision arithmetic. Of course, if the auxiliary processor (in a card reader or elsewhere) were to be trusted, then the auxiliary processor could be given the secret RSA key and asked to perform the whole calculation. However, this is not normally the case, and the whole objective of these schemes is to make use of the auxiliary processor without compromising the secrecy of the RSA private key.

The main purpose of this paper is to suggest choices for parameters for the MKI schemes, and to indicate how they can be used without risking compromise of the secret RSA key. There is a marked absence of any existing published guidance on the choice of parameters for such schemes, and providing this guidance is a nontrivial problem.

B. Preliminary Notation and Assumptions

The basic objective of the protocols discussed herein is to compute an RSA signature using a smart card. We assume throughout that the smart card is equipped with a secret RSA key, consisting of a secret exponent d, and a pair of primes p, q with the property that

$$3d \equiv 1 \pmod{\lambda(N)},$$

where $N = pq$ and, hence, $\lambda(N) = \mathrm{lcm}(p-1, q-1)$ (see, for example, [3, p. 19]). In other words, we are assuming that the RSA key is chosen so that the public exponent is 3; this is a reasonable and apparently secure assumption given that the key is used only for signature and not for data encryption [2].

Then we assume that the smart card needs to compute a signature on a message M, where $0 \le M \le N-1$; that is, the smart card wishes to compute

$$M^d \bmod N.$$

To make maximum use of the auxiliary processor, it will be necessary to give certain information to this processor regarding the value of d. However, this information must be limited so that an infeasible amount of work remains for this processor to perform to deduce the value of d. Throughout this paper we assume that the information released to the auxiliary processor regarding d must be limited so that at least $2^{64} \sim 10^{19}$ trials will still be required to deduce d.

Finally, to make the procedure worthwhile, we assume that the smart card has very limited computing power, but that the auxiliary processor is capable of performing fast multiprecision arithmetic (in particular, fast multiprecision modular exponentiation).

II. MKI Schemes

A. RSA-S1

The first scheme, called RSA-S1 by Matsumoto *et al.* [4] is the simpler of the two. It operates as follows.

The fundamental idea is to represent d (the secret exponent) as

$$d \equiv f_1 d_1 + f_2 d_2 + \cdots + f_k d_k \pmod{\lambda(N)},$$

where $f_1, f_2, \cdots, f_k$ are all small.

The auxiliary processor is equipped with the values $d_1, d_2, \cdots, d_k$ and N and is requested to compute the following values:

$$z_1 = M^{d_1} \bmod N, \quad z_2 = M^{d_2} \bmod N, \cdots,$$

$$z_k = M^{d_k} \bmod N,$$

Manuscript received June 4, 1991; revised September 18, 1992, and January 4, 1993. This work was supported in part by Hewlett-Packard Laboratories, Bristol, England.

J. Burns is with Hewlett-Packard Laboratories, Bristol, England.
C. J. Mitchell is with the Department of Computer Science, Royal Holloway, University of London, Surrey, England.

IEEE Log Number 9212692.

which are returned to the smart card. The smart card then computes

$$M^d \equiv \prod_{i=1}^{k} (z_i)^{f_i} \pmod N,$$

which should involve relatively small numbers of modular multiplications given that k and $f_1, f_2, \cdots, f_k$ are not too large. Typically one would require each f_i to satisfy $0 \leq f_i \leq a-1$ for some a.

Note that the scheme actually described in [4] has $a = 2$; that is, every f_i is either 0 or 1. In addition, Matsumoto *et al.* suggest requiring that at most ℓ of the values $f_1, f_2, \cdots, f_k$ be nonzero (thereby limiting the number of multiplications that the smart card is required to perform).

B. RSA-S2

This algorithm improves the performance of the RSA-S1 algorithm by using the Chinese Remainder Theorem (CRT) and taking advantage of the fact that the smart card can, without loss of security, be equipped with the factors p and q of N.

The basic idea for speeding up secret key exponentiation by using the CRT is well known, and dates back to Quisquater and Couvreur [5]. It works as follows. Suppose that $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. If $S_p = M^{d_p} \bmod p$ and $S_q = M^{d_q} \bmod q$, then

$$M^d = (S_p w_p + S_q w_q) \bmod N,$$

where

$$w_p = q(q^{-1} \bmod p)$$

and

$$w_q = p(p^{-1} \bmod q)$$

and, hence, $w_p \equiv 1 \pmod p, w_p \equiv 0 \pmod q$, $w_q \equiv 0 \pmod q$, and $w_q \equiv 1 \pmod p$.

Now computing S_p and S_q is, in general, significantly easier than computing M^d directly, and, since w_p and w_q can be precomputed and stored, this gives a significantly faster way of computing a signature.

This idea can be applied to the RSA-S1 scheme described earlier, and this is the basis of RSA-S2, which we now describe. Suppose w_p, w_q, d_p, and d_q are as given earlier. Represent d_p as

$$d_p \equiv f_1 d_1 + f_2 d_2 + \cdots + f_k d_k \pmod{p-1}$$

and represent d_q as

$$d_q \equiv g_1 d_1 + g_2 d_2 + \cdots + g_k d_k \pmod{q-1},$$

where $f_1, f_2, \cdots, f_k, g_1, g_2, \cdots, g_k$ are all small.

As in RSA-S1, the auxiliary processor is equipped with the values $d_1, d_2, \cdots, d_k$ and N and is requested to compute the following values:

$$z_1 = M^{d_1} \bmod N, \quad z_2 = M^{d_2} \bmod N, \cdots,$$

$$z_k = M^{d_k} \bmod N,$$

which are returned to the smart card. The smart card then computes the following:

$$S_p \equiv \prod_{i=1}^{k} (z_i)^{f_i} \pmod p$$

$$S_q \equiv \prod_{i=1}^{k} (z_i)^{g_i} \pmod q$$

and

$$M^d = (S_p w_p + S_q w_q) \bmod N,$$

which involves relatively small numbers of modular multiplications given that the values of $k, f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ are not too large. Note also that all these modular multiplications (apart from the last two) are of "half length," that is, modulo p and q which are of the order of $\sqrt{N}$. As before, one would normally require each f_i and g_i to fall in the range $[0, a-1]$ for some a.

The preceding algorithm actually differs in two small respects from the algorithm described by Matsumoto *et al.* [4]. First, they assume that $a = 2$, that is, every f_i and g_i is either 0 or 1; however, they do note that other values of a can be used. Second, they have the additional constraint that at most ℓ of the values $f_1, f_2, \cdots, f_k, g_1, g_2, \cdots, g_k$ are nonzero, thereby limiting the number of multiplications that the smart card is required to perform.

Before proceeding, observe that, to simplify much of the discussions that follow, we will assume that a is a power of 2; that is, we assume that $a = 2^c$ for some positive integer c.

C. Other Schemes

First, note that an even simpler scheme for reducing the computation required from the smart card would be to choose the secret RSA exponent to be small, say choose d so that $d < 2^{64}$. Although this would mean that the public exponent e would typically be of the order of N, this would not necessarily be a problem because checking RSA signatures will often be performed by large, powerful processors. However, this approach has been shown to be fatally flawed by Wiener [8]. Basically, Wiener has shown that if e is public and it is known that d satisfies

$$0 \leq d \leq N^{1/4}$$

then N can be factorized in polynomial time. Nevertheless, as shown in [8], the attack that finds d if $d < N^{1/4}$ can be circumvented by adding a multiple of $\lambda(N)$ to e so that $e > N^{3/2}$. This is another possible approach to reducing the load on the smart card.

Another alternative is the adoption of an additional variant of RSA-S1 proposed by Quisquater and De Soete [6]. This variant, although not without certain advantages, requires a significantly greater amount of computation to be performed by the smart card as well as a much greater volume of data to be transferred between the smart card and the auxiliary processor. We do not consider it further here.

III. Computations Must Always be Checked

Next, we consider what countermeasures may be required against an auxiliary processor that attempts to obtain information about d by supplying the smart card with incorrect values for certain of $z_1, z_2, \cdots, z_k$. We start by examining RSA-S1.

One possible attack is as follows. Suppose the auxiliary processor supplies the smart card with correct values for z_i for all values of i except for $i = j$, in which case it provides $z_j = tM^{d_j} \bmod N$ for some t with the property that $t^0, t^1, t^2, t^3, \cdots, t^{a-1}$ are all distinct (mod N). Using the supplied values, the smart card will actually compute

$$t^{f_j} M^d \bmod N,$$

which we suppose it outputs as the required signature, S say. Assuming the auxiliary processor has access to this "signature", it can deduce the value of f_j by the following procedure. First, compute

$$S^3 \equiv t^{3f_j} M^{3d} \equiv t^{3f_j} \pmod N.$$

Now compute $t^{3i} \bmod N\,(i = 0, 1, 2, \cdots)$ until a match is found with S^3. The corresponding value of i will then equal f_j. Note that, in practice, the values of $t^{3i} \bmod N$ can be precomputed.

If the main objective of the auxiliary processor is to fraudulently obtain the smart card's signature on a message of its own choosing, then an even simpler attack will work. Suppose the message the auxiliary processor wishes to have signed is X. Then, when requested for the values $M^{d_i} \bmod N$, the auxiliary processor instead supplies $X^{d_i} \bmod N$ for every i. The signature output by the smart card will then be simply $X^d \bmod N$, as required by the auxiliary processor.

The only way to defeat these attacks is for the smart card to have some means of checking either the provided information or the final signature. The latter seems easiest since we have already assumed that the public RSA exponent is always 3. To check the signature, it is therefore only necessary to cube it mod N and verify that it agrees with the original message. This has the relatively small cost of adding two modular multiplication operations to the total work that needs to be performed by the smart card. Note that the need to protect against possible fraud by the auxiliary processor has been mentioned elsewhere, as has the possibility of checking the computation by cubing it mod N [6].

We now consider the same problem for the RSA-S2 scheme. The situation is not quite as simple as for RSA-S1, although attacks are still possible. We start by examining an attack due to Shimbo and Kawamura [7]. Suppose the auxiliary processor supplies the smart card with correct values for z_i for all values of i except for $i = j$, in which case it provides $z_j = -M^{d_j} \bmod N$. Using the supplied values, the smart card will actually compute

$$S_p \equiv \prod_{i=1}^{k} (z_i)^{f_i} \equiv (-1)^{f_j} M^d \pmod p$$

$$S_q \equiv \prod_{i=1}^{k} (z_i)^{g_i} \equiv (-1)^{g_j} M^d \pmod q,$$

and then the signature S will be derived as

$$S = (S_p w_p + S_q w_q) \bmod N.$$

If S is output by the smart card and intercepted by the auxiliary processor, then the auxiliary processor will (almost always) be able to use it to factorize N if $f_j \not\equiv g_j$ (mod 2). To show how this attack operates we assume without loss of generality that f_j is even and g_j is odd. Hence $S_p \equiv M^d \pmod p$ and $S_q \equiv -M^d \pmod q$, and thus $S \equiv M^d \pmod p$ and $S \equiv -M^d \pmod q$.

First, the cryptanalyst computes

$$X = S^e + M \bmod N.$$

Hence, $X \equiv 2M \pmod p$ and $X \equiv 0 \bmod q$). By inspection,

$$2w_p M \equiv 2M \pmod p$$

and

$$2w_p M \equiv 0 \pmod q$$

and thus,

$$X \equiv 2w_p M \pmod N.$$

Next, suppose that $p \nmid M$ (hence the "almost always" earlier). Then,

$$(X, p) = (2w_p M, p) = 1,$$

since $w_p \equiv 1 \pmod p$. Hence, $(X, N) = q$, and one application of the Euclidean algorithm will reveal the factorization of N.

Shimbo and Kawamura [7] go on to describe two ways in which the attack might be prevented. First, they suggest checking the signature before disclosing it, that is, computing $S^3 \bmod N$ and verifying that this gives M. Second, they suggest choosing the values $f_1, f_2, \cdots, f_k, g_1, g_2, \cdots, g_k$ such that $f_i \equiv g_i$ (mod 2) for every i.

This latter suggestion certainly defeats the attack described earlier. However, it is not secure against the following naive attack. Suppose the main objective of the auxiliary processor is to obtain fraudulently the smart card's signature on a message of its own choosing. If the message the auxiliary processor wishes to have signed is X, then, when requested for the values $M^{d_i} \bmod N$, the auxiliary processor instead supplies $X^{d_i} \bmod N$ for every i. The signature output by the smart card will then be simply $X^d \bmod N$, as required by the auxiliary processor.

The only way to defeat this latter attack is for the smart card to have some means of checking either the information provided or the final signature. The latter seems easier since we have already assumed that the public RSA exponent is always 3, making the first proposal of Shimbo and Kawamura the obvious choice. This conclusion leads naturally to the topic of the next section, namely Gollmann's attack.

IV. Gollmann Attack and Possible Remedies

A. Preliminary Remarks

The Gollmann attack [1] arises out of the need for the smart card to check signatures before releasing them. Following such a check, the smart card will be faced with two possible scenarios.

- The signature is correct, in which case it should be issued from the smart card as required.
- The signature is incorrect, in which case the supplied values z_i should be rejected and the signature not output (lest it reveal information to a malicious auxiliary processor).

Therefore, it seems inevitable that the auxiliary processor will always be aware of whether the values z_i supplied to the smart card result in the correct signature.

This observation then leads to Gollmann's attacks on the RSA-S1 and RSA-S2 protocols. We consider these attacks and possible countermeasurs separately below.

B. RSA-S1

We need to consider two possible cases.

1) Suppose the auxiliary processor supplies the smart card with correct values for z_i for all values of i except for $i = j$, in which case it provides $z_j = -M^{d_j} \bmod N$. Using the supplied values, the smart card will actually compute

$$S \equiv \prod_{i=1}^{k} (z_i)^{f_i} \equiv (-1)^{f_j} M^d \pmod N.$$

That is, the smart card will compute the correct signature, if and only if

$$f_j \equiv 0 \pmod 2).$$

2) Suppose the auxiliary processor supplies the smart card with correct values for z_i for all values of i except for $i = j$, in which case it provides $z_j \neq \pm M^{d_j} \bmod N$. Using the supplied values, the smart card will actually compute

$$S \equiv \prod_{i=1}^{k} (z_i)^{f_i} \equiv (z_j M^{-d_j})^{f_j} M^d \pmod N.$$

That is, the smart card will compute the correct signature if and only if

$$(z_j M^{-d_j})^{f_j} \equiv 1 \pmod N.$$

Given that $z_j \neq \pm M^{d_j} \bmod N$, the probability of the preceding equation being satisfied with randomly chosen z_j is vanishingly small unless

$$f_j = 0.$$

Using these two possible means of attack, it is therefore impossible to prevent a conspiring group of malicious auxiliary processors from discovering whether 1) $f_j \equiv 0 \pmod 2$ and 2) $f_j = 0$ for any $j\,(1 \leq j \leq k)$.

The best option for the system user wishing to counter this attack would appear to be to choose f_i odd for every $i, 1 \leq i \leq k$, in which case neither condition 1 nor 2 above would be true for any $i\,(1 \leq i \leq k)$. This would have the effect that any attempt to deceive the smart card would be detected; that is, every time false data are presented to the smart card it results in an incorrect signature, and resultant rejection of the data supplied by the auxiliary processor.

This means that, for the cryptanalyst attempting to determine $f_1, f_2, \cdots, f_k$ (and hence d), only 2^{c-1} values will need to be tried instead of 2^c. Thus the size of the search space for the cryptanalyst effectively becomes $2^{(c-1)k}$ instead of 2^{ck}. However, this reduction in search space size is of minor significance by comparison with the effect of the divide-and-conquer attack on RSA-S1 described in Section V.

C. RSA-S2

As with RSA-S1, we need to consider two possible cases.

1) Suppose the auxiliary processor supplies the smart card with correct values for z_i for all values of i except for $i = j$, in which case it provides $z_j = -M^{d_j} \bmod N$. Using the supplied values the smart card will actually compute

$$S_p \equiv \prod_{i=1}^{k} (z_i)^{f_i} \equiv (-1)^{f_j} M^d \pmod p$$

$$S_q \equiv \prod_{i=1}^{k} (z_i)^{g_i} \equiv (-1)^{q_j} M^d \pmod q$$

and then the signature S will be derived as

$$S = (S_p w_p + S_q w_q) \bmod N.$$

This means that the computed signature S will satisfy

$$S \equiv (-1)^{f_j} M^d \pmod p$$

and

$$S \equiv (-1)^{g_j} M^d \pmod q.$$

That is, the smart card will compute the correct signature, if and only if

$$f_i \equiv g_j \equiv 0 \pmod 2).$$

2) Suppose the auxiliary processor supplies the smart card with correct values for z_i for all values of i except for $i = j$, in which case it provides $z_j \neq \pm M^{d_j} \bmod N$. Using the supplied values the smart card will actually

compute

$$S_p \equiv \prod_{i=1}^{k}(z_i)^{f_i} \equiv (z_j M^{-d_j})^{f_j} M^d \pmod p$$

$$S_q \equiv \prod_{i=1}^{k}(z_i)^{g_i} \equiv (z_j M^{-d_j})^{g_j} M^d \pmod q$$

and then the signature S will be derived as

$$S = (S_p w_p + S_q w_q) \bmod N.$$

This means that the computed signature S will satisfy

$$S \equiv (z_j M^{-d_j})^{f_j} M^d \pmod p$$

and

$$S \equiv (z_j M^{-d_j})^{g_j} M^d \pmod q.$$

That is, the smart card will compute the correct signature, if and only if

$$(z_j M^{-d_j})^{f_j} \equiv 1 \pmod p$$

and

$$(z_j M^{-d_j})^{g_j} \equiv 1 \pmod q.$$

Given that $z_j \neq \pm M^{d_j} \bmod N$, the probability of the preceding equations being satisfied with randomly chosen z_j is vanishing small unless

$$f_j = g_j = 0.$$

Using these two possible means of attack it is, therefore, impossible to prevent a conspiring group of malicious auxiliary processors from discovering whether 1) $f_j \equiv g_j \equiv 0 \pmod 2$ and 2) $f_j = g_j = 0$ for any $j\,(1 \leq j \leq k)$.

The best option for the system user wishing to counter this attack would appear to be to choose f_i and g_i so that at most one of them is even for every $i\,(1 \leq i \leq k)$; that is, neither condition 1 or 2 above can be true for any $i\,(1 \leq i \leq k)$. This would have the effect that any attempt to deceive the smart card would be detected; that is, every time false data are presented to the smart card it results in an incorrect signature, and resultant rejection of the data supplied by the auxiliary processor. There appear to be two main approaches to doing this.

1) Suppose that $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ are chosen at random subject to the constraint that at least one of f_j and g_j is odd for every $j\,(1 \leq j \leq k)$. This means that, for the cryptanalyst attempting to determine $f_1, f_2, \cdots, f_k$ (and hence d_p and d) it holds that $f_j \equiv 1 \pmod 2$ with probability 2/3; that is, the entropy of the least significant bit of f_j is

$$(\log_2(3) + 2\log_2(3/2))/3 = \log_2(3) - 2/3 \simeq 0.9183.$$

Thus, the size of the search space for the cryptanalyst effectively becomes $2^{(c-0.0817)k}$ instead of 2^{ck}. To see the effect of this reduction we tabulate $(c - 0.0817)k$ (i.e., the effective "number of bits of security") for some typical values of c and k (see Table I).

TABLE I
EFFECTIVE NUMBERS OF BITS OF SECURITY

c	k	$(c-0.0817)k$
1	64	58.8
2	32	61.4
4	16	62.7
8	8	63.3
16	4	63.7
32	2	63.8

2) An alternative approach appears to avoid reducing the effective system security below 2^{ck}. In this approach, we again choose $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ at random, this time subject to the (more restrictive!) constraint that for every j exactly one of f_j and g_j is even (and exactly one is odd).

Paradoxically, this would appear to make work for the cryptanalyst just as difficult as if $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ were chosen at random with no constraints at all! Of course, work is made simpler for anyone attempting to discover $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ simultaneously, but there is little point in making such an attempt since knowledge of $f_1, f_2, \cdots, f_k$ alone will easily reveal d (as discussed in Section VI-B-1).

It is also curious that this method of choosing $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ is precisely the opposite of one of the means suggested by Shimbo and Kawamura [7] for preventing the original malicious auxiliary processor attack (see Section III). Hence, not only does Shimbo and Kawamura's suggestion fail to prevent auxiliary processor attacks, it actually is the converse of what is required!

It would appear that the second of the preceding two options is the best to follow, given that it defeats Gollmann's attack without reducing the effective security of the system. Note also that this gives an additional advantage for RSA-S2 over RSA-S1, since Gollmann's attack on RSA-S1 cannot be counteracted without reducing overall system security.

V. DIVIDE-AND-CONQUER ATTACK ON RSA-S1

Next, we present a simple yet very effective attack, due to Wiener, on the security of the RSA-S1 protocol that essentially makes it much less practical to use. However, a similar attack on RSA-S2 has yet to be found.

After a single use of the RSA-S1 protocol, an untrustworthy auxiliary processor (or any cryptanalyst party to the communications between the smart card and the auxiliary processor) will be equipped with a set of values $d_1, d_2, \cdots, d_k$. These values have the property that

$$d \equiv f_1 d_1 + f_2 d_2 + \cdots + f_k d_k \pmod{\lambda(N)},$$

where d is the smart card secret exponent and $f_1, f_2, \cdots, f_k$ are also known only to the smart card. The attacker will also know e (the public exponent) and N (the RSA modulus). For the sake of simplicity, we assume that k is even, say $k = 2\ell$. The attack we describe allows the attacker to deduce the values of $f_1, f_2, \cdots, f_k$ (and hence d) in at most $2^{(c-1)k/2+1}$ trials, that is, effectively reducing the "number of bits of security" by approximately 50%.

Before describing the attack we need some notation. Let F_1 be the set of all possible values for the parameters $f_1, f_2, \cdots, f_\ell$, that is,

$$F_1 = \{(g_1, g_2, \cdots, g_\ell): 0 \le g_i < a\}.$$

Then $|F_1| = 2^{c\ell}$, so label the elements of F_1

$$\mathbf{g}_0, \mathbf{g}_1, \cdots, \mathbf{g}_{2^{c\ell}-1}.$$

Let F_2 be the set of all possible values for the parameters $f_{\ell+1}, f_{\ell+2}, \cdots, f_{2\ell}$, that is,

$$F_2 = \{(g_{\ell+1}, g_{\ell+2}, \cdots, g_{2\ell}): 0 \le g_i < a\}.$$

Then, $|F_2| = 2^{c\ell}$, so label the elements of F_2

$$\mathbf{h}_0, \mathbf{h}_1, \cdots, \mathbf{h}_{2^{c\ell}-1}.$$

The attacker now proceeds as follows.

1) Choose a message m at random, and compute $x = m^e \bmod N$.
2) Compute $w = x^{-1} \bmod N$ (e.g., using the Euclidean Algorithm).
3) For each $\mathbf{g}_i \in F_1, \mathbf{g}_i = (g_1, g_2, \cdots, g_\ell)$ say, compute

$$y_i = mw^{g_1 d_1 + g_2 d_2 + \cdots + g_\ell d_\ell} \bmod N.$$

   Store these results in some form by which they may be accessed easily by the values of y_i (e.g., either sorted by these values or stored in a hash table).
4) For each $\mathbf{h}_j \in F_2$, $\mathbf{h}_j = (g_{\ell+1}, g_{\ell+2}, \cdots, g_{2\ell})$ say, compute

$$z_j = x^{g_{\ell+1} d_{\ell+1} + g_{\ell+2} d_{\ell+2} + \cdots + g_{2\ell} d_{2\ell}} \bmod N.$$

5) Then check to see if there exists a y_i such that $z_j = y_i$ (this is straightforward since we assume that the values y_i are stored in a form where this type of check is simple, for example, in a hash table). If so then do the following: Suppose $\mathbf{g}_i = (g_1, g_2, \cdots, g_\ell)$ and $\mathbf{h}_j = (g_{\ell+1}, g_{\ell+2}, \cdots, g_{2\ell})$. Then compute and store

$$\delta_i = g_1 d_1 + g_2 d_2 + \cdots + g_k d_k.$$

At the end of the preceding procedure the attacker will have a set of values δ_i, and it is straightforward to check that one of these will be congruent to d modulo $\lambda(N)$, which is essentially equivalent to knowing d. Hence the attacker needs to perform only $2a^{k/2} = 2^{ck/2+1}$ exponentiations and $a^{k/2}$ comparisons to be guaranteed to discover the RSA secret key.

In fact, the situation is a little worse than this, since, to defeat the attack described in Section IV-B, we have already assumed that f_i is odd for every $i\,(1 \le i \le k)$. Hence the sets F_1 and F_2 need contain only $\lfloor a/2 \rfloor^{k/2}$ entries, and so the attacker needs to perform only $2\lfloor a/2 \rfloor^{k/2} = 2^{(c-1)k/2+1}$ exponentiations and $\lfloor a/2 \rfloor^{k/2} = 2^{(c-1)k/2}$ comparisons to be guaranteed to discover the RSA secret key.

This means that, for the cryptanalyst attempting to determine $f_1, f_2, \cdots, f_k$ (and hence d), only $2^{(c-1)k/2+1}$ checks will need to be performed instead of 2^{ck}. Thus the size of the search space for the cryptanalyst effectively becomes $2^{(c-1)k/2+1}$ instead of 2^{ck}. Hence, for RSA-S1, one must choose c and k larger than would otherwise be the case to compensate for this reduced freedom in choosing the values $f_1, f_2, \cdots, f_k$—this point is discussed in more detail in Section VI-B-1.

VI. Choice of Parameters

A. Complexity of Algorithms

We start by reviewing the complexity of the algorithms in terms of the expected number of calculations the smart card will be required to perform.

Theorem 6.1: Using the RSA-S1 Protocol, and assuming that $f_i \equiv 1 \pmod 2$ for every i, a single modular exponentiation will require the smart card to perform the following expected number of modular multiplication operations (mod N):

$$3(c-1)k/2 + 2^{1-c}k - 1.$$

Proof: We start by assuming that, apart from being odd, the values of f_i are selected randomly; this is a good assumption since this will maximize the work involved for the cryptanalyst in computing d from the values of $d_1, d_2, \cdots, d_k$. Since every f_i is odd, it follows that $f_i \ne 0$ for every i, and hence the total number of modular multiplications will be $k-1$.

Within each exponentiation calculation (assuming the use of right-to-left square-and-multiply), the number of multiplications will be $w - 1$, where w is the number of ones in the binary representation of f_i, and the number of squaring operations will be $h - 1$, where h is the length of the binary representation of f_i. The expected value of w is simply

$$\sum_{j=0}^{2^{c-1}-1} \mathrm{wt}(2j+1) \Big/ 2^{c-1} = (c+1)/2$$

(where $\mathrm{wt}(i)$ denotes the binary weight of i). The expected value of h is given by

$$\sum_{j=0}^{2^{c-1}-1} \mathrm{bl}(2j+1) \Big/ 2^{c-1} = \sum_{i=2}^{c} i2^{i-2} + 1 \Big/ 2^{c-1}$$

(where $\mathrm{bl}(i)$ denotes the binary length of i). If we let

$$R(c) = \sum_{i=2}^{c} i2^{i-2}$$

then it trivially follows that

$$R(c) = R(c-1) + c2^{c-2}.$$

Hence, (by induction) it is simple to show that

$$R(c) = (c-1)2^{c-1}.$$

Using this it should now be clear that the expected value of h is simply $c - 1 + 2^{1-c}$. Hence, the expected number of operations in each exponentiation operation is

$$3c/2 - 5/2 + 2^{1-c}.$$

Hence the overall total for the expected number of modular multiplications is

$$k - 1 + 3kc/2 - 5k/2 + 2^{1-c}k,$$

which simplifies to

$$3(c-1)k/2 + 2^{1-c}k - 1.$$

and the proof is complete. □

Theorem 6.2: Using the RSA-S2 protocol, a single modular exponentiation will require the smart card to perform the following expected numbers of operations:

$$3ck - 4k(1-2^{-c}) + 2^{-ck+1} - 2$$

modular multiplications (mod p or mod q) together with $2k(1-2^{-c})$ modular reductions (from mod N to mod p or mod q), two modular multiplications (mod N), and a single modular addition (mod N).

Proof: We start by considering the computation of S_p, that is, the computation of

$$\prod_{i=1}^{k}(z_i)^{f_i} \bmod p.$$

As before, we assume that the values of f_i are selected randomly.

The probability of any particular parameter f_i being zero is 2^{-c}; hence, the expected number of nonzero elements in the set $F = \{f_1, f_2, \cdots, f_k\}$ is $k(1-2^{-c})$. Now the total number of modular reductions from mod N to mod p that will need to be performed will be exactly equal to the number of nonzero elements in F. Similarly, the total number of modular multiplications will be precisely one less than the number of nonzero elements in F, unless all the elements in F are zero.

Hence, the expected number of modular reductions from mod N to mod p is

$$k(1-2^{-c}),$$

and the expected number of modular multiplications (mod p) into the partial product is

$$k(1-2^{-c}) - 1 + 2^{-ck}.$$

Within each exponentiation calculation (assuming the use of right-to-left square-and-multiply), the number of multiplications will be $w-1$, where w is the number of ones in the binary representation of f_i (except if $f_i = 0$ when we set $w = 1$), and the number of squaring operations will be $h-1$, where h is the length of the binary representation of f_i (again except if $f_i = 0$ when we set $h = 1$). The expected value of w is simply

$$\sum_{i=1}^{2^c-1} \mathrm{wt}(i) + 1 \Big/ 2^c = c/2 + 2^{-c}$$

[where $\mathrm{wt}(i)$ denotes the binary weight of i]. The expected value of h is given by

$$\sum_{i=1}^{2^c-1} \mathrm{bl}(i) + 1 \Big/ 2^c = \sum_{i=1}^{c} i2^{i-1} + 1 \Big/ 2^c$$

[where $\mathrm{bl}(i)$ denotes the binary length of i]. If we let

$$S(c) = \sum_{i=1}^{c} i2^{i-1}$$

then it trivially follows that

$$S(c) = S(c-1) + c2^{c-1}.$$

Hence (by induction) it is simple to show that

$$S(c) = (c-1)2^c + 1.$$

Using this it should now be clear that the expected value of h is simply $c - 1 + 2^{1-c}$. Hence the expected number of operations in each exponentiation operation is

$$3(c/2 - 1 + 2^{-c}).$$

Hence, the overall total for the expected number of modular multiplications (mod p) is given by

$$3ck/2 - 2k(1-2^{-c}) + 2^{-ck} - 1.$$

Given that the computation of S_q has a similar complexity the result follows. □

B. Choice for a and k

The preceding results will help us with the selection of these two fundamental parameters. First, we note some other vital considerations regarding the choice of these two values.

1) Elementary Lower Bound: For RSA-S1, it is clearly necessary for the values k and a to be chosen so that it is infeasible for any cryptanalyst to deduce d from knowledge of $d_1, d_2, \cdots, d_k$ alone. In general, if the only information the cryptanalyst has is $d_1, d_2, \cdots, d_k$ and the value of a (the upper bound on the values of $f_1, f_2, \cdots, f_k$) then, using the divide-and-conquer attack described in Section V, there will be $2\lfloor a/2 \rfloor^{k/2}$ exponentiation operations to perform. Hence, we require that

$$2\lfloor a/2 \rfloor^{k/2} \geq 2^{64}.$$

If, as before, we assume that $a = 2^c$, then this inequality becomes

$$(c-1)k \geq 126.$$

For RSA-S2 the situation is a little more complex. We are now dealing with two values, that is, d_p and d_q, the value of which to the cryptanalyst is not so obvious. However, we now show that knowledge of just one of these values is sufficient to enable a cryptanalyst to factorize N.

Suppose a cryptanalyst has by some means obtained the value of d_p. Then

$$3d_p \equiv 3d \equiv 1 \pmod{p-1},$$

that is, $p-1 | 3d_p - 1$. Now, by definition, $d_p \leq p-1$. Hence, $3d_p - 1 < 3(p-1)$, and thus $3d_p - 1$ is either $p-1$ or $2(p-1)$. Hence, given d_p, one simple test will reveal p, and thus N is factored.

This means that it is necessary to make it as difficult to deduce d_p or d_q from $d_1, d_2, \ldots, d_k$ and a as it was to deduce

TABLE II
SMART CARD COMPLEXITIES OF RSA-S1 AND RSA-32

k	64	32	16	8	4	2
c (for RSA-S1)	3	5	9	17	33	64
RSA-S1 complexity	207	193	191	191	191	188
c (for RSA-S2)	1	2	4	8	16	32
RSA-S2 complexity	62+64	94+48	130+30	158+16	174+8	182+4

d from $d_1, d_2, \cdots, d_k$ and a in the RSA-S1 algorithm. Hence, similar to RSA-S1, we require that

$$a^k \geq 2^{64},$$

that is,

$$ck \geq 64.$$

Note that the preceding analysis differs from that given by Matsumoto *et al.* [4], where it is incorrectly assumed that the cryptanalyst will need to find both d_p and d_q to deduce d.

Before proceeding observe that, although the preceding attack relies on the value of the public exponent e being 3, knowledge of d_p can always reveal the factorization of N, regardless of the value of e. This can be seen from the following well-known analysis.

Choose any value m and compute $X = m^{d_p e - 1} - 1 \bmod N$. Then, since $d_p e \equiv 1 \pmod{p-1}$, that is, $(p-1) | d_p e - 1$, we immediately have $p|X$. Given that $d_p \not\equiv d \pmod{q-1}$ (a reasonable assumption since otherwise d_p would be a valid secret key), the probability that $(X, q) = 1$ will be high. The cryptanalyst then will be able to simply derive p by using the Euclidean algorithm, since if $(X, q) = 1$ then $(X, N) = p$.

2) Tabulating Complexities: For the purpose of minimizing the work done by the smart card, suppose that c and k are to be chosen so that the cryptanalyst has 2^{64} possibilities to try. This means choosing c and k so that either $(c-1)k = 126$ or $ck = 64$ (for RSA-S1 and RSA-S2, respectively). For the possible values of c and k that result, Table II shows the computational load for the smart card in computing a single signature using RSA-S1 and RSA-S2 (by Theorems 6.1 and 6.2). For RSA-S1, the number given is the number of modular multiplications mod N. For RSA-S2 the complexity is given in the form $a + b$, where a denotes the number of modular multiplications mod p or q, and b denotes the number of modular reductions from mod N to mod p or q. Note that for RSA-S2 we have ignored the two mod N multiplications and the single mod N addition. From Table II, it should be clear that, for RSA-S2, the best choice from the point of view of smart card computation is $k = 64$ and $c = 1$ (i.e., every f_i is either 0 or 1). For RSA-S1, however, all choices give much the same smart card complexity.

To compare the two algorithms, first we make two observations.

- A modular multiplication mod p or q will take somewhere between one-fourth and one-half of the time required for a modular multiplication mod N.
- The time required to perform a single modular reduction from mod N to mod p or q should be substantially less than (say of the order of 50%) the time required for a single modular multiplication mod p or q.

By examining the table, it should now immediately be evident that RSA-S2 offers superior performance to RSA-S1 for all values of c and k. Before proceeding observe that, in suggesting choices for $a(= 2^c)$ and k, we have so far ignored two aspects of the signature computation, namely,

1) The computational load on the auxiliary processor, that is, the k modular exponentiations required to compute $z_i, 1 \leq i \leq k$.
2) The communications overhead in transferring the values $z_i, 1 \leq i \leq k$, from the auxiliary processor to the smart card.

In both cases the work factor is directly proportional to k. Unfortunately, k is maximized when we attempt to minimize smart card computations for algorithm RSA-S2.

The first problem, that is, the computation of $z_1, z_2, \cdots, z_k$, can be mitigated by careful selection of the values $d_1, d_2, \cdots, d_k$, as discussed subsequently. However, the communications overhead is inescapable, and unless the communications channel between the smart card and the auxiliary processor has sufficient bandwidth, the final choice of a and k will be subject to an appropriate tradeoff between communication time and smart card processing time.

C. Choosing the Values $d_1, d_2, \cdots, d_k$

We now consider what general strategies might be used to choose the values $d_1, d_2, \cdots, d_k, f_1, f_2, \cdots, f_k$, and (for RSA-S2) $g_1, g_2, \cdots, g_k$. We must bear in mind two potentially conflicting constraints.

- We must choose the values $d_1, d_2, \cdots, d_k$ to minimize the work load on the auxiliary processor. One way this might be achieved is by choosing most of these values to be relatively small.
- The probability of collisions (defined subsequently) must be kept sufficiently small so that the secrecy of the RSA secret key is not compromised.

Some specific algorithms for choosing these parameters for both RSA-S1 and RSA-S2 are given subsequently. In each case, the constraints necessary to defeat Gollmann's attack (see Section IV) have been built into these algorithms; that is, that $f_i \equiv 1 \pmod 2$ for every i (for RSA-S1) and that $f_i \equiv g_i + 1 \pmod 2$ for every i (for RSA-S2).

1) Collisions: Before proceeding, we define what we mean by a collision, and in doing so we will show why these two constraints are potentially in conflict. Given a set $d_1, d_2, \cdots, d_k$, a collision is said to occur when two different sets of values f_i yield the same value $\Sigma_{i=1}^k f_i d_i$. Consider the set

$$S = \left\{ \sum_{i=1}^{k} f_i d_i \bmod \lambda(N) : 0 \leq f_i \leq a - 1 \right\}.$$

Depending on the values of $d_1, d_2, \cdots, d_k, S$ will contain at most a^k values.

Now, as has already been mentioned, it is sometimes convenient to choose the values $d_1, d_2, \cdots, d_k$ so that most of them are relatively small. More specifically, suppose that most of the values $d_1, d_2, \cdots, d_k$ are chosen at random from

the range $[0, 2^h - 1]$ for some h. In this case the expected size of S will get closer to a^k as h increases, that is, the expected number of collisions will decrease as h increases. Hence the probability that an "incorrect" set of values f_i will yield the "correct" d will also decrease as h increases. If the probability of such an event is high, then the overall security of the scheme will be reduced, since this will reduce the expected size of the search space for anyone trying to find d using only the specified values of $d_1, d_2, \cdots, d_k$ and a.

Hence, if the values of $d_1, d_2, \cdots, d_k$ are restricted in this way, then either h must be chosen so that the probability of collision is at an acceptably low level, or the values $d_1, d_2, \cdots, d_k$ must be specially chosen to avoid such collisions.

2) Strategy A: For RSA-S1 the simplest strategy is to choose all but one of the values $d_1, d_2, \cdots, d_k$ (d_j say) at random; in addition, the accompanying values of f_i are also chosen randomly from the range $[1, a-1]$ [subject to the constraint that $f_i \equiv 1 \pmod 2$]. The value of f_j is chosen randomly from the set of integers in the range $[1, a-1]$ which are relatively prime to $\lambda(N)$, and d_j is set to whatever value is necessary to ensure that $\Sigma_{i=1}^{k} f_i d_i \equiv d \bmod \lambda(N))$. More formally, this simple strategy for RSA-S1 has the following three steps:

1) Choose the integer j at random from the range $[1, k]$ and choose f_j at random from the range $[1, a-1]$ subject to the condition that f_j be coprime to $\lambda(N)$ (and hence f_j must certainly be odd).
2) For every $i\,(i = 1, 2, \cdots, k, i \neq j)$ choose f_i and d_i at random, subject to the constraints $1 \leq f_i \leq a-1, f_i \equiv 1 \pmod 2$, and $0 \leq d_i \leq N-1$.
3) Set

$$d_j = \left(d - \sum_{\substack{i=1 \\ i \neq j}}^{k} f_i d_i \right) (f_j)^{-1} \bmod \lambda(N).$$

Observe that requiring f_j to be coprime to $\lambda(N)$ ensures that $(f_j)^{-1} \bmod \lambda(N)$ exists.

A corresponding strategy for RSA-S2 involves choosing all but two of $d_1, d_2, \cdots, d_k$ at random, together with their corresponding values of f_i and g_i. More formally, it has the five steps listed below. First, it is necessary to choose an integer M; M must be at least as large as any possible p or q. Typically M will be a little larger than $\sqrt{N}$.

1) Choose the integer u at random from the range $[1, k]$. Then choose f_u at random from the range $[1, a-1]$ subject to the condition that f_u is coprime to $\lambda(N)$ (and hence f_u must certainly be odd). Set $g_u = 0$.
2) Choose the integer v at random from the range $[1, k]$ with the constraint that $v \neq u$. Then choose g_v at random from the range $[1, a-1]$ subject to the condition that g_v is coprime to $\lambda(N)$ (and hence g_v must certainly be odd). Set $f_v = 0$.
3) For every $i\,(i = 1, 2, \cdots, k, i \neq u, i \neq v)$ choose f_i, g_i, and d_i at random, subject to the constraints $0 \leq f_i \leq a-1, 0 \leq g_i \leq a-1, f_i \not\equiv g_i \pmod 2$, and $0 \leq d_i \leq M$.
4) Set

$$d_u = \left(d_p - \sum_{\substack{i=1 \\ i \neq u}}^{k} f_i d_i \right) (f_u)^{-1} \bmod p - 1.$$

5) Set

$$d_v = \left(d_q - \sum_{\substack{i=1 \\ i \neq v}}^{k} g_i d_i \right) (g_v)^{-1} \bmod q - 1.$$

Observe that requiring f_u to be coprime to $p-1$ and g_v to be coprime to $q-1$ ensures that $(f_u)^{-1} \bmod p-1$ and $(g_v)^{-1} \bmod q-1$ exist. Note also that if $c = 1$, that is, $a = 2$, then f_u and g_v will always be 1.

While these simple strategies clearly work, they result in a large work load for the auxiliary processor, which, in the case of RSA-S1, will be required to perform k "full-size" mod N exponentiations, and in the case of RSA-S2 will be required to perform k "half-size" mod N exponentiations (i.e., with exponents of the order of $\sqrt{N}$). If k is large, for example, $k = 64$, then this may be an unacceptably large load, and we therefore consider two alternative strategies that ease the computational load on the auxiliary processor.

3) Strategy B: The first alternative we consider is to choose most of the values $d_1, d_2, \cdots, d_k$ to be small, thereby making the computation of the values $z_1, z_2, \cdots, z_k$ easier. The simplest version of this alternative for RSA-S1 is to choose all but one of the values d_i at random from a restricted range, $[0, r]$ say, and then, as before, to choose the last value to make the sum correct. This involves modifying step 2 of Strategy A for RSA-S1 to the following:

2) for every $i\,(i = 1, 2, \cdots, k, i \neq j)$ choose f_i and d_i at random, subject to the constraints $0 \leq f_i \leq a-1, f_i \equiv 1 \pmod 2$, and $0 \leq d_i \leq r$.

A similar change can be made to Strategy A for RSA-S2, in this case involving a change to step 3:

3) For every $i\,(i = 1, 2, \cdots, k, i \neq u, i \neq v)$ choose f_i, g_i, and d_i at random, subject to the constraints $0 \leq f_i \leq a-1, 0 \leq g_i \leq 1-1, f_i \not\equiv g_i \pmod 2$, and $0 \leq d_i \leq r$.

This approach has problems however—in particular, note that if the range is made too small then the collision probability may be so high that the expected work load required for the cryptanalyst to compute the secret RSA key is reduced significantly, thereby reducing the overall system security.

Strategy C: A more promising approach is the following (which guarantees that the collision probability is minimized). The idea stems from the observation that, since $d_1, d_2, \cdots, d_k$ are public, there is little reason to make them random—it is the values $f_1, f_2, \cdots, f_k$ (and, in the case of RSA-S2, $g_1, g_2, \cdots, g_k$) that need to be unpredictable. Computations for the auxiliary processor will then be simplified if most of the values d_i are multiples of one another.

The simplest form of the idea is as follows. First, d_1 is chosen at random. Next, for i satisfying $2 \leq i \leq k-1, d_i$ is chosen so that $d_i = s_i d_{i-1}$ for some small integer s_i. Finally, d_k is chosen to make the sum add to d.

The preceding scheme has two advantages over Strategy A:

- First (in common with Strategy B), it makes the computation of the values z_i much simpler, since (for $2 \le i \le k-1$) z_i can be computed as

$$z_i \equiv M^{d_i} = M^{s_i d_{i-1}} \equiv (z_{i-1})^{s_i} \pmod{N}.$$

- Second, given suitable choices for the multipliers s_i, collisions can essentially be ruled out—a significant advantage over Strategy B.

Typically, assuming that $a = 2^c$, one could choose d_1 to be any value less than $N_0/2^{ck}$, where $N_0$is the minimum possible value for $\lambda(N)$, and $s_2 = s_3 = \cdots = s_{k-1} = 2^c$. Assuming this choice, Strategy C for RSA-S2 is then as follows (note that the corresponding Strategy C for RSA-S1 should be clear).

1) Choose d_1 at random from the range $2 \le d_1 \le N_0/2^{ck}$ [where, as before, N_0 is the minimum possible value for $\lambda(N)$].
2) For $2 \le i \le k-2$, let $d_i = 2^c d_{i-1}$.
3) For $1 \le i \le k-2$, choose f_i and g_i at random, subject to the constraints $0 \le f_i \le 2^c - 1, 0 \le g_i \le 2^c - 1$, and $f_i \not\equiv g_i \pmod 2$.
4) Choose f_{k-1} at random from the range $[1, 2^c - 1]$ subject to the condition that f_{k-1} is coprime to $\lambda(N)$ (and hence f_{k-1} must certainly be odd). Set $g_{k-1} = 0$.
5) Choose g_k at random from the range $[1, 2^c - 1]$ subject to the condition that g_k is coprime to $\lambda(N)$ (and hence g_k must certainly be odd). Set $f_k = 0$.
6) Set

$$d_{k-1} = \left(d_p - \sum_{i=1}^{k-2} f_i d_i\right)(f_{k-1})^{-1} \bmod p - 1.$$

7) Set

$$d_k = \left(d_q - \sum_{i=1}^{k-2} g_i d_i\right)(g_k)^{-1} \bmod q - 1.$$

5) Other Possible Strategies: At the cost of extra work for the auxiliary processor, hybrid strategies could be produced wherein some of the values d_i are chosen at random (as in Strategy A) and some are chosen either small (Strategy B) or as multiples of other values (Strategy C).

6) Comments and Security Considerations: It should be noted that, in all three of the strategies described earlier, a cryptanalyst equipped with the values $d_1, d_2, \cdots, d_k$ will (at least in theory) have less than $2^{(c-1)k/2+1}$ possibilities to try to discover d for RSA-S1, and less than 2^{ck} possibilities to try to discover d for RSA-S2. We start by considering how this occurs for RSA-S1.

For each of the three strategies there is a "special" value among $f_1, f_2, \cdots, f_k$—f_j say—which must be chosen so that f_j^{-1} exists modulo $\lambda(N)$. However, since the cryptanalyst will not know $\lambda(N)$, and since the cryptanalyst knows that f_i is odd for all i, it is not clear that this information is of any additional value to the cryptanalyst even if j is public.

For RSA-S2 the situation is rather more complex. There are now two special values among $f_1, f_2, \cdots, f_k, g_1, g_2, \cdots, g_k$ —f_u and g_v say—which must be chosen so that f_u^{-1} and g_v^{-1} exist modulo $\lambda(N)$. In addition f_v and g_u must satisfy $f_v = g_u = 0$.

TABLE III
RSA-S2 SMART CARD COMPLEXITY (64 BITS OF SECURITY)

c	1	2	4	8	16	32
k	66	34	17	9	5	3
no. of bits of security	64	65	63	63	63	63
smart card complexity	64+66	100+51	138+32	178+18	218+10	274+6

In Strategy A the cryptanalyst will have no means of knowing the values of u and v, and hence will have only the information that at least one of the values $f_1, f_2, \cdots, f_k$ is odd and at least one is zero (similarly for the values $g_1, g_2, \cdots, g_k$). Given k is not too small, this information again appears to be of relatively little value to the cryptanalyst and does not seem to reduce the effective security of the system. However, for the other two strategies, the values of u and v cannot be hidden, and hence the effective security of the system will be reduced by an order of 2^{c+1}. The easiest way of ameliorating this effect is to increase the value of k by one or two. To illustrate the slightly increased smart card work load involved, Table III presents the smart card complexity for RSA-S2 for various parameter choices giving an effective "64 bits of security" for Strategies B and C (amending the values given in Table II).

Finally, we should note that all the preceding strategies, and in particular Strategy C, may be subject to attacks of a type we have not anticipated here. For example, before any such scheme is used, additional research of the type described in Section VIII needs to be performed.

VII. IMPLEMENTATION CONSIDERATIONS

Other problems associated with the algorithms present themselves when implementation is considered. Of particular concern is how information may leak to the auxiliary processor regarding the values of $f_1, f_2, \cdots, f_k$ (and, in the case of RSA-S2, $g_1, g_2, \cdots, g_k$). First, we must consider how the preceding schemes might operate in practice. We have as yet not discussed how the information computed by the auxiliary processor (i.e., $z_1, z_2, \cdots, z_k$) is processed by the smart card.

One possible approach would be for the auxiliary processor to supply one z_i value at a time, waiting at each stage for the smart card to compute $z_i^{f_i}$ (and, for RSA-S2, $z_i^{g_i}$) and to multiply this value into the partial result. This has the advantage of minimizing the amount of information the smart card needs to store at any one time; this could be essential to the practicality of the protocol since smart cards typically have very limited amounts of onboard memory.

However, this approach also has serious security shortcomings. The amount of computation involved at any stage will depend on the value f_i and (g_i). Thus, by monitoring the interval between requests for values z_i, the auxiliary processor could gain valuable information about the secret values, thereby potentially compromising the value of d itself.

Clearly, the ideal solution would be for the smart card to store all the values $z_1, z_2, \cdots, z_k$ simultaneously. The only information that could possibly be revealed to the auxiliary processor would then be the total duration of the computation, a relatively small amount of information. However, this ap-

proach may not be practical because of the limited memory on the smart card.

An alternative approach would be to require the smart card to apparently spend the same amount of time on each stage of the computation; this could be achieved simply by the addition of "idle time." This would require each stage of the computation to take as long as the maximum possible time for such a computation, a potentially considerable overhead that could potentially reduce throughput by up to 50%. If the smart card is very limited in its memory capacity, this is likely to be the best available solution to this problem; other, more sophisticated schemes, such as the Quisquater–De Soete scheme [6], although they solve this problem, are much less efficient. However, in practice it would appear that typical current smart cards are capable of storing a number of values z_i, and although some idle time may still be necessary, it will be small compared with the total computation time.

Other possible implementation-based attacks include the possibility of the auxiliary processor monitoring the power consumption of the smart card, thereby possibly detecting any smart card idle time that may be present to conceal precise timings from the auxiliary processor. This, together with other potential attacks of this type, requires additional investigation.

VIII. Some Research Problems

Certain questions raised herein lack definitive answers, and the underlying problems need additional investigation. Indeed, final decisions as to the choice of algorithm and parameters require some of these questions to be answered, and such additional work is, therefore, of considerable importance. We conclude this paper by discussing three of these problems in a little more detail.

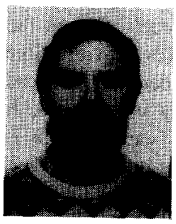
1) The first problem relates to a fundamental assumption of both algorithms discussed herein. That is, releasing some information about the secret exponent d, for example, that it belongs to a set of size 2^{64}, does not compromise the security of the RSA scheme. Of course, if the values $d_1, d_2, \cdots, d_k$ are chosen at random, then the information released regarding the value of d is not manipulated easily.
   However, as discussed earlier, for implementation reasons, it is potentially attractive to choose the values $d_1, d_2, \cdots, d_k$ so that either most of them are relatively small or they have a special form. This leads to the question: If d is known to belong to a small range or is known to be an element of a specified restricted set, how much will this compromise the secret key? In the context of RSA-S2, two obvious possibilities for such a compromise come to mind (similar possibilities exist for RSA-S1).
   - The first possibility is that a generalized version of Wiener's attack [8], briefly discussed in Section II-C, may be possible.
   - The second possibility stems from the observation that, given the public exponent is 3, information can be derived about the possible values of d_p and d_p. The possibility exists that the resulting inequalities could be combined with knowledge of the sets to which d_p and d_q belong to reveal the value of d. We consider the argument for d_p; similar arguments hold for d_q.
     As previously, since $3d_p \equiv 1 \pmod{p-1}$, we have
     $$(p-1)|(3d_p - 1).$$
     By definition, $d_p < p = 1$ and hence $3d_p - 1 < 3(p-1)$; that is, $3d_p - 1$ is either $p-1$ or $2(p-1)$. In the first case, this means that
     $$d_p = p/3$$
     and, in the second case, it means that
     $$d_p = (2p-1)/3.$$
     The first case cannot occur since p is prime, and hence we must have $d_p = (2p-1)/3$ [note that $p \equiv 2 \pmod 3$ is necessary given that the public exponent is 3].
     As before, p and q will normally lie between fixed bounds; typically it might be publicly known that p satisfies
     $$2^{252} + 1 \le p \le 2^{256} - 1.$$
     This immediately translates into a range of values for d_p, that is,
     $$(2^{253} + 1)/3) \le d_p \le 2^{257}/3 - 1.$$
     In fact, the cryptanalyst will be able to deduce tighter bounds on the value of p given knowledge of N. It should be clear that p will lie in the range
     $$\max(2^{252}, N/2^{256}) < p < \min(2^{256}, N/2^{252}),$$
     giving further information regarding the value of d_p.
2) The second problem applies only to algorithm RSA-S2. As a solution to Gollmann's attack, it was proposed in Section IV that $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ should be selected so that $f_i \not\equiv g_i \pmod 2$ for every i $(1 \le i \le k)$. It was stated that this does not appear to reduce the security of the system against cryptanalytic attack. Two things need additional investigation. First, careful checking is required to ensure that the proposed constraint on the selection of $f_1, f_2, \cdots, f_k$ and $g_1, g_2, \cdots, g_k$ really does not reduce security. Second, careful investigation needs to be made into the possibility of producing variants of Gollmann's attack, that is, attacks based on the auxiliary processor providing additional corrupted versions of the values $z_1, z_2, \cdots, z_k$ in the hope of revealing information regarding the value of d.
3) The existence of the divide-and-conquer attack for RSA-S1 (described in section V) raises the possibility that a similar attack may exist for RSA-S2. Additional detailed research is required to see if such an attack can be devised.

ACKNOWLEDGMENT

The authors would like to thank M. Wiener for suggesting a number of extremely important improvements and corrections to the paper, including the attack described in Section V. They would also like to thank S. Lloyd for providing the analysis at the end of Section VI-B-1. The second author would like to thank Hewlett-Packard Laboratories, Bristol, England, for supporting much of the work involved in producing this paper.

REFERENCES

[1] D. Gollmann, private communication, Jan. 1991.
[2] J. Hastad, "On using RSA with low exponent in a public key network," in *Advances in Cryptology—CRYPTO '85 Proc.*, Santa Barbara, CA, 1986, pp. 403–408.
[3] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1981.
[4] T. Matsumoto, K. Kato, and H. Imai, "Speeding up secret computations with insecure auxiliary devices," in *Advances in Cryptology: CRYPTO '88, Proc.*, Santa Barbara, CA, 1990, pp. 497–506.
[5] J-J. Quisquater and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," *Electronics Letters*, vol. 18, pp. 905–907, 1982.
[6] J-J. Quisquater and M. De Soete, "Speeding up smart card RSA computations with insecure coprocessors," in *Smart Card 2000*, D. Chaum, Ed. Amsterdam: North-Holland, 1991, pp. 191–197.
[7] A. Shimbo and S. Kawamura, "Factorization attack on certain server-aided computation protocols for the RSA secret transformation," *Electronics Letters*, vol. 26, pp. 1387–1388, 1990.
[8] M. J. Wiener, "Cryptanalysis of short RSA secret exponents," *IEEE Trans. Information Theory*, vol. IT-36, pp. 553–558, 1990.



John Burns graduated from Birmingham University with a degree in mathematics and applications.

He subsequently worked on various real-time and communications products in Britain and the United States before joining Hewlett-Packard's European Research Laboratories at Bristol, England. Since being there he has worked in the areas of OSI networking, network security, and office products.

Chris J. Mitchell received the B.Sc. and Ph.D. degrees in mathematics from Westfield College, London University, in 1975 and 1979, respectively.

Prior to his appointment as Professor and Head of the Computer Science Department at Royal Holloway, University of London, in March 1990, he was Project Manager in the Networks and Communications Laboratory of Hewlett-Packard Laboratories in Bristol, which he joined in June 1985. Between 1979 and 1985 he was at Racal-Comsec Ltd. (Salisbury, U.K.), latterly as Chief Mathematician. His research interests are in information security, information theory, and combinatorial mathematics.