Moduli for Testing Implementations of the RSA Cryptosystem

Colin D. Walter

Computation Department, U.M.I.S.T.,

PO Box 88, Sackville Street, Manchester M60 1QD, U.K.

www.co.umist.ac.uk

Abstract

Comprehensive testing of any implementation of the RSA cryptosystem requires the use of a number of moduli with specific properties. It is shown how to generate a sufficient variety of these to enable testing which will justify high confidence in the correctness of both the design and the operation of hardware implementations. The tests avoid the necessity of another implementation for comparison. Many of these moduli are also suitable for testing software implementations. Furthermore, the methods apply equally well to other similar modular arithmetic based cryptosystems which use exponentiation, such as Diffie-Helman key exchange.

Key Words: Computer arithmetic, cryptography, RSA modulus, testing, correctness, verification, implementation validation benchmark.

1 Introduction

The RSA cryptosystem [5] is widely used for key exchange and increasingly for the long term storage of sensitive data. A large number of such systems have been designed and built in both software and hardware. In a quest for greater efficiency, software implementations can often be too complex to be fully verified and hardware implementations can have fabrication errors that need to be screened out. Moreover faults can arise during operation in both hardware and software systems for various reasons. A test suite containing a variety of moduli is therefore required to enable development, fabrication and in-service testing of correctness. We show how to build such a test suite. This suite aims to provide exhaustive testing of the main components expected to form the implementation as well as exercising all other components sufficiently to be reasonably certain of their correctness. To our knowledge there is currently no bench mark suite of test values for the correctness of an RSA cryptosystem although plans exist for a correctness validation program in connection with ANSI X9.31-1998 [8]. The discussion here aims to contribute some ideas in this direction. Although most of the discussion is aimed at hardware, many of the observations are directly applicable also to software systems.

Encryption and decryption in RSA are achieved by exponentiating the plain, respectively encrypted, text T to the power of one of two keys e or d modulo a product M of two large primes: $T^e \equiv T \bmod M$ or $T^d \equiv T \bmod M$ respectively. In order to recover the plain text from the obscured text, the two keys d and e must have the property $T^{de} \equiv T \bmod M$, which is guaranteed by selecting them so that $de \equiv 1 \bmod \phi(M)$ for the Euler phi-function ϕ.

The modular exponentiation is achieved by a sequence of modular multiplications which are themselves performed through a repetition of modular additions. Typically, efficient implementations of RSA do not reduce intermediate calculations to yield a least non-negative residue modulo M because this involves full length carry propagation. Instead, the partial results are stored in a redundant form bounded by a small multiple of M. The multiple of M to subtract (or add) in order to keep the result within this range is decided by the first few (the most significant) digits in the traditional algorithm [1] and the last few (the least significant) digits if Montgomery's method [4] is used [2,7]. To check fully the correctness of such implementations, we therefore need at least a set of moduli which include all possible choices for the top few digits and, separately, all possible choices for the bottom few digits. The majority of this paper is concerned with the generation of such moduli.

Intermediate digits in any encryption/decryption will normally all be processed in the same way by a repeated digit slice. So the correctness of this aspect of the implementation's design might reasonably be verified by looking at just one of the most significant or least significant digit slices, and ensuring that all possible inputs are generated for it. The moduli required above for checking the multiple of M for the modular subtraction should also cover this part of testing if sufficiently many end digits are considered.

Fabrication or failure errors can occur at any point in hardware implementations. It may thus be useful to have moduli in which any short subsequence of internal digits can be specified so that specific digit slices can be tested. Hardware generally re-aligns the modulus by a left or right shift. So any digit slice can normally be tested through the most or least significant digits of the modulus just by varying the total number of its digits. We assume that this is the case so that only the end digits and total length of M need to be specifiable. Indeed, by varying the length of M we can check the correctness of such shifting, and the correctness of the associated number of addition cycles.

The third main aspect of implementation is the correctness of the exponentiation algorithm. In a later section we will discuss the adequacy of testing this using the exponents which occur naturally for the moduli which are generated.

Clearly, if the implementation is just a modular multiplier with external access to set any digits, including carries, then a number of single multiplications may suffice for full testing. Otherwise, when internal carries are inaccessible, or only one input T can be presented to the multiplier, it may be only possible to carry out exponentiations in testing. Then, whereas it is possible to verify the exponentiation output against the same computation under a different system, it may be much easier and cheaper to make use of the obvious self-test analogous to $T^{de} \equiv T \bmod M$, namely the property $T^{\phi(M)+1} \equiv T \bmod M$ which is used to determine a suitable d from a given e.

With these three implementation aspects to consider, we therefore initially seek moduli M with the three properties:

- P1) Any short subsequence of end digits can be specified (most or least significant digits) for M of any length;
- P2) $\phi(M)$ can be computed easily; and either
- P3) $T^{\phi(M)} \equiv 1 \bmod M$ holds for a large known set of T, or
- P3') $T^{\phi(M)+1} \equiv T \bmod M$ holds for a large known set of T.

Under the very reasonable assumption that any errors will tend to propagate wildly, the implementation can then be tested by computing $T^{\phi(M)}$ (mod M) or $T^{\phi(M)+1}$ (mod M) and comparing it with the expected result for a sufficient variety of choices for M and T.

In the next section we derive properties which ensure moduli satisfy P3 and also guarantee a straightforward construction to obtain P1 and P2. After performing a construction which enables P1 to hold simultaneously at *both* ends, we look at how to make use of such moduli to form an appropriate test suite, how errors might propagate and how probabilistic arguments might enable fewer moduli to be used than one for every possible sequence of end digits. Some consideration is given to alternative constructions, the time complexities involved, and the implementation errors that may not be covered by such a test suite.

2 The Order Property mod M

Most of the results used in this section can be found in any elementary number theory text such as Chapter 1 of [3]. The classes of residues prime to M form a multiplicative group of order $\phi(M)$. Hence, by Lagrange's theorem (see any book on group theory),

$$T^{\phi(M)} \equiv 1 \bmod M$$

whenever T and M have no common factor. Thus P3 and P3' will hold. Euclid's algorithm can easily determine whether a given T satisfies the co-primality condition. In the case of M being prime, this is just Fermat's Little Theorem, which Euler generalised to

$$T^{\phi(M)+1} \equiv T \bmod M$$

for prime M in order to cover all T, including the case when T shares a factor with M. For general M, by the Chinese Remainder Theorem, the property $T^{\phi(M)+1} \equiv T \bmod M$ holds if, and only if, $T^{\phi(M)+1} \equiv T \bmod Q$ holds for each maximal prime power Q dividing M. Moreover, since ϕ is a multiplicative function, $\phi(Q)$ divides $\phi(M)$ for each such Q. So $T^{\phi(M)+1} \equiv T \bmod Q$ holds if $T^{\phi(Q)f+1} \equiv T \bmod Q$ for every f. When M is square-free, as it is in the RSA cryptosystem, each relevant Q is a prime. Then, by virtue of Fermat's Little Theorem, $T^{\phi(Q)f+1} \equiv T \bmod Q$ holds for every f when T is prime to Q and also, otherwise, because both sides are 0 modulo the prime Q. Thus M being square-free guarantees that $T^{\phi(M)+1} \equiv T \bmod M$ for all T. This proves:

THEOREM 1.
i) Property P3' holds for all T if M is square-free;
ii) The congruences of P3 and P3' hold for any T and M which are relatively prime.

Clearly the congruences of P3 and P3' hold for at least the $\phi(M)$ classes modulo M which are prime to M and this will be a high proportion of all classes if M is not divisible by any small primes. Later we will want to be able to choose T explicitly to be 1, 2 or M–1. As M is generally odd, they will satisfy this theorem.

3 A Construction for Moduli *M*

The key property from which the RSA cryptosystem derives its strength is the difficulty of deriving $\phi(M)$ from M. In effect this requires the prime factorization of M. Hence, if we need to know the value of $\phi(M)$, it follows from the multiplicative nature of ϕ that the best approach to constructing moduli M is via a product of factors for which ϕ can be easily found.

Implementations of the arithmetic for RSA encryption/ decryption rarely make use of any factorization properties of M. Hence we should not feel bound to limit the choice of M to a product of just two large primes. The only exception is where the Chinese Remainder Theorem is used, and we return to this point below (Section 7). Thus there should be no need to generate large primes in order to obtain suitable M; products of small factorisable numbers will normally suffice. However, we may need to exclude some primes from dividing M. The most obvious case is the prime 2: many implementations may assume M is odd because this property holds for the moduli used in the RSA cryptosystem.

Suppose numbers are represented using radix r. This will normally be a small power of 2. It corresponds to the smallest group of bits processed uniformly in the repeated arithmetic operation which performs a modular addition. $r = 2$ or 2^2 is typical for hardware implementations and $r = 2^{32}$ for software implementations which use the built-in machine arithmetic. Suppose also that M has a standard, non-redundant representation, that is, all its digits are in the range $0..r$–1.

Let m be the short sequence of several digits which we wish to have as the *most* significant digits of M. Normally our construction for M will make m a factor of M. So, if m would thereby contribute undesirable factors to M, it must be extended to have one or two more digits which are then chosen to avoid such factors. Further, let k be the number of digits in M other than those which are in m.

A tentative initial proposal for M is to take 0 for each of the digits of M below the required m. Then $M = mr^k$. Although $\phi(M)$ is easy to compute, it will have a high power of 2 (assuming r is a power of 2). Therefore exponentiation by $\phi(M)$ may not fully exercise the non-squaring multiplication hardware, nor allow a sufficient versatility in our later choice of T which are prime to M, nor perhaps even be a legitimate choice for the implementation. So a slightly different choice for M is desirable.

Thus, alternatively, assume M has the form

$$M = mP \quad \text{where } P = \prod_{i=1}^{t}(r^{n_i} \pm 1) \text{ and } k = \sum_{i=1}^{t} n_i$$

The n_i and the signs will be chosen so that the product just exceeds r^k and each factor has a known prime decomposition. The examples section illustrates how the factors of P may be chosen and even varied from the given form in order to obtain specific factorization or congruence properties. Thus $r^{n_i}\pm1$ may be replaced by a set of values around r^{n_i}, if convenient.

We must consider how to choose the signs so that m is indeed the initial sequence of most significant digits. By choosing each n_i sufficiently large, each corresponding factor of P is close to r^{n_i} so that P is close to r^k and M is close to mr^k. The hope is that P should just exceed r^k so that it has a leading digit of 1 followed by a sufficient number of 0s to guarantee that the top digits of M coincide with those of m.

With only + signs in the factors, the product P will exceed r^k whilst with only minus signs P will be less than r^k. Thus the signs can be systematically changed one by one from + to – in order to make P approach r^k more closely, stopping before the product falls below r^k. The 2^t choices give considerable scope for picking P very close to r^k or giving P with other desirable properties. In particular, enough minus signs might be chosen so that changing the sign in a pre-determined factor changes P from more than r^k to less than r^k. Alternatively, so many signs could be changed from + to – that no further sign change keeps P above r^k.

Suppose r^n+1 is one of the factors with a + sign, and assume m has at most n–1 digits. Let P' denote the product given by changing r^n+1 to r^n-1. Assume that this sign change makes P' less than r^k. Then $P' < r^k < P$ and hence

$$\begin{aligned} M = mP &= mP'(r^n+1)/(r^n-1) \\ &< mr^k(r^n+1)/(r^n-1) \\ &= mr^k + 2mr^k/(r^n-1) \\ &\le mr^k + mr^{k+1}/(r^n-1) \\ &< (m+1)r^k \end{aligned}$$

because $2 \le r$ and $mr \le (r^{n-1}-1)r < r^n-1$. But $(m+1)r^k - 1$ is the largest number with leading digit sequence m. Consequently,

THEOREM 2. Take $M = mP$ where P is the product defined above. Suppose n is such that r^n+1 is a factor of P and changing its sign would make P less than r^k. If m has at most n–1 digits then the leading digits of M are those of m.

In practice, the looseness of these inequalities for the factor r^n+1 means that more often than not m can have n or even more digits, all of which will appear at the start of M. Indeed, to obtain more available digits for m, the signs of as many factors as possible could be changed.

Numbers of the form $r^n\pm1$ include the Fermat and Mersenne primes (when r is 2). Their factorization has been widely studied. Clearly by algebra $r^f\pm1$ is a factor of $r^n\pm1$ when the signs are the same, f divides n, and n/f is odd if the shared sign is +. Also by algebra, r^f+1 is a factor of r^n-1 when f divides n and n/f is even. Since no exponent n_i is required to be very large, it is reasonable to assume that all the factors of P have known prime decompositions, so that $\phi(M)$ can be calculated easily. Furthermore, with care it may be possible to make M square free by noting that the highest common factor between $r^n\pm1$ and $r^{n'}\pm1$ is 1, 2, $r^g\pm1$ or $2(r^g\pm1)$ where g is the greatest common divisor of n and n'. (This is derived by repeatedly noting that the highest common factor divides the sum or difference of such numbers, but not the power of r which appears in that sum or difference.) Thus an obvious choice for the n_i is to take them pairwise coprime.

As examples below illustrate, the choice of factors $r^{n_i}\pm1$ can be varied. Each factor could offer a choice between any two factorisable numbers where one is just above r^{n_i} and the other just less than it. The closeness to r^{n_i} in enough cases will make P near enough to r^k to obtain m in the leading digits of M. As before, from an initial choice of the larger factor in each case, the factors are replaced by the smaller alternative, stopping while P is still greater than r^k or when changing a selected factor would make P less than r^k.

Any of these constructions enables P2 to be satisfied, as well as P1 where the specified digit subsequence m appears initially. Moreover, the product P need only be computed once and then used for all choices of the initial, most significant digits m.

4 Least Significant Digits for M

Although setting the lowest digits to a given sequence l can be achieved simply by taking $M = l$, there are situations where a modulus of more than these few digits is desired. In fact, the previous construction provides $l = m$ as the digit sequence at the least significant end of M under suitable conditions. Assume that there are an even number of minus signs in the product P, that l has n digits and that $n_i \geq n$ for each i. Then, viewing P as a polynomial in r, the lowest non-constant power of r in P has exponent at least n. So its lowest n digits except the last are all 0 and the last digit is 1. This means $M = lP$ will have l as a digit subsequence at its least significant end.

One objective is to construct M with m at the most significant end and l at the least significant end. So suppose M has the form

$$M = mr^nP + lP'$$

where n is the number of digits in l. P is constructed as in the previous section so that mr^nP provides m in the right place for the most significant digits of M. Of course, its n least significant digits are all 0. P' will be constructed as in the previous paragraph to make lP' provide l for the least significant digits, but not quite large enough to prevent the digit sequence m from being at the most significant end of M. We will also choose P and P' to share most of their factors so that $\phi(M)$ can be easily found. This is done by starting with $P' = P$. If this choice of P' is too big, factors are removed one by one until the remaining product P' is small enough not to affect the topmost digit sequence m. The lowest digit of P' as a polynomial in r is then ±1. So one more factor (with a $-$) is removed if necessary to produce +1 as the lowest digit. This resulting P' can be further adjusted with other factors if desired, providing the main properties are retained. In consequence, $P = pR$ and $P' = p'R$ for some small p and p' where R is the product of the factors common to P and P' which remain after modifying P to P'. This yields

$$M = (mr^np+lp')R$$

which is easily factored because mr^np+lp' is relatively small. So $\phi(M)$ can be evaluated. Moreover, the wide choice for all the parameters enables the factor mr^np+lp' to avoid any undesired divisibility properties. In particular, by construction p' is already prime to r so that M will share divisors with r only as far as is necessary to obtain l at the least significant end.

5 Examples

Take the radix to be $r = 4$. If we seek a square-free M then it is reasonable to replace each $r^n\pm1$ by a pair of numbers on either side of r^n which are prime to all numbers earlier in the list. Thus, for n = 1,2,3,... the list of selected pairs might be chosen as { (3, 5), (13, 17), (61, 67), (253, 257), (1021, 1031), (4093, 4099), (16381, 16387), (65531, 65537), ... }. As well as (4093, 4099) we might also include the pair (4091, 4111). In this way several factors can be close to the same r^n without losing the square-free property by repeating the same factor.

Suppose M must allow any two initial digits. These yield a number in the range 4 to 15. To maintain the square-free property the initial sequence m requires a third digit, chosen to make it square free whatever the choice of the first two digits. This is always possible: thus the square $4_{10} = 10_4$ at the head of M is given by taking m as $17_{10} = 101_4$ or $19_{10} = 103_4$ and the square $9_{10} = 21_4$ is given by taking l as $37_{10} = 211_4$, $38_{10} = 212_4$ or $39_{10} = 213_4$.

These choices for m may have common factors with (3,5) or (13,17), which are therefore deleted from the list of numbers used for P. Similarly, the choice of m should avoid the 7 dividing 16387 and the 19 dividing 65531 if either is used in P.

Suppose M is to have 20 digits (in radix 4). Then P must be chosen to be just over 4^{17} since m uses three digits. The partition 17 = 8+5+4 leads to P having three factors close to 4^8, 4^5 and 4^4 respectively. One acceptable choice is

$$P = 65531_{10} \times 1021_{10} \times 257_{10} = 100000322033230033_4$$

Here the 257 cannot be replaced by 253 without making P less than 4^{17} since the other factors are already the lower of the two choices from the list. Hence, by Theorem 2 applied to $257 = 4^4+1$, we know that $M = mP$ will provide m for its three initial digits. Indeed, the five 0s after the initial 1 in P ensure that M could provide up to five given initial digits. M is square-free providing m is chosen as above and is not divisible by 19. Then $\phi(M) = \phi(m \times 19 \times 3449 \times 1021 \times 257) = \phi(m) \times 18 \times 3448 \times 1020 \times 256$.

Now consider adding the requirement on M for two arbitrary final digits l. The lowest digits of P' must be 01_4 to achieve l. So a different choice of factors is required in P and P' than above. For convenience, we choose factors $\equiv 1 \bmod 4^2$ so that the same is true of the product. To achieve this the list of factors to choose from might be modified to { ..., (16369, 16417), (65521, 65537), ... }. The form of M is $M = mr^2P + lP'$ where P must now be just above 4^{15} so that mr^2P has 20 digits. Using 15 = 7+8 yields

$$P = 65521_{10} \times 16417_{10} = 1000013103320101_4$$

which again allows m to appear as the leading digit sequence for M. Taking $P' = P$ gives

$$M = (16m+l) \times 65521 \times 16417$$

which produces l in its lowest two digits. However, P' may require further adjustment because it may now be large enough for the term lP' to affect the two leading digits of M, which we want to be given by m. By considering only the first 5 digits of P and P' and ignoring any lower digits, the first 5 digits of M would be given by $16m+l$. Rounding up the rest of P and P', namely $1310..._4$ up to $2000..._4$, we observe that the rest of P and P' contribute at most $2(16m+l)$ div 4^5 to the top 5 digits of M. This quantity is at most 1, and so could affect the 5th digit, propagating a carry. Then, only for $l = 33_4$ might a carry (at most 1) propagate to the third top digit. This position contains the third, chooseable digit of m. It is therefore first selected to be at most 2_4 in order to absorb any carry and, secondarily, to make $16m+l$ square-free if possible. So the top 2 digits of m really are the top digits of M, as required. Again $\phi(M)$ is easily calculated as 65521 and 16417 are both prime.

6 The Test Suite

In this section we wish to consider what pairs of values (M,T) would form an appropriate set for testing the design, fabrication and run-time correctness of a modular exponentiator, where the Ms have been constructed as in the previous sections. Although testing is phrased in terms of hardware, some aspects apply equally to the testing of software implementations.

From the published literature it is quite clear that implementations of modular exponentiation vary widely in design. The suggestions here may therefore need adapting or extending to particular applications. Section 1 outlined the major components most likely to be present and in need of testing: a digit slice which is repeated to form the modular addition cycle, a module receiving end digits (either msd or lsd) which decides the modulus multiple for subtraction, and the component for controlling the squares and multiplies of the exponentiation. In addition there are various counters determining, for example, how many shifts are given to inputs and outputs, and how many addition and multiplication cycles are performed.

The requirement for speed generally means that most operations are entirely locally defined (i.e. within a digit slice or equivalent), so that different components might be fully testable individually. In a well designed system, the main global operation is usually only a final modular reduction and carry propagation step to obtain a non-redundant output within the required range of $[0,M-1]$.

This section concentrates on the testing of those components which will benefit from the above construction for moduli although for completeness some remarks are also made about other components.

The expected algorithm has the following format. For any non trivial text T, if M has n bits, then the computation of $T^{\phi(M)+1}$ mod M will require around $1.5n$ multiplications and $O(n^2)$ addition cycles. This requires two or more registers which hold partial products and powers of T as well as the register containing M. Since $e = 17$ is a typical encryption key for RSA, after only a very small number of multiplication/ squaring cycles, the initial text should have been transformed into a number which can be assumed to be a randomly distributed bit sequence. This fact enables one to compute probabilities for various situations with a high degree of accuracy.

The first component for consideration is the digit slice, whose design can be tested by exhaustively checking all combinations of input digits against a formal specification. However, such testing of individual slices in a fabricated chip is generally impossible since inputs such as carries are not usually accessible for manipulating. The fabrication faults which need to be checked for can exhibit widely differing characteristics and need a range of inputs to detect.

Every input bit for M and T needs to be tested in both positions in case a register bit is stuck in one value. Pick a value for M which is of maximal length. Then choose two values for T, one to be any value less than M but with the same length as M and the other its complement. Raising these Ts to the power 1 and checking the output is still T should test the T input and result output registers for stuck bits.

To test the register M, construct a small set of moduli with known ϕ values such that for each bit position there is a pair of moduli representing both values for the bit. (This can be done just by taking M as a product of numbers with known prime decompositions, i.e. m=1 and l=0.) Suppose the hardware behaves as if M' had been loaded instead of M. If $M' \neq M$ then the expected value for $\phi(M)$ is very unlikely to match the value $\phi(M')$ needed for the exponentiation test to work. Hence, using any non-trivial T satisfying Theorem 1, evaluating $T^{\phi(M)}$ mod M or $T^{\phi(M)+1}$ mod M will generally yield an unexpected value if register M or its I/O is not operating properly: to obtain the expected value of 1 or T the supplied value of $\phi(M)$ would have to be a multiple of the order of T modulo M', which is unlikely. (The likelihood of a false negative is certainly at most the inverse of the average order of an element in the multiplicative group of residue classes of a modulus the size of M.)

In the addition circuitry which forms the main body of the digit slice, a single error at any point tends to have an effect equivalent to adding a bit to, or subtracting a bit from, the output whenever some input condition is satisfied. Due to carry propagation, several result bits may then be affected. Over the large number of additions required for a single modular multiplication, the randomness of the bits ensures that this condition for the error occurring is most likely to be satisfied many times. So most multiplications will be affected. In the multiplication circuitry, a bit error is usually copied to many or all digit slices so that a large number of output bits are affected. It follows then that, as exponentiation involves all bits in multiplications, almost all errors after the input registers should have a catastrophic effect on the output. Thus the tests for register M in the previous paragraph will normally reveal any such errors in the digit slices.

Simulations of errors in a digit slice can easily be run, and thereby good approximations obtained for the probability of an addition or multiplication cycle displaying the error. These can reveal that some errors may not be caught with the above tests. In particular, the inclusion of table driven digit multiplications or carry lookahead circuits can lead to some errors arising only when particular bit combinations arise for the associated digit of M. Because adjacent digit slices interact through carries, it may be necessary to determine sequences of several digits of M in order to generate every situation. The number of such digits can be discovered by investigating the particular implementation. Thus the testing of some implementations will need a collection of moduli M which provides some or all digit sequences of a given length at a particular position. Such moduli are constructed as described in previous sections: the length of M is varied over all allowable choices, and one of m or l is fixed while the other varies over all the necessary sequences. This usually generates a sufficient variety of moduli because the hardware will shift M of a particular length to align the variable m or l with the digit slice to be tested. Then, with the same argument as above, using one non-trivial T with each such M should normally expose any error. Of course, when M becomes short so does $\phi(M)$, so that more values of T may need to be considered in order to maintain any certainty that at least some combinations which expose the error have arisen.

Note that the shifting of M implies that short moduli will not use the digit slice containing the error, but longer ones will. Thus outputs will normally suddenly change from being correct to being incorrect as the length of M is increased past the point of the fault. This enables the general position of the fault to be detected accurately.

Testing the component which decides which multiple of the modulus to subtract (or add) is done in much the same way as testing an end digit slice. The multiple depends on the top (or bottom) several digits of both M and the current partial product. Thus for full testing, m or l, as appropriate, must vary over all values using sufficiently many digits to ensure all possibilities arise. The other of l or m can be kept fixed. Again a single T will normally suffice for each M since its end digits will vary randomly over the many calls to the component during each exponentiation.

Testing the exponentiation algorithm is normally a case of looking at boundary values, such as the maximum and minimum possible exponents, as well as checking that all possible bit combinations arise to exercise fully any bit recoding process which controls the scheduling of multiplications and squarings. This algorithm is normally independent of both the text T and the modulus M so that it would suffice to fix both of these. If it is desired to use $\phi(M)$ or $\phi(M)+1$ as the exponent to obtain self-checkable output, then it may be possible to generate a suitable M after fixing $\phi(M)$. We won't supply details, but the basis of a method would be using the prime decomposition of $\phi(M)$ and the fact that ϕ is a *multiplicative* function (i.e. $\phi(xy) = \phi(x)\phi(y)$ if x and y are coprime). For bit recoding of exponents, the tests already described above may suffice: the values of $\phi(M)$ used should make this clear. The boundary cases of very small exponents can be checked easily by taking $T = 2$, say, and observing what power of T is output for the exponents 0, 1, 2 etc. At the opposite extreme, test cases for maximal exponents should be easy to generate although this seems to require an independent implementation to calculate the expected output – one of the things that has been avoided in the other tests so far. The variety of exponentiation algorithms makes it inappropriate to be more specific. Generally speaking, tests can be developed independently of the choice of M, and so is not the concern here.

The final main component tackles carry propagation and reduction to the range $[0,M-1]$. Input to the carry propagation unit will be the required result in a redundant form. Thus if $T = 2^h < M$ is the expected output of a non-trivial exponentiation, there is likely to be propagation of carries up most of the register to position h. For a full length M, $T = 2^h$ should be used to compute $T^{\phi(M)+1}$ mod M for as large an h as possible as well as for several of the smallest possible hs.

For the reduction to an output in the range $[0,M-1]$ there may be extra subtractions to perform and signs to determine. The obvious test cases are evaluating $T^{\phi(M)+1}$ mod M with texts T close to the boundary points, say T = 0, 1, 2 and $M-1$ for which the wrong multiple of M may be added. If the pre-normalisation bound on the output from the exponentiator is cM (c is often about r) then, before the subtraction, the output should have the form $iM+T$ for some i in $[0,c-1]$. So several different moduli M will need to be used to ensure each possible i arises during the test. These M should be of maximum length to test fully the subtraction and sign determination.

In summary the following pairs (M,T) might constitute a suitable test suite for most implementations of modular exponentiation:

i) For M of maximal length, choose $T < M$ with the same length as M and its complement T'. Raise these to the power 1 and check the output equals the input.

ii) Take a small set of moduli with known ϕ values such that for each bit position there is a pair of moduli representing both values for the bit. For each modulus choose T prime to M with $1<T<M$ and check $T^{\phi(M)}$ mod M yields 1.

iii) With the length of m and l determined as above take a set which includes a modulus M of each possible length constructed as in section 3 or 4 with each possible value for m or l, according as the test for modulus subtraction is performed at the most or least significant end respectively.
For each such M choose T prime to M with $1<T<M$ and check $T^{\phi(M)}$ mod M yields 1.

iv) For an odd modulus M of maximum length and $T = 2$, check the output of T^i mod M is really T^i for $i = 0,1,2,...$ such that $T^i < M$.

v) For a full length odd M with known $\phi(M)$ and each $T = 2^{2^h} < M$ check that $T^{\phi(M)+1}$ mod M yields T.

vi) For a maximum length odd M of known ϕ value and texts T = 0, 1, 2 and $M-1$, check $T^{\phi(M)+1}$ mod M returns T.

If we assume the moduli have already been generated, the number of different m and l is a constant and the maximum M for a specific implementation has n digits, then all the above tests take an effort of at most $O(n)$ times the time for exponentiating with an exponent of n digits.

Finally, observe that these tests can be used to determine the maximum clock speed at which a specific chip will perform correctly because each part of the circuitry is tested under all possible input conditions.

7 RSA using the CRT

Some implementations of RSA (e.g. [6]) make use of the Chinese Remainder Theorem (CRT) and the factorization of $M = pq$ as a product of two primes to split the exponentiation T^d mod M into two smaller, parallel, shorter exponentiations modulo the factors:

$$\begin{aligned} T^d \bmod M = (\ &(T^{d \bmod \phi(p)} \bmod p)(q^{-1} \bmod p)q \\ &+ (T^{d \bmod \phi(q)} \bmod q)(p^{-1} \bmod q)p\ ) \bmod M \end{aligned}$$

The properties used here are the primality and coprimality of the factors. The first enables $\phi(p)$ to be computed as $p-1$ and the second guarantees the existence of q^{-1} mod p, which can be found via the Euclidean algorithm. Similar computations yield the second term. So knowledge of the factorisation of M roughly halves the length of all the numbers, including the exponents, and so cuts the work required by a factor of 4.

We will assume that the modular reduction of d is easy to check and that, by picking $d = 1$, a sufficient variety of values for T, p and q can be chosen to test thoroughly all the other operations apart from the exponentiations.

If the hardware allows $\phi(p)$ or $d \bmod \phi(p)$ to be computed independently of p, then testing the first exponentiation need not require p to be prime. So the previous methods can be used. Thus p can be chosen to have the form mP for product P and leading digits m, and any q prime to p with known $\phi(q)$ can be used to complete the definition of M. Otherwise, if p needs to be prime because $\phi(p)$ is always assumed equal to $p-1$, then we can use $p = mr^n+1$ as a seed for a prime number generator which will keep incrementing p until a prime is found. Then, unless n is very small, p will have the leading digit sequence m which is what we wish for the test suite. Interchanging the roles of p and q will test hardware which performs the other of the two exponentiations.

8 Conclusion

The aim of this investigation has been to develop a suite of moduli M and texts T to test thoroughly the large number hardware exponentiators used in the RSA cryptosystem [5] and Diffie-Helman key exchange [9].

Normally, with randomly generated test cases, an equivalent amount of external calculation needs to be performed to verify any results. Such calculations may have to be done on much slower implementations or using a system which may have the same design errors as the hardware being tested.

However, the tests here are self checking: the output should always be the same as the input or equal to 1, depending on the test chosen. So the large quantity of computation necessary to guarantee that even the rarest fault has been exhibited is performed without having to be matched by equivalent external work.

The effort involved is mainly in the construction of moduli via a small number of products P with known factorization properties. This work can be done entirely independently of any algorithms which may be implemented in the hardware being tested.

The result is a suitably large set of moduli for testing most implementations of the RSA cryptosystem. It contains moduli of every necessary length with any given initial and terminal sequences of digits. In most cases, this should enable the full functional verification of an implementation at any stage in its life cycle.

Acknowledgement. The author would like to thank the anonymous referees for several useful comments to improve the initial version of this paper.

References

[1] E. F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two-Key Cryptography," in *Advances in Cryptology - CRYPTO '82*, Chaum et al., Eds., New York, Plenum, 1983, pp. 51-60.

[2] S. E. Eldridge, "A Faster Modular Multiplication Algorithm", *Intern. J. Computer Math.*, vol. **40**, 1991, pp. 63-68.

[3] N. Koblitz, *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics, vol. **114**, Springer-Verlag, 1987.

[4] P. L. Montgomery, "Modular Multiplication without Trial Division", *Math. Computation*, vol. **44**,1985, pp. 519-521.

[5] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, vol. **21**, 1978, pp. 120-126.

[6] M. Shand, P. Bertin & J. Vuillemin, "Hardware Speedups in Long Integer Multiplication", *ACM Computer Architecture News*, vol. **19**, March 1991, pp. 106-113.

[7] C. D. Walter & S. E. Eldridge, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Trans. Comp.*, vol. **42**, 1993, pp. 693-699.

[8] ANSI X9.31-1998, *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry*, American National Standards Institute, New York, USA, 1998.

[9] W. Diffie & M. E. Hellman, "New Directions in Cryptography", *IEEE Trans Inform Theory*, vol. **22**, 1976, pp. 644-654.