Batch RSA*

Amos Fiat†
Department of Computer Science
Tel-Aviv University
Tel-Aviv, Israel

April 16, 1996

Abstract

We present a variant of the RSA algorithm called Batch RSA with two important properties:

- The cost per private operation is exponentially smaller than other number theoretic schemes ([9, 23, 22, 11, 13, 12], etc.). In practice, the new variant effectively performs several modular exponentiations at the cost of a single modular exponentiation. This leads to a very fast RSA-like scheme whenever RSA is to be performed at some central site or when pure-RSA encryption (*vs.* hybrid encryption) is to be performed.
- An additional important feature of Batch RSA is the possibility of using a distributed Batch RSA process that isolates the private key from the system, irrespective of the size of the system, the number of sites, or the number of private operations that need be performed.

1 Introduction

Almost all number-theoretic cryptographic schemes in use today involve modular multiplication modulo a composite or prime. The private key used must be sufficiently long so as to ensure that an exhaustive search through the key space is impractical. As a consequence, the work performed by such algorithms during the private operation is large.

In this paper we introduce a new concept of amortized private operations. This leads to an exponential improvement in performance. The number of secret bits in the private key and the number of operations required for a private operation is amortized over many individual private operations. Amortization is common in other areas of computer science [4, 25].

The main result in this paper is to obtain fast public *and* fast amortized private operations. For a security parameter k (breaking the scheme should take time $\exp(k)$) we achieve an amortized $\mathrm{polylog}(k)$ multiplications per private operation and use up an amortized $\mathrm{polylog}(k)$ secret bits of the private key per private operation. Clearly, this is only possible in an amortized setting because a private key that had only $\mathrm{polylog}(k)$ secret bits would contradict the assumption that breaking the scheme requires time $\exp(k)$.

---

*A preliminary version of this paper appeared in Advances in Cryptology: Proceedings of Crypto '89, pp. 175–185.

†Work performed at UC Berkeley and ARL, Israel

In various modes of operation the scheme "seems" to be standard RSA from the point of view of the end-user (thus allowing the use of standardized RSA packages), while processing at the center is different than standard RSA.

To justify the practicality of our results, we note that the real problem with performance does not seem to lie in a distributed setting but rather with centralized applications. Todays microprocessors can perform hundreds of modular multiplications in a few seconds. Special purpose hardware can improve this by one to two orders of magnitude. Large central mainframes may be considerably faster, yet much less cost-effective with respect to processing power.

Many applications require a centralized setting. Several suggested applications of digital signatures are almost irrelevant without a large central clearinghouse, and such a clearinghouse may be required to generate digitally signed receipts in response to transactions. Another typical application is a mainframe that has to decrypt many transactions, (financial data, session initiation key exchange, etc.). The scheme presented here is particularly suitable for such centralized applications.

The underlying idea behind our new scheme is to batch transactions. Rather than perform one full-scale modular exponentiation per digital signature as with RSA, the scheme performs one full-size exponentiation and subsequently generates several independent digital signatures.

Given an n bit modulus, our scheme requires $O(\log^2 n)$ multiplications for a batch size of $n/(\log^2 n)$ messages. We also require up to two modular divisions per signature/decryption — this is a low order term and can be ignored.[1] Clearly one must optimize the batch size for a specific modulus size, and one can obtain *better* results for smaller batches if the modulus is (relatively) small.

Generally, we have a tradeoff between the batch size b and the number of multiplications per signature. Let cn denote the number of modular multiplications required for an n bit exponentiation ($c \approx 1.5$). Given a batch of b messages, $b < n$, we can generate b digital signatures at a cost of $cn/b + O(\log^2 b)$ multiplications per signature (plus two modular divisions). For a fixed batch size the work required to generate *all* signatures is asymptotically equal to the work required for *one* RSA signature.

Rather than perform one full-size exponentiation to decrypt an RSA-encrypted block, the new scheme performs one such exponentiation and subsequently decrypts several RSA-encrypted blocks. This is relevant in the context of mainframe decryption (hybrid scheme or pure) and in the context of pure-RSA decryption generally. With respect to a pure RSA encryption scheme, this simply means that the block size is some multiple of the RSA modulus size. We have a tradeoff between block size and time, for larger blocks we spend less time overall.

Another application of the methods presented here is to generate Shamir's cryptographically secure pseudo-random sequence [S] with the same gain in performance. In this context, the block size penalty mentioned above does not occur. It is noteworthy that Shamir himself considered his scheme in [S] impractical due to the great number of multiplications required.

Even if we completely ignore the issue of performance and use full-sized encryption exponents, one important point concerning Batch RSA is that only one root need be extracted, irrespective of the batch size.

Many private operations can be performed by performing *one* private operation. The preliminary work to merge the batch into one problem involves no secret data, neither does the split-up phase after the root extraction.

[1]Modular division is equivalent to multiplication for quadratic algorithms ($O(n^2)$ bit operations — e.g., [K] section 4.5.2 problem 35) and equivalent to $O(\log n)$ multiplications asymptotically (*I.e.*, $O(n \log^2 n \log\log n)$ bit operations — [AHU] section 8.10).

Imagine a scenario where many link encryptors are to decrypt RSA encrypted data. Rather than store the private key at each and every such encryptor, the private key can be stored at one secure site. This could be a relatively weak processor, such as a smartcard on the CEO's desk, irrespective of the number of private operations to be performed.

The merge and split up phases can be performed in a distributed setting, and an important feature of the scheme is that the communications required to transmit a "merged" sub-batch to the next merge phase is equal to n bits, the length of the modulus, irrespective of the number of private operations in the batch. Similarly, the communications required to transmit a "merged" response to the next split phase is only n bits, irrespective of the number of responses this value represents. This makes it very efficient to perform such a process in a large distributed setting.

Now, the merge phase and the break-up phase involve no secret keys, but if an eavesdropper is listening to the communications he can obtain the decrypted data from the split up phase. One way to ensure that this does not happen is for the encryptor to multiply a random factor of the form r^e to the real value to be decrypted c, the value $c \cdot r^e$ is then input to the distributed batch RSA process, and the result, $c^{1/e} \cdot r$ is obtained. The encryptor may now divide out by r and obtains the cleartext $c^{1/e}$.

2 Background and Central Observation

An RSA digital signature to a message M is simply the e'th root of M modulo N. The public key is the pair (N, e) whereas the private key is the prime factorization of N, e is chosen to be relatively prime to Euler's totient function ϕ of the public key modulus N.

To generate a digital signature on M one first computes $d = e^{-1} \pmod{\phi(N)}$ and then computes

$$M^d \pmod{N} = M^{1/e} \pmod{N}.$$

Thus, every digital signature consists of one full-sized modular exponentiation. ([QC] suggest the use of the Chinese remainder theorem so digital signature generation is slightly faster).

Fundamental to getting polylog(n) multiplications per private operation is the use of (relatively) small encryption exponents for RSA. Using a small encryption exponent means choosing e to be some small constant (say 3), and generating the public key N so that $\phi(N)$ is relatively prime to e. However, choosing a small encryption exponent says nothing about the decryption exponent d. Generally, d will be $\Omega(\phi(N))$. In fact, if d were too small (less than exponential in the security parameter), it would allow the cryptanalyst to attack the scheme. In some sense, we attain the effect of a very small d (length polylog in the security parameter), without compromising security. Weiner [26] gives attacks on short secret RSA exponents.

Our RSA variant grants some leeway in the value of e. For example, choose two parameters S and R so that S and $R - S$ are small (e.g., $S = n^c$, $R = S + n$.). A public key N is chosen so that $\phi(N)$ is indivisible by all primes in the range $S, \ldots, R$. A valid digital signature is of the form $(s, M^{1/s} \bmod n)$, where s is *any* prime in the range $S, \ldots, R$.

To motivate this variant consider the following example:

Example 2.1:

Given two messages $0 < M_1, M_2 < N$, we wish to compute the two digital signatures $M_1^{1/3} \pmod{N}$ and $M_2^{1/5} \pmod{N}$.

Let

$$\begin{aligned} M &= M_1^5 \cdot M_2^3 \pmod{N}, \\ I &= M^{1/15} \pmod{N}. \end{aligned}$$

Now, we can solve for $M_1^{1/3}$, $M_2^{1/5}$ as follows:

$$\begin{aligned} \frac{I^6}{M_1^2 \cdot M_2} &= M_2^{1/5} \pmod{N}; \\ \frac{I}{M_2^{1/5}} &= M_1^{1/3} \pmod{N}. \end{aligned}$$

Note that we require *one* full-sized exponentiation to compute $I = M^{1/15} \pmod{N}$ and a constant number of modular multiplications/divisions for preprocessing and to extract the two digital signatures. The rest of this paper is devoted to the generalization of example 2.1. The reason that we were able to perform this "magic" trick is because the greatest common divisor of 3 and 5 is 1.

3 Batch RSA

As above, let N be the RSA modulus, $n = \log_2(N)$, and let b be the batch size.

Let $e_1, e_2, \ldots, e_b$ be b different encryption exponents, relatively prime to $\phi(N)$ and to each other. Choosing encryption exponents polynomial in n implies that their product, $E = \prod_{i=1}^{b} e_i$, is $O(b \log n)$ bits long. Choosing the encryption exponents as the first b odd primes gives us $\log(E) = O(b \log b)$.

Given messages m_1, m_2, ..., m_b, our goal is to generate the b roots (digital signatures/decryptions):

$$m_1^{1/e_1} \pmod{N}, \quad m_2^{1/e_2} \pmod{N}, \ldots, \quad m_b^{1/e_b} \pmod{N}.$$

Let T be a binary tree with leaves labeled $e_1, e_2, \ldots, e_b$. Let d_i denote the depth of the leaf labeled e_i, T should be constructed so that $W = \sum_{i=1}^{b} d_i \log e_i$ is minimized — similar to the Huffman code tree construction. For our main result of $O(\log^2 n)$ multiplications per RSA operation we could simply assume that T is a full binary tree, asymptotically it makes no difference. In practice, there is some advantage in using a tree that minimizes the sum of weight times path length because the work performed is proportional to the sum W above.

Note that $W = O(\log b \log E)$. We will show that the number of multiplications required to compute the b roots above is $O(W + \log N)$.

Our first goal is to generate the product

$$M = m_1^{E/e_1} \cdot m_2^{E/e_2} \cdots m_b^{E/e_b} \pmod{N}.$$

We now show that this requires $O(W)$ multiplications.

Use the binary tree T as a guide, working from the leaves to the root. At every internal node, take the recursive result from the left branch (L), raise it to the power E_R where E_R is the product of the labels associated with leaves on the right branch. Similarly, take the result from the right branch (R) and raise it to the power E_L which is the product of the labels on the left branch. Save the intermediate results L^{E_R} and R^{E_L} (required later). The result associated with this node is $L^{E_R} \cdot R^{E_L}$. The product

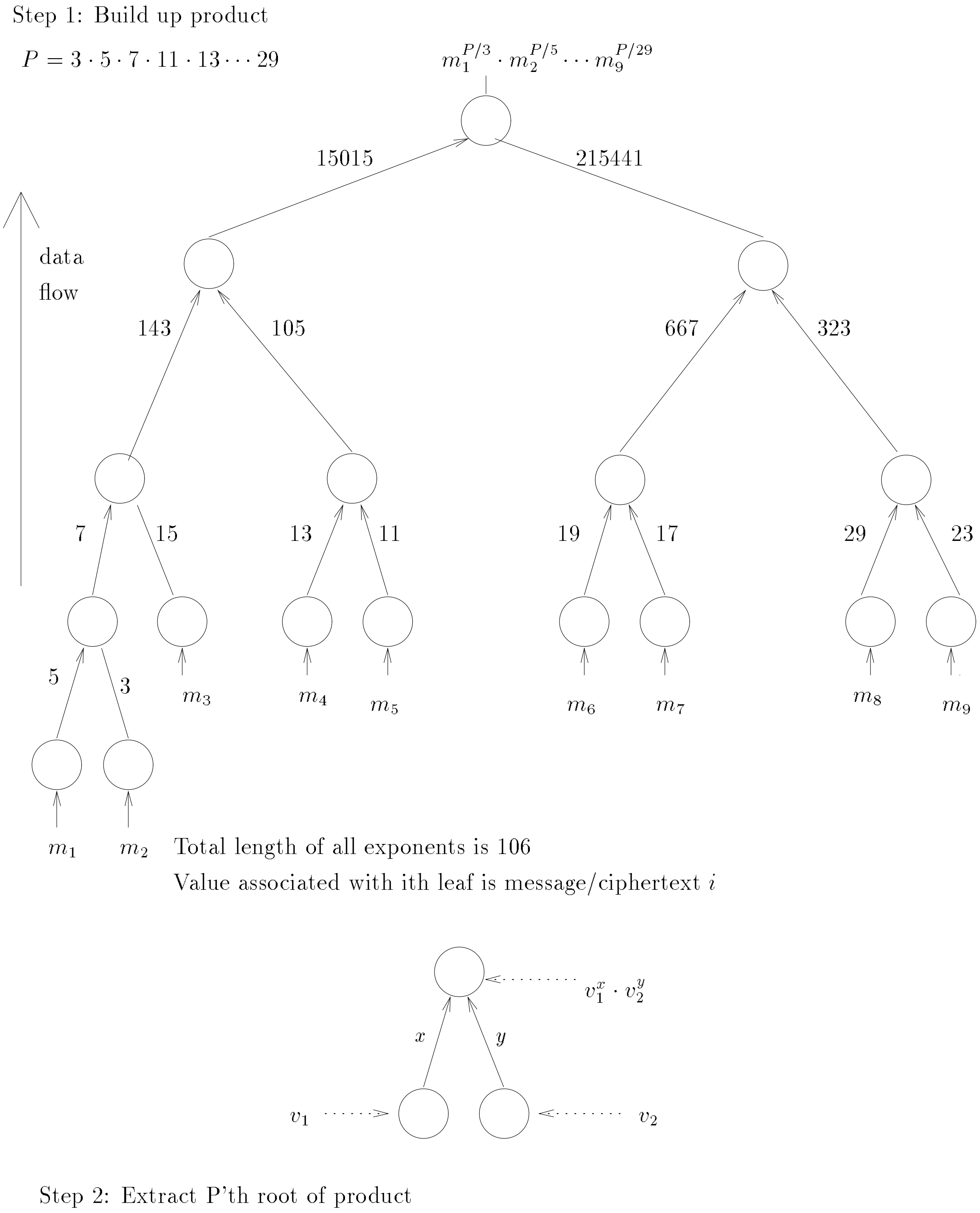

Figure 1: Build up Product and Extract Root

M is simply the result associated with the root. (See figure 1, the ith leaf is labeled with the ith odd prime).

We now extract the Eth root of the product M:

$$M^{1/E} = m_1^{1/e_1} \cdot m_2^{1/e_2} \cdots m_b^{1/e_b} \pmod N.$$

This involves $O(\log N)$ modular multiplications — equivalent to one RSA decryption.

The factors of $M^{1/E}$ are the roots we require. Our next goal is to break the product $M^{1/E}$ into two subproducts, the breakup is implied by the structure of the binary tree T used to generate the product M. We repeat this recursively to break up the product into its b factors. (See figure 2).

Let $e_1, e_2, \ldots, e_k$ be the labels associated with the left branch of the root of the binary tree T. We define an exponent X by means of the Chinese remainder theorem:

$$\begin{aligned}
X &= 0 \pmod{e_1}, \\
X &= 0 \pmod{e_2}, \\
&\vdots \\
X &= 0 \pmod{e_k}, \\
X &= 1 \pmod{e_{k+1}}, \\
X &= 1 \pmod{e_{k+2}}, \\
&\vdots \\
X &= 1 \pmod{e_b}.
\end{aligned}$$

There is a unique solution for X modulo $\prod_{i=1}^{b} e_i = E$.

By definition

$$\begin{aligned}
X &= (\prod_{i=1}^{k} e_i) \cdot X_1, \\
X - 1 &= (\prod_{i=k+1}^{b} e_i) \cdot X_2.
\end{aligned}$$

Let $P_1 = \prod_{i=1}^{k} e_i$ and $P_2 = \prod_{i=k+1}^{b} e_i$, then $X = P_1 \cdot X_1$ and $X-1 = P_2 \cdot X_2$. Note that $\log X < \log E$, $\log X_1 + \log P_1 = \log X$, $\log X_2 + \log P_2 = \log X$, $\log E = \log P_1 + \log P_2$, and thus $\log X_1 + \log X_2 < \log X$.

Denote

$$\begin{aligned}
M_1 &= m_1^{P_1/e_1} \cdot m_2^{P_1/e_2} \cdots m_k^{P_1/e_k}, \text{and} \\
M_2 &= m_{k+1}^{P_2/e_{k+1}} \cdot m_{k+2}^{P_2/e_{k+2}} \cdots m_b^{P_2/e_b}.
\end{aligned}$$

Note that M_1 and M_2 have already been computed, as the left and right branch values of the root, during the tree based computation of M.

Raise $M^{1/E}$ to the Xth power modulo N:

$$\begin{aligned}
(M^{1/E})^X &= (\prod_{i=1}^{b} m_i^{1/e_i})^X \\
&= (\prod_{i=1}^{k} m_i^{1/e_i})^{P_1 \cdot X_1} \cdot (\prod_{i=k+1}^{b} m_i^{1/e_i})^{P_2 \cdot X_2} \cdot \prod_{i=k+1}^{b} m_i^{1/e_i} \\
&= M_1^{X_1} \cdot M_2^{X_2} \cdot \prod_{i=k+1}^{b} m_i^{1/e_i}.
\end{aligned}$$

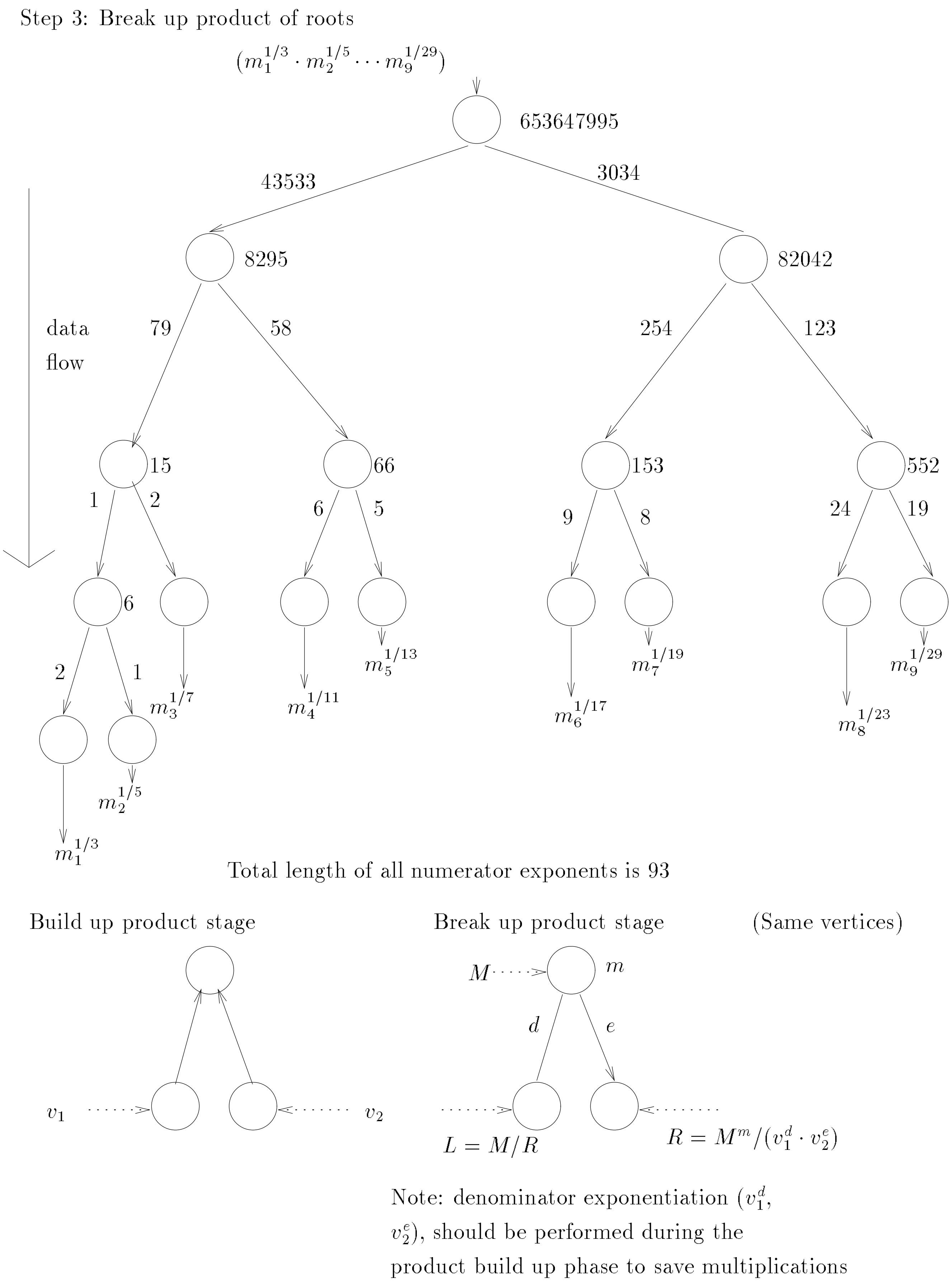

Figure 2: Break up Product of Roots

To solve for $P_1 = \prod_{i=k+1}^{b} m_i^{1/e_i}$ we raise M_1 to the power X_1, raise M_2 to the power X_2, and divide out. To solve for $P_2 = \prod_{i=1}^{k} m_i^{1/e_i}$ we divide $M^{1/E}$ by $\prod_{i=k+1}^{b} m_i^{1/e_i}$.

We know recursively break up the two products P_1 and $P - 2$.

Every leaf labeled l contributes $\log l$ bits to the appropriate exponents (X and X_1 or X and X_2) for *every* level between the leaf and the root. Thus, the overall number of multiplications is $O(W)$. The number of modular divisions required is $O(b)$.

To summarize:

Lemma 1 *Let $e_1, e_2, \ldots, e_b$ be b different encryption exponents, relatively prime to $\phi(N)$ and to each other. Given messages m_1, m_2, ..., m_b, we can generate the b roots*

$$m_1^{1/e_1} \pmod N, \quad m_2^{1/e_2} \pmod N, \ldots, \quad m_b^{1/e_b} \pmod N$$

in $O(\log b(\sum_{i=1}^{b} \log e_i) + \log N)$ modular multiplications and $O(b)$ modular divisions.

By choosing the e_i exponents to be polynomial in n and choosing the batch size $b = n/\log^2 n$ we get $O(n)$ multiplications overall and $O(\log^2 n)$ multiplications per root.

Remark: Even if the encryption exponents were exponential in polylog(n), say $2^{\log^c n}$ for some $c \geq 1$, we would still get polylog(n), $(\log^{c+O(1)} n)$, multiplications for both encryption and decryption operations.

4 Comparison with Other Schemes

Various schemes for key-exchange, public key cryptosystems, and digital signature have been proposed ([DH], [RSA], [R], [E], [GMR], [FS], ...). In several schemes the secret is used as an exponent during the private operation and therefore the number of modular multiplications required is at least the security parameter.

The security of any computationally secure scheme is parameterized by a security parameter k, where the assumption is that the work required to break the scheme is proportional to 2^k. For number theoretic schemes, the security parameter k determines the size of the composite or prime modulus to be used. The modulus size is determined by appropriate complexity assumptions on the the best factoring or discrete log algorithms. (As the case may be).

The heuristic expected running times for some of the problems of interest in cryptography are

1. Factoring an n bit integer using the general number field sieve [18]:

$$e^{(c+o(1))n^{1/3}(\ln^{2/3} n)},$$

where $c \approx 2$ [7].

2. Discrete logarithms in GF(p), for an n bit prime p, using number field sieve:

$$e^{(c+o(1))n^{1/3}(\ln^{2/3} n)},$$

where $c \approx 2$.

It follows that for number theoretic schemes, to obtain a security parameter of k, the modulus length n must be proportional to $n(k) = k^3/\log^{O(1)} k$. (If factoring

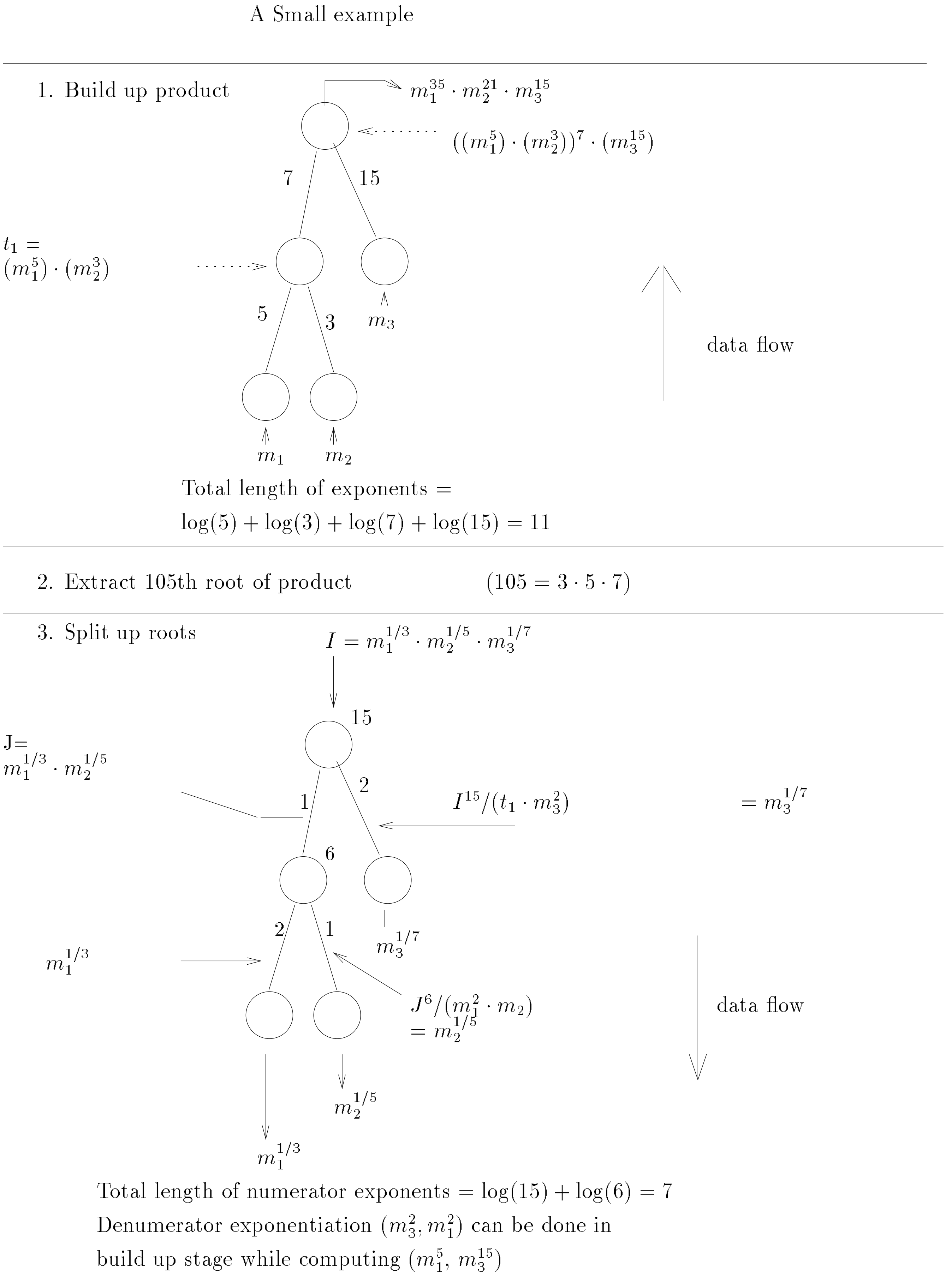

Figure 3: A small example

algorithms advance further, the function describing the dependence of n on k will grow further too).

The private operation of the RSA scheme is simply a root extraction, this is performed by taking the inverse of the encryption exponent modulo Euler's totient function of the modulus and computing a root by exponentiation with this inverse. Because this inverse will in general be as long as the modulus, (and cannot be too short see [26]) it follows that the number of multiplications required for RSA with a security parameter of k is $n(k)$. Using the Chinese remainder theorem, these multiplications can be done modulo each of the prime factors of the public modulus independently, but the total number of multiplications remains at least $n(k)$.

For a security parameter k, both Diffie-Hellman key exchange and the El-Gamal signature scheme [11] require exponentiation modulo a prime of length $n(k)$, where the exponent is $n(k)$ bits long. At least for Diffie-Hellman it has been suggested that the exponent be shorter but it certainly cannot be shorter than k for a security parameter of k. Other attempts to improve efficiency are to use $GF(p^n)$ where the multiplication operation is more efficient but then the key sizes are much longer. Using elliptic curves over $GF(p^n)$ has also been studied, leading to shorter key sizes [16, 19], but some problems arise as well [20].

The Fiat-Shamir signature scheme [12] is much more efficient than RSA for signatures simply because it requires only k multiplications of $n(k)$ bit numbers rather than $n(k)$ multiplications of $n(k)$ bit numbers. Variants of this scheme include Guillou-Quisquater [14] which require approximately $3k$ multiplications of $n(k)$ bit numbers, and the Digital Signature Algorithm (DSA) which uses exponents of fixed length 160 ($> k$) modulo primes of length $n(k)$ for generating signatures and exponents of length 160 modulo primes of length $n(k)$ for verification. The DSA signature process can be split into a pre-computation stage that requires most of the work, and a very fast message-dependant stage.

Excluding the schemes based on operations in $GF(p^n)$ or elliptic curves over $GF(p^n)$, all of the schemes described above require at least k multiplications of $n(k)$ bit integers for performing the private operation. Batch RSA requires an amortized $O(\log^2 k)$ multiplications of $n(k)$ bit integers, *i.e.*, an exponential improvement over all other schemes, albeit only amortized performance.

For some applications, the DSA can be viewed as competitive with Batch RSA because the pre-computation can be performed at off-peak times and can be considered as free. However, then the DSA public operation (verification) is much more expensive than RSA with (relatively) short exponents, see discussion below.

5 Notes on Security

Use of small encryption exponents for RSA was first suggested by Knuth [K]. A problem arises in the use of small exponents for standard RSA *encryption* if the message is numerically small or messages related via some known polynomials are encrypted to several different recipients. (M. Blum [B] for identical messages, Håstad [H] for fixed polynomials).

We have a more serious problem with encryption in that if the same message is encrypted with different (relatively prime) encryption exponents modulo the *same* modulus then the message can be reconstructed.

In both cases, if RSA is used for key exchange then there is no problem, all the cryptanalyst can learn are random values modulo N. Otherwise, it seems that standard cryptographic practices of randomizing cleartext and appropriate feedback mechanisms effectively overcome these attacks.

Shamir has shown that knowing $m^{1/p_1}, m^{1/p_2}, \ldots, m^{1/p_k}$ cannot give us m^{1/p_0} for pairwise relatively prime p_i [S] (as we could extract m^{1/p_0} using this procedure as a black box).

The fact that Batch RSA makes use of different encryption exponents may be advantageous relative to standard RSA in that the multiplicative relationship between different roots no longer holds.

6 Security of Batch RSA *vs.* RSA

As far as we know today, the security assumption for RSA can be stated as follows: "There exists some constant c such that for any sufficiently large k and appropriately chosen moduli N, where $\log N = n = n(k) \approx k^3$, and for arbitrarily chosen encryption exponents e, relatively prime to $\phi(N)$, and random inputs m, computing $m^{1/e} \bmod N$ cannot be done with probability p and time T where $T/p < \exp(c \cdot k)$.

If this is so, then the Batch RSA process is as secure as RSA. Imagine that there were a constant $c' \leq c - o(1)$, a black box that gets inputs, $N = \exp(k^3)$, m, and a set S of exponents, $|S| = \mathrm{polylog}(N)$. Assume that in time T, with probability p, taken over the inputs m and the internal coin tosses of the box, the black box outputs the value $m^{1/e} \bmod N$ for some $e \in S$. Imagine too that $T/p < \exp(c'k)$.

The assumptions above imply that at least for one specific element $x \in S$, the value $m^{1/x} \bmod N$ can be computed with probability $p/|S|$ in time T. However, for sufficiently large k,

$$\begin{aligned} T|S|/p &< \exp(c'k + \log(|S|)) \\ &= \exp(c'k + \log k + O(1)) \\ &\leq \exp((c' + o(1))k) \leq \exp(ck). \end{aligned}$$

Thus, taking a set of $|S|$ exponents for whom the security assumption of RSA holds, and using the Batch RSA black box, we can disprove the RSA security assumption for at least one of these exponents.

If the RSA security assumption is that RSA is secure only for randomly chosen encryption exponents (not the assumption today), then any specific range of exponents will not be secure, but a random set of exponents, or even a $\mathrm{polylog}(N)$ random sequence of successive exponents relatively prime to $\phi(N)$ will be secure.

7 Constants and Practical Considerations

We assume that modular exponentiation requires $c \cdot k$ modular multiplications for an exponent of length k, $c = 1.5$. This is true for the standard exponentiation algorithm if the exponent is chosen at random. None of our exponents are really random but this is a reasonable upper bound on the work required.

Generating M as described in the proceeding section requires $c \cdot W$ multiplications. Taking the Eth root requires $c \cdot \log(N)$ multiplications, extracting the factors from $M^{1/E}$ requires $2c \cdot W$ multiplications.

In fact, we can do better — extracting the factors of $M^{1/E}$ can be done in $c \cdot W$ multiplications provided that about $W/4$ additional multiplications are done when computing M. Overall, the number of multiplications required to extract the b roots is therefore $2c \cdot W + W/4 + c \cdot \log N$.

Reducing the number of multiplications to extract the factors of $M^{1/E}$ involves a slight digression: Our goal is to compute $y^{Z_1} \pmod N$ and $y^{Z_2} \pmod N$. If Z_1 and Z_2 are random then it seems that this requires $c \cdot (\log Z_1 + \log Z_2)$ modular multiplications. In fact, we can compute $y^{Z_1 \cap Z_2} \pmod N$ where $Z_1 \cap Z_2$ denotes the bitwise and operation between Z_1 and Z_2, compute $y^{Z_1 \cap \overline{Z}_2}$ and $y^{Z_2 \cap \overline{Z}_1}$ and multiply the appropriate results to get y^{Z_1} and y^{Z_2}. It is not hard to see that computing the three intermediate products can be done in $\log N + 3/4 \cdot \log N$ multiplications given that Z_1 and Z_2 are chosen at random in the range $1 \ldots N$. In addition, this can be done without any significant cost in storage other than the area required to hold the three intermediate results.

We can use the trick described in the last paragraph so as to compute the values $M_1^{X_1}$ and $M_2^{X_2}$ as a byproduct of computing M, at a cost of $1/4(\log X_1 + \log X_2)$ multiplications. Recall that the final stage in computing M involves raising M_1 to some exponent R and M_2 to some exponent L. The same holds for all levels of the recursion.

8 Modes of Operation

Another practical concern in the context of Batch RSA is what exponents should be used? We consider the following variants:

1. When two-way communication is possible, one possibility is to enter into a negotiation process where unique small exponents are assigned to remote sites for public encryption operations. This has a disadvantage in that there is a preliminary negotiation phase.

2. For public encryption operations, the remote station could choose an exponent at random from a small preset domain. Because there may be duplicate exponents, this means that several batches may have to be solved. If our domain is the set of the k smallest primes greater than some initial value $(2^{16} + 1?)$, then a batch of k private operations, with randomly chosen exponents, will require up to $\log k / \log \log k$ batch RSA operations. Choosing a domain of size k^2 means that the expected number of batches required is only $O(1)$.

3. As one may assume that the number of users is polynomial in n, it follows that unique exponents can be assigned to each and every remote station, while still requiring only polylog(n) multiplications per private operation. This has the additional advantage that the users are completely unaware of the Batch RSA process and view the process as though it were standard RSA. A variant of this scheme is to map several users (based on account number?) to the same public exponent.

4. One variant of our scheme would be to use different encryption exponents for every encryption or digital signature. *E.g.*, the ith prime for the ith operation. As long as there are no more than a polynomial number of transactions (in n) then both private and public operations would require polylog(n) multiplications.

8.1 Distributed Batch RSA

As the Batch RSA merger and split-up phases involve no secret information, this lets us use Batch RSA to isolate the private key from the system, irrespective of it's size. All private transactions can be reduced to one root extraction that can be solved on a weak and isolated processor. Every link-encryptor, mainframe, etc., requests one root, these roots can be merged again so the entire system is driven by one root extraction.

Distributed Batch RSA never requires more than n bits transmitted to and from a site, thus distributed Batch RSA is very efficient in communications.

In the context of using RSA for secrecy, it is desirable to ensure security in transit. E.g., from the link encryptors to the smartcard and back (especially back). To do this we can use the standard Zero-Knowledge trick of multiplying the real value with a random value raised to an appropriate power. *E.g.*, to extract the 17th root of a ciphertext C the link encryptor chooses a random value R and computes $R^{17} \cdot C$, this is then used as input to the distributed batch RSA process. Within the link encryptor, the result $R \cdot C^{1/17}$ is divided by R to obtain $C^{1/17}$.

9 Subsequent Related Work

Batch Diffie-Hellman, [5], makes use of the Batch RSA process in a centralized setting. This is a version of Diffie-Hellman with a composite modulus, and where the different secret exponents used by the central site are equivalent to the root extractions of Batch RSA.

The use of Batch RSA was also suggested in [10] in the context of existentially unforgeable signature schemes where they use multiple different roots in the signature scheme, suitable for the batch RSA process.

10 Acknowledgments

This work has its origins in Shamir's cryptographically secure pseudo random sequence [S] and in David Chaum's observation that multiples of different relatively-prime roots are problematic in the context of untraceable electronic cash [CFN] as the roots can be split apart.

I with to thank Noga Alon, Miki Ben-Or, Manuel Blum, Benny Chor, Shaffi Goldwasser, Dick Karp, Silvio Micali, Moni Naor, Ron Rivest, Claus Schnorr, Adi Shamir, Ron Shamir and Yossi Tulpan for hearing me out on this work.

Very special thanks to the J. of Cryptology editor, Gilles Brassard, and to the anonymous referees of this paper.

References

[1] Abadi, M., Feigenbaum, J., and Kilian, J., On Hiding Information from an Oracle, Proceedings of the 19th Annual ACM Symposium on Theory of Computing, pp. 195–203. New York City, May 25–27, 1987.

[2] Aho, A.V., Hopcroft J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] Blum, M., Personal communication.

[4] Blum, M., Floyd, R.W., Pratt, V., Lewis, R.L., and Tarjan, R.E., Time bounds for selection, J. Comput. System Sci. 7 (1973), pp. 448–461.

[5] Yacobi, Y., and Beller, M.J., Batch Diffie-Hellman Key Agreement Systems, Proceedings of Eurocrypt '92, pp. 208–217.

[6] Coppersmith, D., Fast evaluation of logarithms in fields of characteristic two, IEEE Trans. Inform. Theory, vol. IT-30, no. 4, pp. 587–592, July 1984.

[7] Coppersmith, D., Modifications to the number field sieve, IBM Research Report #RC 16264.

[8] Chaum, D., Fiat, A., and Naor, M., Untraceable Electronic Cash, Proceedings of Crypto '88, pp. 319–227. 1976.

[9] Diffie, W. and Hellman, M.E., New Directions in Cryptography, IEEE Trans. on Information Theory, Vol IT-22, 1976.

[10] Dwork, C., and Naor, M., An efficient existentially unforgeable signature scheme and its applications, Advances in Cryptology – Crypto '94, Lecture Notes in Computer Science, No. 839, Springer Verlag, 1994, pp. 234–346.

[11] El Gamal, T., A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Transactions on Information Theory, Vol IT-31, no. 4, pp. 459–472, July 1985.

[12] Fiat, A., and Shamir, A., How to Prove Yourself: Practical Solutions to Identification and Signature Problems, Advances in Cryptography: Proceedings of Crypto '86, pp. 186–194, Berlin: Spinger-Verlag, 1987.

[13] Goldwasser, S., S. Micali, and R.L. Rivest, A digital signature scheme secure against adaptive chosen message attacks, SIAM J. Comput., vol. 17, no. 2, pp. 281–308, April, 1988.

[14] Guillou, L.C., and Quisquater, J.J., A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, In Advances in Cryptology: Proc. Eurocrpyt '88, C.G. Gunther, Ed., Davos, Switzerland, May 25–27, pp. 123–128, 1988.

[15] Håstad, J., On using RSA with Low Exponent in a Public Key Network, Proceedings of Crypto '85, pp. 403–408.

[16] Koblitz, N., Elliptic curve cryptosystems, Math. Comp., vol. 48, pp. 203–209, 1987.

[17] Knuth, D., The Art of Computer Programming, vol. 2: Seminumerical Algorithms, 2nd ed., Addison-Wesley, 1981.

[18] Lenstra, A.K., Lenstra, Jr., H.W., Manasse, M.S., and Pollard, J.M., The number field sieve, Proc. 22nd ACM Symp. Theory of Computing, p. 464–572, 1990.

[19] Menezes, A., and Vanstone, S. The implementation of elliptic curve cryptosystems, In Advances in Cryptology: Auscrypt'90, J. Seberry, J. Pieprzyk, Eds, Sydney, Australia, Jan. 1990, pp. 2–13.

[20] Menezes, A., Okamoto, T., and Vanstone, S., Reducing elliptic curve logarithms to logarithms in a finite field, Proceedings of the 22nd annual ACM Symposium on the Theory of Computing, pp. 80–89, 1991.

[21] Quisquater, J.J., and Couvreur, C., Fast decipherment algorithm for RSA public-key cryptosystem, Electronic letters, vol. 18, no. 21, 1982, pp. 905–907.

[22] Rabin, M.O., Digitalized signatures, in Foundations of Secure Computation, Academic Press, NY, 1978.

[23] Rivest, R.L., Shamir, A. and Adleman, L., A Method for Obtaining Digital Signatures and Public Key Cryptosystems, Comm. ACM, Vol. 21, No. 2, 1978.

[24] Shamir, A., On the Generation of Cryptographically Strong Pseudorandom Sequences, ACM Trans. on Computer Systems, Vol. 1, No. 1, 1983.

[25] Tarjan, R.E., Amortized computational complexity, SIAM J. Alg. Discrete Meth., 2(6)(1985) pp. 306–318.

[26] Wiener, M.J., Cryptoanalysis of Short RSA exponents, IEEE Transactions on Information Theory, Vol. 36, No. 3, May 1990, pp. 553–558.