Batch Diffie-Hellman Key Agreement Systems and their Application to Portable Communications

Michael J. Beller Yacov Yacobi

Bellcore
New Jersey, U.S.A.

June 19, 1991

Abstract.
RSA (Rivest, Shamir and Adleman) is today's most popular public key encryption scheme. Batch-RSA (due to Fiat) is a method to compute many ($n/log_2^2(n)$, where n is the security parameter) RSA decryption operations at a computational cost approaching that of one normal decryption. It requires that all the operations use the same modulus, but distinct, relatively prime in pairs, short, public exponents. A star-like key agreement scheme could use such a system to slash computational complexity at the center. We show a real life example of such a system – secure portable telephony. Unfortunately, in this system Batch-RSA cannot be employed effectively, due to a delay component which arises from the nature of RSA key exchange. We show that mathematical ideas similar to Fiat's can lead to a Batch-Diffie-Hellman key agreement scheme, that does not suffer such delay and is comparable in efficiency to Batch-RSA. We prove that with some precautions, this system is as hard to break as RSA with short public exponent. In practice our method improves processing time at the center by a factor of 6 to 17 when compared to (non-batch) Diffie-Hellman schemes with full-size exponents and moduli in the practical range. Smaller improvements (on the order of 1.6 to 3) are obtainable when compared to a Diffie-Hellman scheme employing abbreviated exponents.

1 Introduction

RSA (Rivest, Shamir and Adleman) [13] is today's most popular public key encryption scheme. Batch-RSA [7] is a method to compute many ($n/log^2(n)$, where n is the security parameter, throughout logarithms are to the base 2) RSA decryption operations at a computational cost approaching that of one normal decryption. It requires that all the operations use the same modulus, but distinct, relatively prime in pairs, short, public exponents. A star-like key agreement scheme could use such a system to slash complexity at the center. We show a real life example of such a system – secure portable telephony. Unfortunately, in this system Batch-RSA can not be employed effectively,

R.A. Rueppel (Ed.): Advances in Cryptology - EUROCRYPT '92, LNCS 658, pp. 208-220, 1993.

due to a delay component which arises from the nature of RSA key exchange. We then show that mathematical ideas similar to Fiat's can lead to a Batch-Diffie-Hellman key agreement scheme, that does not suffer such delay and is comparable in efficiency to Batch-RSA. We prove that with some precautions, this system is as hard to break as RSA with short public exponent.

In practice our method improves processing time at the center by about an order of magnitude when compared to (non-batch) Diffie-Hellman schemes with full-size exponents and moduli in the practical range. Smaller improvements (on the order of 1.6 to 3) are obtainable when compared to a Diffie-Hellman scheme employing abbreviated exponents.

Section 2 describes Fiat's original Batch-RSA method, in section 3 we describe how we plan to use it with a Diffie-Hellman-like system, and in section 4 we analyze the security of our proposed system. Section 5 of this paper describes a manner in which the method of this paper can be applied to improve efficiency in a real portable communications system (PCS), and quantifies the achievable improvement. Section 6 explains why batch RSA introduces unacceptable delays, and motivates batch DH.

2 Batch-RSA

Suppose we have to compute $m_i^{1/e_i} \bmod N$ for $i = 0,1,2,\ldots,b-1$, where b is the batch size, and the e_i's are relatively small (and the inverses $1/e_i \bmod \lambda(N)$ are large, $\lambda(N) = (p-1)\cdot(q-1)/2$). The main idea of Batch-RSA is to compute first $c \equiv \prod_{i=0}^{b-1} m_i^{p/e_i} \bmod N$, where $p = \prod_{i=0}^{b-1} e_i$, using a special efficient binary tree structure, to be described later. The second phase is to compute $m \equiv c^{1/p} \bmod N$, which is a full size modular exponentiation (but the cost is spread over b computations). The last phase is to break $m \equiv \prod_{i=0}^{b-1} m_i^{1/e_i} \bmod N$ into its b separate components $m_i^{1/e_i} \bmod N$, $i = 0,1,2,\ldots,b-1$, which is the desired output of our computation. This is done using the binary tree developed in the first step in a very efficient way.

The binary tree: To simplify explanations we assume that $b = 2^k$, for some k. Create a complete binary tree where the leaves are labeled $m_0, m_1, m_2, , \ldots, m_{b-1}$. A path is identified with the corresponding binary sequence $\in \Sigma^*$, $\Sigma = \{0,1\}$. We use the symbol ϵ to denote the string of length zero. Right sons are associated with 1, and left sons with 0. We refer to any arc in the tree using the unique path leading to it, i.e. $\eta \in \Sigma^{k'}$, $k' < k$ is the arc in depth k' from the root, which is approached when traversing the tree from the root according to η. If η is of length k then it leads to a leaf labeled m_i s.t. η is the binary representation of i. Let $x \in \Sigma$. ηx denotes a sequence composed of x concatenated to the right of η, $\bar{x}$ denotes the complement of x. Each arc η has label $l(\eta)$. Arcs in the tree are labeled bottom-up according to the following rule.

- Let ηx be a path leading to a leaf associated with message m_i, then $l(\eta\bar{x}) = e_i$.
- For $\eta \in \Sigma^{k'}$, $k' < k-1$, $l(\eta x) = l(\eta\bar{x}0)\cdot l(\eta\bar{x}1)$.

The above labeling procedure is independent of the actual messages $\{m_i\}$. It depends on the exponents only, hence if those are fixed, this procedure may be done off line, once and for all. Nodes in the tree contain data, which depend on both the messages and the exponents. We refer to each node by the path leading to it. The data stored in node η is denoted $d(\eta)$. Initially the content of each leaf i is the corresponding message m_i. The content of each node in the tree is computed bottom-up, after its sons were computed using $d(\eta) \equiv d(\eta 0)^{l(\eta 0)} \cdot d(\eta 1)^{l(\eta 1)} \bmod N$. It follows that the content of the root is the desired $d(\epsilon) \equiv \prod_{i=0}^{b-1} m_i^{p/e_i} \bmod N$. This concludes the first phase (see Fig-1).

After computing $M \equiv d(\epsilon)^{1/p} \equiv \prod_{i=0}^{b-1} m_i^{1/e_i} \bmod N$ (second phase), we use the tree (top-down) to break M into its components (third phase). This is done recursively as follows.

Let $0, 1, \ldots, q-1$ be the leaves associated with the left son of the root (and $q, q+1, \ldots, b-1$ with the right son). Note that $l(0) = \prod_{i=0}^{q-1} e_i$, $l(1) = \prod_{i=q}^{b-1} e_i$, and $p = l(0) \cdot l(1)$. Using the Chinese Remaindering algorithm [1] compute X, s.t. $X \equiv 0 \bmod e_i$, $i = 0, \ldots, q-1$, and $X \equiv 1 \bmod e_i$, $i = q, \ldots, b-1$. Here we use the fact that the e_i's are relatively prime in pairs. From the construction of X it follows that $X \equiv 0 \bmod l(0)$ and $X \equiv 1 \bmod l(1)$, hence there exist X_1 and X_0 s.t. $X = l(1) \cdot X_1 + 1$ and $X = l(0) \cdot X_0$. Denote $M_0 \equiv \prod_{i=0}^{q-1} m_i^{1/e_i} \bmod N$, and $M_1 \equiv \prod_{i=q}^{b-1} m_i^{1/e_i} \bmod N$. For convenience we use the shorthand l_0, l_1, d_0, d_1 instead of $l(0), l(1), \ldots$, etc. Since $M_0 \equiv d_0^{l_0/p} \bmod N$, and $M_1 \equiv d_1^{l_1/p} \bmod N$, we have $d_1 \equiv M_1^{l_0} \bmod N$ and $d_0 \equiv M_0^{l_1} \bmod N$. $c \equiv d_0^{l_0} \cdot d_1^{l_1} \bmod N$, hence, $M^X \equiv c^{X/p} \equiv d_0^{l_0(l_1 X_1 + 1)/p} \cdot d_1^{l_1 l_0 X_0/p} \equiv d_0^{X_1} d_1^{X_0} d_0^{l_0/p}$. But the last multiplicand is just M_0, hence $M_0 \equiv M^X/(d_0^{X_1} \cdot d_1^{X_0}) \bmod N$, and $M_1 \equiv M/M_0 \bmod N$. The d's were computed in phase I, and are stored in the nodes. The X's do not depend on the messages and may be computed off-line. The process repeats recursively, until at the leaves the desired output is reached (see Fig-2).

As is shown in [7] the total complexity of a batch computation approaches that of one full size exponentiation.

Using Batch-RSA together with Montgomery's modular reduction
Montgomery's modular reduction [11] is very popular among public-key implementors [6]; so we must address the question "can the two methods coexist?" The answer is positive both theoretically and practically. Montgomery's method addresses additions and multiplications, which comprise the bulk of the computation in Batch-RSA (and Batch-DH, which will be discussed later in the paper). The only operations which can not be done using Montgomery's method are the two divisions that have to be done in each node when going down the tree (phase 3). Asymptotically they are negligible, and practically they are small compared to the larger exponentiations done in each node. The reason is the following: We have to do modular $a \boxminus b/cd$, $e \equiv f/a$, where each of b, c, d, f is given as a Montgomery number (i.e. a multiple of $R = 2^n \bmod N$, where n is the length in bits of the modulus). Let $\mathcal{M}(a \cdot b)$ denote a Montgomery multiplication of a and b, (i.e. $\mathcal{M}(ab) \equiv abR^{-1} \bmod N$). Here is a proposed procedure

Begin

1. $cd = \mathcal{M}(c \cdot d)$;
2. regular $a \equiv b/cd \bmod N$; (result: non-Montgomery number)
3. regular $e \equiv f/a \bmod N$; (result: a Montgomery number)
4. $a \equiv \mathcal{M}(a \cdot (R^2 \ mod \ N))$; (to get back a Montgomery number).

End

So the cost of the (relatively small number of) divisions is inflated by a small factor (2) for the loss of the advantage associated with the use of Montgomery's reduction method. Also added to each division is the cost of step 4 above.

There is some increased software complexity due to the need for both regular and Montgomery numerical routines. However, this small increase in code size is not significant in the central office application.

3 Batch Diffie-Hellman

In this section we show how the ideas of Batch-RSA can be extended to support Batch Diffie-Hellman. We use the example of a portable communications system which has many portable radio telephones ("portables") accessing a central office via a matrix of fixed access points called "ports".

Let (S_i, P_i) be the secret and public keys, respectively, of portable i, $i = 1, \ldots, n$, and let (S'_j, P'_j) be the secret and public key of a port j. The central facility is trusted by all ports. (In section 5 we show that the fixed assignment of keys to ports–i.e. key j to port j–is naive, and leads to an inefficient real-time system. Nonetheless, this example is adequate for the purposes of this section.)

In the basic Diffie-Hellman [5] scheme there is some prime modulus, N, common to the whole system, and some primitive element of $GF(N)$, denoted α, s.t. $(\forall i)[P_i \equiv \alpha^{S_i} \bmod N]$, similarly, $(\forall j)[P'_j \equiv \alpha^{S'_j} \bmod N]$, and a session key between i and j is $SK_{ij} \equiv \alpha^{S_i \cdot S'_j} \bmod N$, efficiently computable exclusively by i and j (or the center on behalf of j). Many other variations exist. For example, sometimes it may be desirable to choose a composite modulus [14] [9], and most of the time we need key distribution systems that authenticate the users, and are dynamic [15].

In order for the central authority to be able to use Batch-DH we have to introduce additional constraints.

- First, the modulus should be a composite, with secret prime factorization (two large primes) known only to the central facility, and such that factorization of N is hard.

- Second, the secret key of each port j, S'_j, is not chosen at random. Rather a relatively small e_j, is chosen, and its multiplicative inverse modulo $\lambda(N)$, is computed. This is S'_j. As before, $P'_j \equiv \alpha^{S'_j} \bmod N$. For modulus of size n bits we need $e_j < log_2(n)$.
- Third,the e_js must be relatively prime in pairs.

No new constraints are needed for the portable keys. They can be chosen with abbreviated secret exponents (130 bits seem currently secure) [16], or they can be identity based systems [10], or whatever. In that respect the system may be heterogeneous, i.e. users may choose inexpensive portables (with abbreviated secret exponents), or more expensive portables, with long secret exponents, and with the added convenience of being identity based [10]. This may be important if the cryptosystem is used for additional applications, such as monetary transactions.

4 Security

The following lemma is well known.

Lemma 1: Let α and β be generators of the multiplicative groups Z_p^* and Z_q^*, respectively, and let $\gamma \in Z_N^*$, $\gamma \equiv \alpha \bmod p$, $\gamma \equiv \beta \bmod q$, (i.e. γ is obtained from α and β using Chinese Remaindering). Then γ generates a maximal cyclic subgroup of Z_N^*, it is of size $\lambda = (p-1)(q-1)/2$ (for our choice of p and q), this cyclic subgroup, denoted M is the Cartesian product of Z_p^* and Z_q^*, and therefore is independent of the choice of α and β.

Assuming the base element, γ, for our CDH scheme is chosen according to lemma 1, we now give evidence to the security of our scheme, namely, if we assume that RSA with short public key is hard to break on the average, over the subset M, then so is CDH. This follows from Lemma 2, and the corollary which immediately follows it.

Lemma 2: Composite Diffie-Hellman key agreement scheme is at least as hard to break on the average as RSA for half the messages (here we average over M).

Proof: Assume there exists an oracle AL, s.t. for all N, β, x, y $AL(N, \beta, \beta^x, \beta^y) \equiv \beta^{xy} \bmod N$, and let $e \equiv x^{-1} \bmod \lambda(N)$. Let γ be the generator of the maximal cyclic subgroup M of Z_N^*, discussed in lemma 1.

Given RSA cracking problem, defined by Input: $N, e, c \equiv m^e \bmod N$, $c \in M$ ($m \in M$ implies $c \in M$ by M's closure); Output: m. We use oracle AL to solve it, as follows.

1. Find $\gamma \in Z_N^*$, a generator of M (using the construction of Lemma 1).
2. Compute $\beta \equiv \gamma^e \bmod N$.
3. Call oracle $AL(N, \beta, \gamma, c)$.

The oracle's answer in the last step is m, because $m \in M$ and hence $(\exists y)[m \equiv \gamma^y]$, hence $\gamma \equiv \beta^x, c \equiv \beta^y, m \equiv \beta^{xy}$, where all these congruences are modulo N.

Once γ is fixed, the above mapping is one to one, therefore, the reduction is measure preserving [2], and therefore preserves average case complexity.

Q.E.D.

Corollary: The reduction of Lemma 2 does not depend on the length of e. The case in which we are interested is CDH with short e. RSA with short public key reduces to CDH with exactly one short inverse by the same construction.

Lemma 3: Composite Diffie-Hellman key agreement scheme, in which exactly one of the exponents has a short inverse ($< O(n)$) is at most as hard to break on the average (over all messages) as RSA with short public key.

Proof: Let $AL2$ be an oracle that solves RSA problem, where the public exponent e is short ($e < O(n)$), i.e. $AL2(N, c, e) = m$, s.t. $m^e \equiv c \bmod N$. Given a CDH problem defined by Input: N, a, a^x, a^y; Output: $a^{xy} \bmod N$, where $x^{-1} \equiv e \bmod \lambda(N)$ is short, we use oracle AL2 to solve it as follows:

1. Find e, s.t. $(a^x)^e \equiv a \bmod N$ (since e is short exhaustive search is feasible).
2. Call oracle $AL2(N, a^y, e)$.

The answer of oracle $AL2$ is a^{xy}, as required.

In this reduction the mapping is one to one. Therefore, it is measure preserving [2], and therefore preserves average case complexity.

Q.E.D.

Lemma 4: Composite Diffie-Hellman key agreement scheme, in which both exponents have short inverses is easily breakable for $6/\pi^2$ of the instances.

Proof: If both secret exponents (x, y) have short inverses (e_i, e_j) then, as before, the adversary can find them by exhaustive search (raising the public keys to the power of $e_...$ until he gets α). If in addition the two short inverses are relatively prime (happens with probability $6/\pi^2$, see a theorem by Dirichlet in Knuth II, pp. 324) the adversary can break the system. First he finds a and b such that $ae_i + be_j = 1$, using extended Euclid's gcd algorithm [1]. Multiplying this equation by xy we get $ay + bx \equiv xy \bmod \lambda(N)$. It follows that the adversary can compute the session key $\alpha^{xy} \equiv (\alpha^x)^b \cdot (\alpha^y)^a \bmod N$.

Q.E.D.

Of course, the reduction of Lemma 2 does not hold when both exponents have short inverses.

5 System Considerations for Batch-Diffie-Hellman

The method of this paper can be used to improve the efficiency of network processing in a Portable Communications System key agreement and authentication subsystem. An

example Portable Communications System is described in [4]. An example key agreement and authentication protocol, involving Diffie-Hellman computations, and tuned to the requirements of portable communications, is described in [3] as the last of the candidate protocols described in that paper.

The general framework of the system is that portable telephones access the network via a matrix of radio ports. Providing service involves establishing an encrypted, authenticated channel in the initial phase. The encryption is needed to protect the user's privacy from eavesdroppers using radio scanners, and the authentication is needed to protect the service provider from attempts to obtain service fraudulently (i.e. without intent to pay for the call).

During operation, the portable units make requests for service (e.g. to originate or answer a call). Each of these requests begins by employing a protocol to establish a private and authenticated channel. The protocol requires the network to perform a Diffie-Hellman exponentiation. It is necessary for the network to perform these computations quickly, so as not to unduly delay the user's access to service (and increase the holding time of the radio channel and network resources). Since there will often be many users requesting service at the same time, we can make use of the method of this paper to perform a number of Diffie-Hellman computations in parallel, and thereby increase the effectiveness of the processing power available in the network. This can ultimately decrease the cost of providing the PCS service.

The primary issue to be addressed is how much improvement can be obtained. The first question to ask is "how can we ensure that when requests come in, they can be processed in a batch?" One naive method (mentioned in section 3) is to assign a different e_j to each port. However, we can not ensure that there will always be exactly one service request available from each of the ports. Thus, this system would require an enormous (impractical) precomputation phase. This is because the method requires us to precompute a "tree" for each possible combination of exponents we wish to process simultaneously. A system with many ports will require a number of trees proportional to the number of combinations of the ports taken b at a time, where b is the optimal batch size. Any fixed assignment of e_j's to network equipment will either have enormous precomputation, or will be inefficient at obtaining the maximum batch size for processing, if it is still to provide the required response time.

Thus we propose to dynamically assign e_j's on a request-by-request basis. For example, for a given exponent length, we have an optimum batch size b. We choose b exponents e_j. When finished processing the previous batch of requests, the cryptoserver looks in its request queue. It scoops up b requests from the queue (assuming they are present–we discuss this issue later in this section) and assigns to each request one of the exponents e_j. The network immediately sends out the (precomputed) corresponding public keys α^{1/e_j}, and any other required information such as a certificate associated with the assigned public key, to each of the requesters. While the requesters do their Diffie-Hellman computations, the network will process these b computations in a batch. This "dynamic exponent" approach is not usable with Batch-RSA, as will be shown in

the next section.

What is the improvement which can be obtained with this method? As shown in the complexity section above, if we assume a n-bit modulus with n-bit exponents $1/e_j$, this allows improvement by a factor of $n/log_2^2(n)$. However, currently some Diffie-Hellman systems use abbreviated exponents. It is considered that the current best algorithm for breaking Diffie-Hellman systems with abbreviated exponents is due to Pollard (the "Lambda Method of Catching Kangaroos" [12]). By examining the relationship of the complexity of "catching a Kangaroo" vs. the difficulty of factoring using the best algorithm available (the number field sieve [8]), we come up with the following formula for the required length l (in bits) of an abbreviated exponent as a function of modulus N.

$$\sqrt{2^l} = e^{(1.9+o(1))\cdot ln(N)^{1/3}\cdot(lnln(N))^{2/3}}$$

By algebraic manipulation, we can get the "safe" exponent size for abbreviated Diffie-Hellman exponents as a function of modulus N. The o(1) in the formula refers to a small constant, values for which have been estimated at between 0.1 and 10. In our comparisons, we use values of o(1)=0, and o(1)=1. The value 0 gives the shortest abbreviated exponents (and therefore, by comparison, casts the method of this paper in the most negative light). The value o(1)=1 represents comparison to a more typical (conservative) system, and was chosen because it leads to an abbreviated exponent size of around 190 bits for a 512-bit modulus, which is currently considered to be an acceptable abbreviated exponent size for that modulus size.

To make our comparison, we take the exponent size obtained from the above formula, and divide it by the complexity value for Batch-Diffie-Hellman computations $n/b = log^2(n)$ obtained in Section 2 (because the batch method performs b exponentiations for the cost of one full n-bit exponentiation). The results are tabulated in Table-1 for some interesting values of n. (Where n is equal to $log(N)$, i.e. the number of bits in the modulus N). The table shows gain figures, as well as the corresponding short exponent lengths for o(1)=0 and o(1)=1.

Modulus Size (bits)	Long Exponent Gain Factor	Abbreviated Exponent Gain Factor			
		$o(1) = 0$		$o(1) = 1$	
		Exp. Size	Gain	Exp. Size	Gain
512	6	126	1.6	193	2.4
768	8	151	1.6	231	2.5
1024	10	171	1.7	262	2.6
2048	17	231	1.9	353	2.9

Table 1: Gain Factors for Batch-DH Over Other DH Systems.

From the table, one can see that the method of this paper can give a factor of 6 improvement over Diffie-Hellman with full-sized exponents and 512-bit modulus. The factor increases to 17 with a 2048-bit modulus. In comparison to a Diffie-Hellman system

with minimum-sized abbreviated exponents (corresponding to $o(1) = 0$), the method of this paper gives a factor of 1.6 improvement with a 512-bit modulus. As modulus sizes increase, the improvement factor also increases, reaching 2.0 near 2000 bit-modulus size. For more conservative abbreviated exponents (corresponding to $o(1) = 1$), the method of this paper gives improvements from 2.4 for 512-bit modulus, reaching 3.0 near 2000-bit modulus size. It is noted that any improvements made in attacking DH with abbreviated exponents will increase the attractiveness of the method of this paper.

It is noted that Fig-3 shows maximum possible improvement figures for this method. In order to achieve this maximum, there must be at least b requests waiting for service whenever the cryptoserver looks in its queue. In reality the requests for service are not evenly spaced in time (they're better modeled by a Poisson process). Thus, it will be difficult to assure that the server always sees b inputs in the queue, while also maintaining some reasonable performance constraint on the delay between service requests and their associated responses.

One way to mitigate this somewhat is to precompute not just batches of size b, but also $b/2$, $b/4$, etc. This will at least allow some improvement over straight n-bit exponentiation for those cases where fewer than b requests are in the queue. One caution is that, as the batch size gets below 4, it may be more efficient to use standard abbreviated-exponent DH. Given the processing scenario we've developed, we can readily switch back and forth between techniques, only using the parallel method when it is advantageous; if efficiency is not very high when there are few requests it does not matter, so long as the installed computational power is not exceeded.

6 Why not Batch-RSA?

Batch-DH and Batch-RSA are comparable in real-time computational requirements. However, in this section we show that, when used for key agreement in a system where response time is important (e.g. the Portable Communications system described above), Batch-RSA introduces unacceptable delays. As was explained in the previous section, the use of Batch-RSA or Batch-DH requires dynamic assignment of exponents to service requests in order to make the precomputation manageable while maintaining appropriate response time.

To use Batch-RSA, j has to use the same modulus N_j with different short exponents e_j's (there is no security hole here). The corresponding secret exponents d_j's are full size (In order to minimize complexity in the portable unit, we would use a medium-sized d_i).

With RSA, the network must send its public key to the portable. The portable then encrypts information using the public key, and sends results to the network. The network then computes the inverse function. Thus there is a random delay (dependent on the response time of the portable unit) between the network transmitting its public key information, and the time when the network could begin its secret computation. With DH, however, there is no such random delay. Immediately upon receipt of a public-key

from the portable, the network can begin computation of the secret operation.

Thus, if batch methods are used, DH will allow the network to define a batch and compute it all at once. RSA, on the other hand, will require the network to define a batch, send out information, and wait for all the portable units involved in the batch to return responses before it can begin computation. This will reduce everyone's performance unacceptably (by making the response time for all members of the batch dependent on the slowest-responding portable in the batch). Therefore, Batch-RSA is not useful for key agreement protocols in a Portable Communications System.

7 Conclusions

We have shown how the techniques of Batch-RSA can be expanded to support Batch-Diffie-Hellman. We have also shown that Batch-Diffie-Hellman can be used to reduce processing requirements of a central public-key cryptoserver in a portable communications system by a factor of 6 to 17 over a Diffie-Hellman system with full-sized exponents. Improvements on the order of 1.6 to 3 are obtainable when compared to a Diffie-Hellman scheme employing abbreviated exponents. We have noted that the benefit of Batch Diffie-Hellman increases with new advances in attacks against Diffie-Hellman with abbreviated exponents. We have also shown that it is not feasible to employ Batch-RSA to obtain similar improvements in a key-agreement system where response time is important.

8 Acknowledgement

We wish to thank Rafi Heiman and Arjen K. Lenstra for reviewing this paper, and for their many helpful comments, and to anonymous referee for his important criticism regarding our evidence of security.

References

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

[2] Ben-David, S., Chor, B., Goldreich, O., Luby, M., *On the Theory of Average Case Complexity*, Proc. STOC 1989, pp. 204-216.

[3] M. J. Beller, L. F. Chang, Y. Yacobi, *Privacy and Authentication on a Portable Communications System*, IEEE Globecom '91 Conference Proceedings, Phoenix, December 1991.

[4] D. C. Cox, *Portable Digital Radio Communications–An Approach to Tether-less Access*, IEEE Communications Magazine, Vol. 27, No. 7, July 1989.

[5] W. Diffie and M.E. Hellman, *New directions in cryptography*, IEEE Trans. on Inform. Theory, vol. IT-22, pp. 664-654, Nov. 1976.

[6] S.R. Dusse and B.S. Kaliski, *A Cryptographic Library for the Motorola DSP56000*, Advances in Cryptology: Proceedings of Eurocrypt '90, I.B. Damgard (Ed.), LNCS 473, Springer Verlag, May 1990, pp. 230-243.

[7] A. Fiat: *Batch RSA*, Proc. Crypto'89, pp 175-185.

[8] A.K. Lenstra, Private communication.

[9] K.S. McCurley, *A key distribution system equivalent to factoring*, J. Cryptology, vol. 1, no. 2, 1988.

[10] U.M. Maurer and Y. Yacobi *Non-interactive Public Key Cryptography* Proc. Eurocrypt'91.

[11] P.L. Montgomery, *Modular Multiplication Without Trial Division*, Math of Computation, Vol. 44, 1985, pp. 519-521.

[12] J.M. Pollard, *Monte Carlo Methods for Index Computation (mod P)*, Math, Comp. 32 (1978), 918-924.

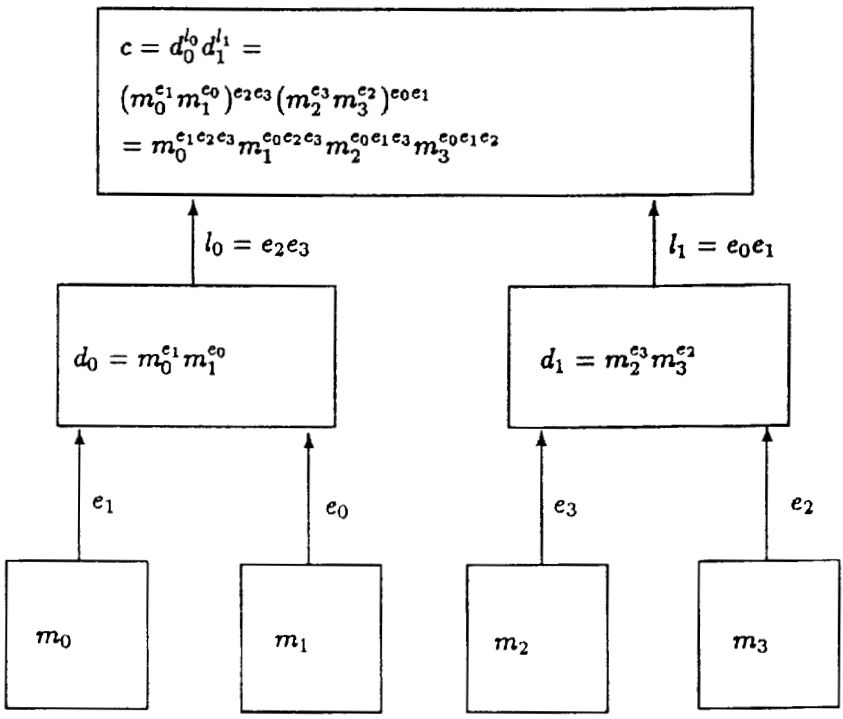
[13] R.L. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, vol. 21, pp. 120-126, 1978.

[14] Z. Shmuely, *Composite Diffie-Hellman public-key generating systems are hard to break*, TR 356, CS Dept., Technion, Feb. 1985.

[15] Y. Yacobi, *A key distribution "paradox"*, Proc. CRYPTO'90 Santa Barbara, CA, Aug. 11-15, 1990.

[16] Y. Yacobi, *Discrete-Log With Compressible Exponents* Proc. CRYPTO'90, Santa Barbara, CA, Aug. 11-15, 1990.

PHASE I:

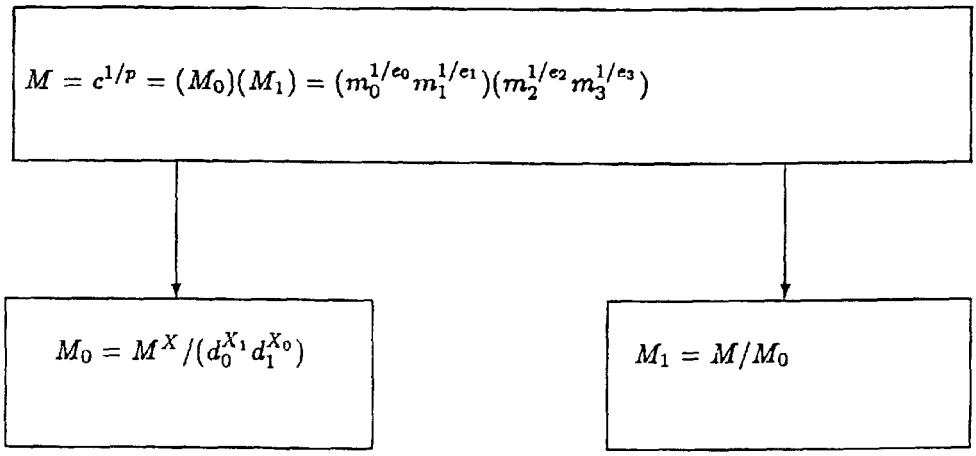


PHASE II:

$$c^{1/p} = m_0^{1/e_0} m_1^{1/e_1} m_2^{1/e_2} m_3^{1/e_3}$$

Figure 1: Phases I and II of Batch-RSA

PHASE III:



Continue recursively, with different X, X_0, X_1

in each level. d's and X's are precomputed.

Figure 2: Phase III of Batch RSA