J. Cryptology (1997) 10: 89–96

Journal of
CRYPTOLOGY

Batch Diffie–Hellman Key Agreement Systems*

Yacov Yacobi
Microsoft, Redmond, WA 98052, U.S.A.

Michael J. Beller
PenVision Information Systems, Inc., 670 North Beers St.,
Bldg 2, Holmdel, NJ 07733, U.S.A.

Communicated by Claude Crépeau

Received 6 July 1992 and revised 14 March 1996

Abstract. Batch Diffie–Hellman key agreement schemes are described, motivated, and analyzed for security and efficiency.

Key words. RSA, Diffie–Hellman, Key agreement.

1. Introduction

RSA (Rivest, Shamir, and Adleman) [8] is today's most popular public key encryption scheme. Batch–RSA [4] 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 cannot be employed effectively, 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 this system is as hard to break on the average as RSA with short public exponent, where the average is over all messages in the maximal cyclic subgroup of Z_N^*, N being the modulus.

The current system may also have application in Discrete-log based e-cash systems, where a bank can process many e-coins in parallel, and in DL based key escrow systems. The gain factor of $n/\log^2(n)$ is becoming more significant as we move from $n = 512$ to $n = 1024$ and beyond.

* Work done at Bellcore, Morristown, NJ, U.S.A.

Section 2 describes Fiat's original Batch–RSA method, in Section 3 we show how the same ideas could be used with a Diffie–Hellman-like system, and we motivate this transition. Section 4 presents security analysis for the newly proposed system.

2. Batch–RSA

Let $N = p \cdot q$, $p - 1 = 2p'$, $q - 1 = 2q'$, p, q, p', q' are primes. 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 such that 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 $d(\varepsilon) \equiv \prod_{i=0}^{b-1} m_i^{p/e_i} \bmod N$. This concludes the first phase.

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

Let $0, 1, \ldots, r-1$ be the leaves associated with the left son of the root (and $r, r+1, \ldots, b-1$ 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, such that $X \equiv 0 \bmod e_i$, $i = 0, \ldots, r-1$, and $X \equiv 1 \bmod e_i$, $i = r, \ldots, b-1$. 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 such that $X = l(1) \cdot X_1 + 1$ and $X = l(0) \cdot X_0$. Denote $M_0 \equiv \prod_{i=0}^{r-1} m_i^{1/e_i} \bmod N$, and $M_1 \equiv \prod_{i=r}^{b-1} m_i^{1/e_i} \bmod N$. 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 $d_0 \equiv M_0^{l_1} \bmod N$. The reader can easily verify that $M_0 \equiv M^X/(d_0^{X_1} \cdot d_1^{X_0}) \bmod N$, and this is the computation carried in the third phase when going down from the root to its left son. The right son is simply $M_1 \equiv M/M_0 \bmod N$. 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.

As is shown in [4] the total complexity of a batch computation approaches that of one full size exponentiation, so the gain factor is $O(n/\log^2 n)$, which is true for Batch Diffie–Hellman as well.

3. Batch Diffie–Hellman and Its Motivation

We consider a star-like graph representing, for example, a personal communications system. In this graph, the center, through the leaves (called "ports") agrees upon a session key with each of the nomadic units (called "portables"). The ports are part of the network, *and are trusted.* Each port holds a secret key that is used to authenticate the network in a transaction. As is shown at the end of the section on security, a port has sufficient data to factor the modulus. This implies that for RSA-based schemes, each port must be trusted at the highest level. In the case of DL based systems, we can still have large enough factors p and q such that DL modulo each of them is difficult (i.e., even after factorization). In practice, ports sometimes have poor physical protection, in which case it makes sense to do crypto-calculations on behalf of ports in some better protected central office. The port in this case is reduced to just an antenna + transmitter.

A naive system might assign a fixed public exponent e_i to port i. However, the center, when batch processing takes place, cannot know ahead of time which ports will be active (assuming, for example, that a portable communicates via the nearest port, and only when it requires some service from the center). Thus the use of this fixed assignment will require an untenable pre-processing phase (since precomputed binary trees will be required for all possible combinations of ports). Instead, we can precompute a tree for a group of exponents associated with the center (not the ports) and assign these exponents to individual key-agreement transactions dynamically, upon service request by a portable.

A representative RSA-based key agreement protocol is one where each of the parties picks a random number, encrypts it with the counterpart's public key, and sends it to the counterpart. The parties decrypt the received cryptograms, and use some combination of the two random numbers as a session key. Given the constraint of dynamic exponent assignment (described in the previous paragraph), it is not possible for a portable to obtain the center's public key (which for RSA includes modulus and exponent) until after the portable has requested service, because this is when the "batch" would be defined and exponents assigned. Thus, a batch implementation of this protocol would allow batch processing to begin only after the last encrypted message is received at the center. This would mean that batch processing would incur a delay which is dependent upon the response time of the slowest portable in the batch. (For example, a noisy radio link to a single portable could degrade the key-agreement response times of the network to other portables.) This property is most likely unacceptable in any real-time system (e.g., the users are waiting to make calls). While this delay is asymptotically immaterial (a constant), in practice it is significant.

A Diffie–Hellman key agreement scheme with comparable security does not suffer from such a delay. When a portable signals its request for key agreement, it can accompany the request with sufficient data for the center to begin batch processing.

We summarize the difference between RSA- and DH-based batch processing in this context as follows: In RSA the process involves: Portable contacts a port, and and requests service, port is assigned a public key, and sends it to the portable, and finally *maybe after delays* the port sends a message encrypted with that public key to the port, who sends it to the center for batch processing. With a DH-based scheme the portable sends the request together with the crypto-message (we cannot call it cryptogram) to the port, and then the port is assigned a public key. Batch processing starts as soon as enough requests have arrived (and no later than some small upper bound). The above *delay incurred in the RSA scheme is absent in the DH scheme.*

The current system may also have application in Discrete-log based e-cash systems, where a bank can process many e-coins in parallel, and in DL-based key escrow systems. The gain factor of $n/\log^2(n)$ is becoming now more significant as we move from $n = 512$ to $n = 1024$ and beyond.

The following is a basic DH scheme, as was first published in [3]. Let (S_i, P_i) be the secret and public keys, respectively, of portable i, $i = 1, \dots, n$, and let (S'_j, P'_j) be the secret and public keys dynamically assigned to port j. The central facility is trusted by all ports. For simplicity we describe here Batch DH for the simplest DH system.

In the basic Diffie–Hellman scheme there is some prime modulus, N, common to the whole system, and some primitive element of $GF(N)$, denoted α, such that $(\forall i)[P_i \equiv \alpha^{S_i} \bmod N]$, 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 (in which we refer to the system as CDH) [9], [7], and most of the time we need key agreement systems that authenticate the users, and are dynamic [10].

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. The base element is created as described in Lemma 1 (see section on security).
- Second, the secret key of each port j (or the jth member of a group of secret keys available for dynamic assignment at the center), 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_j's must be relatively prime in pairs.

4. Security

The following lemma is well known. As before, let $p - 1 = 2p'$, and $q - 1 = 2q'$, where p' and q' are large primes.

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 $\lambda(N) = (p-1)(q-1)/2$ (for our choice of p and q). We denote this cyclic subgroup M.*

Comment. In fact, for the above choice of p and q, if we choose γ at random with uniform distribution over Z_N^*, we get a generator for M with probability $\approx 1/8$: If we pick γ with uniform distribution in Z_N^*, then with probability $1/2$ it is in M, with probability $\approx 1/2$, each of $\mathrm{ord}_p(\gamma) = p - 1$, and $\mathrm{ord}_q(\gamma) = q - 1$ (there are $\varphi(q-1)$ different generators of Z_q^*, see, e.g., Proposition II.1.2 of [5], so for $q - 1 = 2q'$ we have $\varphi(q-1) = q' - 1$, which means about half the elements are generators).

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 restricted CDH. This follows from Lemma 2, and the corollary which immediately follows it. RSA is usually used with short public exponents. If such RSA was easily breakable over M, then it would not be considered secure, since M covers about half the instances.

Lemma 2 says that the Composite Diffie–Hellman key agreement scheme is at least as hard to break on the average as RSA for messages in M, to within a small constant.

The basic problems are:

The RSA cracking problem:

Input. (N, e, c) such that e is invertible mod $\lambda(N)$ and c is invertible mod N.
Output. m between 1 and $N - 1$ such that $c \equiv m^e \bmod N$.
The solution is uniquely defined by $m \equiv c^x \bmod N$ with $x \equiv e^{-1} \bmod \lambda(N)$.

The D–H cracking problem is:

Input. N, a, b, c such that b and c are powers of a mod N.
Output. $a^{xy} \bmod N$ such that $b \equiv a^x \bmod N$ and $c \equiv a^y \bmod N$. x and y are uniquely defined mod $\mathrm{order}_N(a)$, and so is xy. Thus, the solution is uniquely defined.

When N is a composite the DH system is called *Composite DH* (*CDH*). A CDH system is *restricted* if a (and hence b, c) are restricted to the multiplicative subgroup M, as defined above. Similarly, RSA is *restricted* if the message, m, and hence the cryptogram, c, are in M.

Lemma 2. *Suppose there exists an algorithm that breaks the restricted Composite Diffie–Hellman key agreement scheme, with modulus $N = pq$, $p-1 = 2p'$, $q-1 = 2q'$,* *where p' and q' are primes, in time $t(n)$, with probability $q(n)$, where $n = \log_2(N)$ is a security parameter. Then there exists an algorithm to break restricted RSA in time $t(n)$ with probability $q(n)/8$.*

Proof. Assume there exists an oracle AL, such that for all N, A, B, C, where each of A, B, C is in M. $AL(N, A, B, C) \equiv A^{xy} \bmod N$, where $B \equiv A^x \bmod N$, and $C \equiv A^y \bmod N$, and let $e \equiv x^{-1} \bmod \lambda(N)$.

Given a restricted RSA cracking problem, we use oracle AL to solve it, as follows:

1. Pick a random $\gamma \in Z_N^*$ (with probability 1/8 it is a generator of M, see Comment after Lemma 1).
2. Compute $\beta \equiv \gamma^e \bmod N$.
3. Call oracle $AL(N, \beta, \gamma, c)$, where c is the cryptogram given in the RSA problem.

Suppose γ which was picked in the first step is a generator of M. Then 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. Success probability of this reduction is $q(n)/8$ from Step 1. □

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 inverse. RSA with short public key reduces to CDH with exactly one short inverse by the same construction.*

Lemma 3. *A Composite Diffie–Hellman key agreement scheme, in which exactly one of the exponents has a short inverse ($< O(\log(N))$) 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 the RSA problem, where the public exponent e is short ($e < O(\log(N))$), i.e., $AL2(N, e, c) = m$, such that $m^e \equiv c \bmod N$. 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, such that $(a^x)^e \equiv a \bmod N$ (since e is short, exhaustive search is feasible).
2. Call oracle $AL2(N, e, a^y)$.

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. □

The following lemma shows that a port has sufficient data to factor the modulus. As mentioned before, this is devastating for RSA-based systems, but for DL-based systems, if we choose, say, p and q of size 500 bits each, then DL is still hard per today's best published algorithms.

Lemma 4. *Suppose $N = pq$, p and q are large primes, such that $p - 1 = 2p'$, and $q - 1 = 2q'$, where p' and q' are primes, $n = \log_2(N)$, and $xy \equiv 1 \bmod \lambda(N)$, such that x is of size $\log(n)$ bits. Then there exists a polynomial time algorithm (in n) that given y factors N.*

Proof. Try all candidates x of size $\leq \log(n)$ bits. For each compute $xy - 1$. This is a small multiple of $\lambda(N)$ for the right x, so one can find all small factors of $xy - 1$, the rest is $p'q'$, which yields N's factorization. □

The next lemma states that if we use a CDH where both parties have secret exponents with short inverses, then a complete outsider can effectively calculate the session key.

Lemma 5. *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, p. 324) the adversary can break the system. First he finds a and b such that $ae_i + be_j = 1$, using the extended Euclid 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 $\alpha^{xy} \equiv (\alpha^x)^b \cdot (\alpha^y)^a \bmod N$. □

Of course, the reduction of Lemma 2 does not hold when both exponents have short inverses (we remind the reader that portables use secret exponents with long inverses).

Recently Don Coppersmith found that if two similar messages are RSA encrypted with the same small public exponent, then the messages could be efficiently found (e.g., if they are 90% similar, then exponent 3 is too small). This is irrelevant to our scheme, or to the evidence of security given above (the reductions are from RSA encryption with one message).

Conclusions

Batch–RSA can significantly slash computation load in centralized servers that must do many large exponentiations simultaneously, but if the purpose is establishing distinct key agreements with each of many clients (such as in cellular communication) it incurs delays, where the slowest client delays the whole batch. We show that similar mathematical ideas apply to DH-based schemes, that are free of those delays. We prove that the new scheme is as secure as RSA encryption with short public exponent, given one cryptogram.

Acknowledgments

We wish to thank Rafi Heiman and Arjen K. Lenstra for reviewing earlier versions of this paper, and for their many helpful comments. Special thanks go to the anonymous referee of *Journal of Cryptology* for most educating discussions, which greatly improved this paper.

References

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

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

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

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

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

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

[7] K. S. McCurley, A key distribution system equivalent to factoring, *J. Cryptology*, vol. 1, no. 2, pp. 95–105, 1988.

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

[9] Z. Shmuely, Composite Diffie–Hellman public-key generating systems are hard to break, Technical Report 356, Computer Science Department, Technion, Feb. 1985.

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