(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 December 2001 (20.12.2001)

PCT

(10) International Publication Number
WO 01/97443 A2

(51) International Patent Classification⁷: H04L 9/30

(21) International Application Number: PCT/US01/18878

(22) International Filing Date: 12 June 2001 (12.06.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/211,023	12 June 2000 (12.06.2000)	US
60/211,031	12 June 2000 (12.06.2000)	US
Not furnished	8 June 2001 (08.06.2001)	US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:

US	60/211,023 (CIP)
Filed on	12 June 2000 (12.06.2000)
US	60/211,031 (CIP)
Filed on	12 June 2000 (12.06.2000)
US	Not furnished (CIP)
Filed on	8 June 2001 (08.06.2001)

(71) Applicant *(for all designated States except US)*: **INGRIAN SYSTEMS, INC.** [US/US]; 3071 Edison Way, Redwood City, CA 94063 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **SHACHAM, Hovav** [US/US]; 3826 Mumford Place, Palo Alto, CA 94306 (US). **BONEH, Dan** [IL/US]; 3349 Louis Road, Palo Alto, CA 94303 (US). **BERI, Sanjay** [CA/US]; 562 Kendall Avenue, Unit #4, Palo Alto, CA 94306 (US).

(74) Agent: **GREGORY, Richard, L., Jr.**; Wilson Sonsini Goodrich & Rosati, 650 Page Mill Road, Palo Alto, CA 94304-1050 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

(54) Title: METHOD AND APPARATUS FOR ENHANCING NETWORK SECURITY PROTECTION SERVER PERFORMANCE

(57) **Abstract:** Presented is a method and system for improving the efficiency of network security protections communication protocols such as Secure Socket Layer ("SSL") using enhanced Rivest-Shamir-Adleman ("RSA") encryption and decryption techniques. During the establishment of the initial handshake of SSL communications, where a client is coupled to a server, the server generates a RSA public/private key pair. The public key is formed using two distinct prime numbers. By reducing the size of these prime numbers and arriving at the decrypted message using the Chinese Remainder Theorem, the efficiency of establishing a secure communications session is increased. Likewise if during generation of the public key, the prime numbers possess a mathematical relationship to the public key such that the prime numbers on the order of a third of the size of the public key then the efficiency of establishing the initial handshake is again improved.

WO 01/97443 A2

METHOD AND APPARATUS FOR ENHANCING NETWORK SECURITY PROTECTION SERVER PERFORMANCE

FIELD OF THE INVENTION

The claimed invention relates to the field of secure communications. More particularly it relates to improving the efficiency of secure network communications.

BACKGROUND OF THE INVENTION

Many network transactions today require secure communications. To establish a secure communication link protocols such as Secure Socket Layer ("SSL") and Transport Layer Security ("TLS") must be accomplished. Today SSL is the most widely deployed protocol for securing communication on the World Wide Web ("WWW"). The protocol is used by most E-commerce and financial web sites as it guarantees privacy and authenticity of information exchanged between a web server and a web browser. Currently, the number of web sites using SSL to secure web traffic is growing at a phenomenal rate and as the services provided on the World Wide Web continue to expand so will the need for security using SSL.

Unfortunately, neither SSL or TLS are cheap. A number of studies have shown that web servers using the SSL protocol perform far worse than web servers that do not encrypt web traffic. In particular, a web server using SSL can handle 30 to 50 times fewer transactions per second than a web server using clear-text communication only can. The exact transaction performance degradation depends on the type of web server used by the site. To overcome this degradation web sites using secure connections typically buy significantly more hardware in order to provide a reasonable response time to their customers.

Web sites often use one of two techniques to overcome security's impact on performance. The first method, as indicated above, is to deploy more machines at the web site and load balance connections across these machines. This is problematic since more machines are harder to administer. In addition, mean time between failures decreases significantly. The other solution is to install a hardware acceleration card inside the web server. The card handles most of the secure protocol workload thus enabling the web server to focus on its regular tasks. Accelerator cards are available from a number of vendors and while these cards reduce the penalty of using secure protocols, they are relatively expensive and are non-trivial to configure. Thus there is a need to quickly establish secure transactions at a lower cost.

SUMMARY OF THE INVENTION

A method and apparatus for enhancing security protection server performance in a computer network is provided when a web browser first connects to a web server using secure protocols, the browser and server execute an initial handshake protocol. The outcome of this protocol is a session encryption key and a session integrity key. These keys are only known to the web server and web browser, and establish a secure session.

Once session keys are established, the browser and server begin exchanging data. The data is encrypted using the session encryption key and protected from tampering using the session integrity key. When the browser and server are done exchanging data the connection between them is closed. This process begins when the web browser connects to the web server and sends a client-hello message. Soon after receiving the message, the web server responds with a server-hello message. This message contains the server's public key certificate that informs the client of the server's Rivest-Shamir-Adleman algorithm ("RSA") public key. Having received the public key, the browser picks a random 48-byte string, *R*, and encrypts it using the key. Letting *C* be the resulting cipher-text of the string R, the web browser then sends a client-key-exchange message containing *C*. The 48-byte string *R* is called the pre-master- secret. Upon receiving the message, from the browser, the web server uses its RSA private key to decrypt *C* and thus learns *R*. Both the browser and server then use *R* and some other common information to derive the session keys. With the session keys established, encrypted message can be sent between the browser and server with impunity.

The decryption of the encrypted string, R, is the expensive part of the initial handshake. An RSA public key is made of two integers $\langle N, e \rangle$. In an embodiment $N = pq$ is the product of two large primes and is typically 1024 bits long. The value e is called the encryption exponent and is typically some small number such as $e = 65537$. Both N and e are embedded in the server's public key certificate. The RSA private key is simply an integer d satisfying $e \cdot d = 1 \bmod (p-1)(q-1)$. Given an RSA cipher-text C, the web server decrypts C by using its private key to compute $C^d \bmod N$ that reveals the plain-text message, R. Since both d and N are large numbers (each 1024 bits long) this computation takes some effort.

At a later time, the browser may reconnect to the same web server. When this happens the browser and server executes the resume handshake protocol. This protocol causes both server and browser to reuse the session keys established during the initial handshake saving invaluable resources. All application data is then encrypted and protected using the previously established session keys.

Of the three phases, the initial handshake is often the reason why secure connections degrade web server performance. During this initial handshake the server performs an RSA decryption or an RSA signature generation. Both operations are relatively expensive and the high cost of the initial handshake is the main reason for supporting the resume handshake protocol. The resume handshake protocol tries to alleviate the cost of the initial handshake by reusing previously negotiated keys across multiple connections. However, in the web environment, where new users constantly connect to the web server, the expensive initial handshake must be executed over and over again at a high frequency. Hence, the need for reducing the cost of the initial handshake protocols.

One embodiment presents an implementation of batch RSA in an SSL web server while other embodiments present substantial improvements to the basic batch RSA decryption algorithms. These embodiments show how to reduce the number of inversions in the batch tree to a single inversion. Another embodiment further speeds up the process by proper use of the Chinese

Remainder Theorem ("CRT") and simultaneous multiple exponentiation. While the Secure Socket Layer ("SSL") protocol is a widely utilized technique for establishing a secure network connection, it should be understood that the present invention can be applied to the establishment of any secure network based connection using a plurality of protocols.

A different embodiment entails architecture for building a batching secure web server. The architecture in this embodiment is based on using a batching server process that functions as a fast decryption oracle for the main web server processes. The batching server process includes a scheduling algorithm to determine which subset of pending requests to batch.

Yet other embodiments improve the performance by reducing the handshake work on the server per connection. One technique supports web browsers that deal with a large encryption exponent in the server's certificate, while another approach supports any browser.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example in the following figures in which like references indicate similar elements. The following figures disclose various embodiments of the claimed invention for purposes of illustration only and are not intended to limit the scope of the claimed invention.

Figure 1 is a flow diagram of the initial handshake between a web server and a client of an embodiment.

Figure 2 is a flow diagram for increasing efficiency of the initial handshake process by utilizing cheap keys of an embodiment.

Figure 3 is a flow diagram for increasing efficiency of the initial encryption handshake by utilizing square keys in an embodiment.

Figure 4 is a block diagram of an embodiment of a network system for improving secure communications.

Figure 5 is a flow diagram for managing multiple certificates using a batching architecture of an embodiment.

Figure 6 is a flow diagram of batching encrypted messages prior to decryption of an embodiment.

DETAILED DESCRIPTION

The establishment of a secure network connection can be improved by altering the steps of the initial handshake. One embodiment for the improvement to the handshake protocol focuses on how the web server generates its RSA key and how it obtains a certificate for its public key. By altering how the browser uses the server's public key to encrypt a plain-text *R*, and how the web server uses its private key to decrypt the resulting cipher-text *C*, this embodiment provides significant improvements to Secure Socket Layer ("SSL") communications. While the Secure Socket Layer protocol is a widely utilized technique for establishing a secure network connection, it should be understood that the techniques described herein can be applied to the establishment of any secure network-based connection using any number of protocols.

The general process in establishing a Secure Socket Layer communication between a browser or client and a server or host is depicted in **Figure 1**. The process begins with a request from the browser to establish a secure session 110. The client forms a hello message requesting a public key and transmits the message to the server 114. Upon receiving the client-hello message, the web server responds with a server-hello message containing a public key 118. The public key is one half of a public / private key pair. While the server transmits the public key back to the browser the server keeps the private key. Once the client receives the public key 122 a random number R is generated 126. This random number is the session key. The client encrypts R by using the private key that it received from the server 132. With the number R encrypted, the client sends the cipher-text to the web-server 138. Upon receiving the cipher-text 142 the web server user the private key portion of the public / private key pair to decrypt the cipher-text 146. With both the client and the server possessing the session key R, a new encrypted secure socket layer session 160 is established using R as the session key 158. This session is truly encrypted since only the client and the web server possess the session key for encryption and decryption.

In one embodiment that improves the establishment of a secure connection a server generates an RSA public/private key pair by generating two distinct n-bit primes p and q and computing $N = pq$. While N can be of any arbitrary size, assume for simplicity that N is 1024 bits long and let $w = \gcd(p - 1, q - 1)$ where gcd is the greatest common divisor. The server then picks two random k-bit values r_1, r_2 such that $\gcd(r_1, p - 1) = 1$, $\gcd(r_2, q - 1) = 1$, and $r_1 = r_2 \bmod w$. Typically k falls in the range of 160 -512 bits in size. Although other larger values are also acceptable, k is minimized to enhance performance. The server then computes d such that $d = r_1 \bmod p - 1$ and $d = r_2 \bmod q - 1$. Having computed d, e' is found by solving the equation $e' = d^{-1} \bmod \varphi(N)$ resulting in the public key being $\langle N, e'\rangle$ and the private key d, which is a function of two random numbers, $\langle r_1, r_2\rangle$.

The server then sends the public key to a Certificate Authority (CA). The CA returns a public key certificate for this public key even though e' is very large, namely on the order of N. This is unlike standard RSA public key certificates that use a small value of e, e.g. $e = 65537$. Consequently, the CA must be willing to generate certificates for such keys.

To find d the Chinese Remainder Theorem is typically used. Unfortunately, $p - 1$ and $q - 1$ are not relatively prime (they are both even) and consequently the theorem does not apply. However, by letting $w = \gcd(p - 1, q - 1)$, knowing that $\frac{p-1}{w}$ and $\frac{q-1}{w}$ are relatively prime, and recalling that $r_1 = r_2 = a \bmod w$, the CRT can be used to find an element d' such that

$$\begin{aligned} d' &= \frac{r_1 - a}{w}\left(\bmod \frac{p-1}{w}\right) \\ d' &= \frac{r_2 - a}{w}\left(\bmod \frac{q-1}{w}\right) \end{aligned}.$$

Observing that the required d is simply $d = w \cdot d' + a$ and indeed, $d = r_1 \bmod p - 1$ and $d = r_2 \bmod q - 1$, if w is large, the requirement that $r_1 = r_2 \bmod w$ reduces the entropy of the private key. For this reason it is desirable to ensure that w is small and one embodiment lets $w = 2$, or namely that $\gcd(p - 1, q - 1) = 2$. Recall that $\gcd(r_1, p - 1) = 1$ and $\gcd(r_2, q - 1) = 1$.

It follows that $\gcd(d, p-1) = 1$ and $\gcd(d, q-1) = 1$ and consequently $\gcd(d, (p-1)(q-1)) = 1$. Hence, d is invertible modulo $\varphi(N) = (p-1)(q-1)$.

The web browser obtains the server's public key certificate from the server-hello message. In this embodiment, the certificate contains the server's public key $\langle N, e\rangle$. The web browser encrypts the pre-master-secret R using this public key in exactly the same way it encrypts using a normal RSA key. Hence, there is no need to modify any of the browser's software. The only issue is that since e' is much larger than e in a normal RSA key, the browser must be willing to accept such public keys.

When the web server receives the cipher-text C from the web browser the web server then uses the server's private key, (r_1, r_2), to decrypt C. To accomplish this the server computes $R_1' = C^{r_1} \bmod p$ and $R_2' = C^{r_2} \bmod q$. Using CRT the server then computes an $R \in \mathbb{Z}_N$ such that $R = R_1' \bmod p$ and $R = R_2' \bmod q$, noting that $R = C^d \bmod N$. Hence, the resulting R is a proper decryption of C.

Decryption using a standard RSA public key is completed with C^d mod N using the CRT. Typically $R_1 = C^{(d \bmod p - 1)} \bmod p$ and $R_2 = C^{(d \bmod q - 1)} \bmod q$ is first computed and then the CRT is applied to R_1, R_2 to obtain $R \bmod N$. Note that the exponents $d \bmod p - 1$ and $d \bmod q - 1$ are typically as large as p and q, namely 512 bits each. Hence, to generate the signature the server must compute one exponentiation modulo p and one exponentiation modulo q. When N is 1024 bits, the server does two full exponentiations modulo 512-bit numbers.

In one embodiment, the server computes R_1', R_2' and then applies CRT to R_1', R_2'. As in normal RSA, the bulk of the work is in computing R_1', R_2'. However, computing R_1' requires raising C to the power of r_1, which is minimized. Since the time that modular exponentiation takes is linear in time to the size of the exponent, computing R_1' takes approximately one third the work and one third of the time of raising C to the power of a 512 bit exponent. Hence, computing R_1' takes one third the work of computing R_1. Therefore, during the entire decryption process the server does approximately one third the work as in a normal SSL handshake.

To illustrate the implementation of this embodiment suppose Eve is an eavesdropper that listens on the network while the handshake protocol is taking place. Eve sees the server's public key $\langle N, e'\rangle$ and the encrypted pre-master-secret C. Suppose $r_1 < r_2$. It can be shown that an adversary who has $\langle N, e', C\rangle$ can mount an attack on the system that runs in time $O\left(\sqrt{r_1}\log r_1\right)$.

Let $\langle N, e'\rangle$ be an RSA public key with $N = pq$ and let $d \in \mathbb{Z}$ be the corresponding RSA private key satisfying $d = r_1$, mod $p - 1$ and $d = r_2$ mod $q - 1$ with $r_1 < r_2$. If r₁ is m bits long and it is assumed that $r_1 \neq r_2 \bmod 2^{m/2}$, then given $\langle N, e'\rangle$ an adversary can expose the private key d in time $O\left(\sqrt{r_1}\log r_1\right)$. One skilled in the art knows that $e' = (r_1)^{-1} \bmod (p-1)$. But, suppose r_1 is m-bits long. If $r_1 = A \cdot 2^{m/2} + B$ where A, B are in $[0, 2^{m/2}]$ and a random $g \in \mathbb{Z}_N$ is selected combined with the definition

$$G(X) = \prod_{i=0}^{2^{m/2}} \left(g^{e' \cdot 2^{m/2} \cdot i} \cdot X - g \right),$$

then if follows that $G(g^{e' \cdot B}) = 0 \bmod p$. This occurs since one of the products above is

$$\left(g^{e' \cdot 2^{m/2} \cdot A} \cdot g^{e' \cdot B} - g \right) = g^{e' r_1} - g = 0 \pmod{p}.$$

Since $r_1 \neq r_2 \bmod 2^{m/2}$, it can be shown that $G(g^{e' \cdot B}) \neq 0 \bmod q$. Hence, gcd $(N, G(g^{e' \cdot B}))$ gives a non-trivial factor of N. Hence, if $G(x) \bmod N$ is evaluated at $x = g^{e' \cdot j}$ for $j = 0, \ldots, 2^{m/2}$ at least one of the values will expose the factorization of N. Evaluating a polynomial of degree $2^{m/2}$ at $2^{m/2}$ values can be done in time $2^{m/2} \cdot m/2$ using Fast Fourier Transform methods. This algorithm requires $\tilde{O}\left(2^{m/2}\right)$ space. Hence, in time at most $O\left(\sqrt{r_1}\log r_1\right)$ we can factor N. Thus in order to obtain security of 2^{80}, both r_1 and r_2 must be at least 160 bits long.

Figure 2 is a flow diagram for improving secure socket layer communications of an embodiment by altering the public / private key pair. In operation, the server generates an RSA public / private key pair initiating a normal initial handshake protocol 210. At this point the server generates two distinct prime numbers 215 and takes the product of the numbers to produce the N component of the public key 220. Similarly, the server picks two random values to create the private key 225. Using the prime numbers, 215, and the random values of the private key 225 the server computes the value d, 230, and correspondingly the value e' 235. The result is a new public / private key pair 240 that the client uses to encrypt the pre-master-secret R 250. Once R has been encrypted with the new public key and transmitted to the server as cipher-text C, the server uses it private key to decrypt the pre-master-secret 260. Once R_1 and R_2 have been determined 265 they are combined to find R 270. Having the value of the pre-master-secret intact, the server and client can establish a secure session 280.

A further embodiment dealing with the handshake protocol reduces the work per connection on the web server by a factor of two. This embodiment works with all existing browsers. As before, the embodiment is illustrated by describing how the web server generates its RSA key and obtains a certificate for its public key. This embodiment continues in describing how the browser uses the server's public key to encrypt a plain-text R, and the server uses its private key to decrypt the resulting cipher-text C.

In this embodiment the server generates an RSA public/private key pair by generating two distinct n-bit primes p and q such that the size of each distinct prime number is on the order of one third of the size of N. Using this relationship the server computes N' as $N' = p^2 \cdot q$. The relationship between the prime numbers and N is dependent on the power by which one of the prime number is raised. For example if one of the prime numbers was raised to the fourth power the prime numbers would be on the order of one fifth the size of N. The exponent of at least one of the prime numbers must be greater than one. While clearly N' can be of arbitrary size, assume, in the situation where p is raised to the power of two and q is raised to the power of one, that N' is 1024 bits long, and hence p and q are 341 bits each instead of the typical 512 bits. The server uses the same e used in standard RSA public keys, namely $e = 65537$ as long as $\gcd(e, (p-1)(q-1)) = 1$. The server then computes $d = e^{-1} \bmod (p-1)(q-1)$ as well as $r_1 = d \bmod p - 1$ and $r_2 = d \bmod q - 1$. With the public key being $\langle N', e\rangle$ and the private key being d, which is a function of (r_1, r_2), the server sends the public key, $\langle N', e\rangle$, to a Certificate Authority (CA) and the CA returns a public key certificate. The public key in this case cannot be distinguished from a standard RSA public key.

The web browser obtains the server's public key certificate from the server-hello message. The certificate contains the server's public key $\langle N', e\rangle$. The web browser encrypts the pre-master-secret R using this public key in exactly the same way it encrypts using a normal RSA key.

When the web server receives the cipher-text C from the web browser the web server decrypts C by computing $R_1' = C^{r_1} \bmod p$ and $R_2' = C^{r_2} \bmod q$. Note that $(R_1')^e = C \bmod p$ and $(R'_2)^e = C \bmod q$. Lifting the server constructs an R_1'' such that $(R_1'')^e = C \bmod p^2$. More precisely, the server computes

$$R_1'' = R_1' - \frac{(R_1')^e - C}{e \cdot (R_1')^{e-1}} \quad \left(\bmod p^2\right).$$

Using CRT, the server computes an $R \in \mathbb{Z}_N$ such that $R = R_1'' \bmod p^2$ and $R = R_2' \bmod q$ noting that $R = C^d \bmod N$. Hence, the resulting R is a proper decryption of C. Recall that when N is 1024 bits, the server typically does two full exponentiations modulo 512-bit numbers. In this embodiment the alteration of the multiplicity of the roots is compensated by the lifting mechanism.

In this embodiment the server computes R_1', R_2', R_1'' and then applies CRT to R_1'', R_2'. The bulk of the work is in computing R_1', R_2', R_1'' but computing R_1' requires a full exponentiation modulo a 341-bit prime rather than a 512-bit prime. The same holds for R_2'. Hence in this embodiment, computing R_1', R_2' takes approximately half the time of computing R_1, R_2. Furthermore, computing R_1'' from R_1' only requires a modular inversion modulo p^2. This takes little time when compared with the exponentiations for computing R_1', R_2'. Hence, using this embodiment the handshake takes approximately half the work of a normal handshake on the server.

Some accelerator cards do not provide support for modular inversion. As a result, the inversion is preformed using an exponentiation. This is done by observing that for any $x \in Z_p^*$ the inverse of *x* is given by:

$$x^{-1} = x^{p^2-p-1} \pmod{p^2}.$$

Unfortunately, using an exponentiation to do the inversion hurts performance. As discussed herein a better embodiment for inversion in this case is batching. One can invert two numbers $x_1, x_2 \in Z_p^*$ as a batch faster than inverting the two numbers separately. To do so use the fact that

$$x_1^{-1} = x_2 \cdot (x_1 x_2)^{-1} \text{ and } x_2^{-1} = x_1 \cdot (x_1 x_2)^{-1} \pmod{p^2}.$$

Hence, at the cost of inverting $x_1 \cdot x_2$ it is possible to invert both x_1 and x_2. This embodiment shows that an inversion of *k* elements $x_1, \ldots, x_k \in Z_p^*$ is at the cost of one inversion and $k \log_2 k$ multiplications. Thus, the amortized cost of a single inversion is $1/k$ of an exponentiation plus $\log_2 k$ multiplications.

To take advantage of batched inversion in the SSL handshake a number of instances of the handshake protocol are collected from among different users and the inversion is preformed on all handshakes as a batch. As a result, the amortized total number of exponentiations per handshake is $2 + \frac{1}{k}$. This approximately gives a factor of two improvement in the handshake work on the server as compared to the normal handshake protocol.

The security of the improved handshake protocol depends on the difficulty of factoring integers of the form $N = p^2 \cdot q$. When 1024 bit keys are used the fastest factoring algorithms (i.e. the number field sieve) cannot take advantage of the special structure of *N*. Similarly, *p* and *q* are well beyond the capabilities of the Elliptic Curve Method (ECM).

Figure 3 is a flow diagram for modifying the public key of an embodiment to facilitate an improvement in secure socket layer communication. As in other embodiments, the process begins with the servers generation of a RSA public / private key pair 310. In this embodiment, the public key is modified. The web server generates two distinct prime numbers 312 and computes a new N' 318. Using the same exponent 320 the server computes the value d 322 which it uses to find the private key 328. The result is a pubic / private key combination 330 that the sever then sends to the client for the encryption of the pre-master-secret 340. Once the server receives the encrypted pre-master-secret, R, from the client 350 the server decrypts R 360 by computing R1 362 and R2 368 and combining the results 370. Once R has been determined the client can establish a secure session with the client using the new session key 380.

The establishment of a secure connection between a server and a browser can also be improved by batching the initial SSL handshakes on the web server. In one embodiment the web server waits until it receives b handshake requests from b different clients. It treats these b handshakes as a batch, or set of handshakes, and performs the necessary computations for all b handshakes at once. Results show that, for $b = 4$, batching the SSL handshakes in this way results in a factor of 2.5 speedup over doing the b handshakes sequentially, without requiring any additional hardware.

One embodiment improves upon a technique developed by Fiat for batch RSA decryption. Fiat suggested that one could decrypt multiple RSA cipher-texts as a batch faster than decrypting them one by one. Unfortunately, experiments show that Fiat's basic algorithm, naively implemented, does not give much improvement for key sizes commonly used in initial secure handshakes.

A batching web server must manage multiple public key certificates. Consequently, a batching web server must employ a scheduling algorithm that assigns certificates to incoming connections, and picks batches from pending requests, so as to optimize server performance.

To encrypt a message M using an RSA public key $\langle N, e\rangle$, the message M is formatted to obtain an integer X in $\{1, \ldots, N\}$. This formatting is often done using the PKCS1 standard. The cipher-text is then computed as $C = X^e \bmod N$. This process occurs during the initial stages of the initial handshake between a client and server when attempting to create a secure connection.

To decrypt a cipher-text C the web server uses its private key d to compute the e'th root of C in $\mathbb{Z}_N$. The e th root of C is given by $C^d \bmod N$ as previously noted. Since both d and N are large numbers (each 1024 bits long) this is a lengthy computation on the web server. It is noted that d must be taken as a large number (i.e., on the order of N) since otherwise the RSA system is insecure.

When using small public exponents, e_1 and e_2, which are components of the public key, it is possible to decrypt two cipher-texts for approximately the price of one. Suppose v_1 is a cipher-text obtained by encrypting using the public key $\langle N, 3\rangle$. Similarly, imagine v_2 is a cipher-text obtained by encrypting using the public key $\langle N, 5\rangle$. To decrypt v_1 and v_2, computing $v_1^{1/3}$ and $v_1^{1/5}$ mod N are made by setting $A = (v_1^5 \cdot v_2^3)^{1/15}$ it can be shown that

$$v_1^{1/3} = \frac{A^{10}}{v_1^3 \cdot v_2^2} \text{ and } v_2^{1/5} = \frac{A^6}{v_1^2 \cdot v_2}.$$

Hence, at the cost of computing a single 15th root both v_1 and v_2 can be decrypted.

This batching technique is most useful when the public exponents e_1 and e_2 are very small (e.g., 3 and 5). Otherwise, the extra arithmetic required can be expensive. Also, only cipher-texts encrypted using distinct public exponents can be batch decrypted. Indeed, it can be shown that it is not possible to batch when the same public key is used. That is, it is not possible to batch the computation of $v_1^{1/3}$ and $v_2^{1/3}$.

This observation to the decryption of a batch of b RSA cipher-texts can be generalized. In one embodiment there are b distinct and pairwise relatively prime public keys $e_1, \ldots, e_b$, all sharing a common modulus $N = pq$. Furthermore, assume there are b encrypted messages, $v_1, \ldots, v_b$, one encrypted with each key, that are desirable to decrypt simultaneously, to obtain the plain-texts $m_i = v_i^{1/e_i}$.

The batch process is implemented around a complete binary tree with b leaves, possessing the additional property that every inner node has two children. In one embodiment the notation is biased towards expressing locally recursive algorithms: Values are percolated up and down the tree. With one exception, quantities subscripted by L or R refer to the corresponding value of the left or right child of the node, respectively. For example, m is the value of m at a node; m_R is the value of m at that node's right child and so forth.

Certain values necessary to batching depend on the particular placement of keys in the tree and may be pre-computed and reused for multiple batches. Pre-computed values in the batch tree are denoted with capital letters, and values that are computed in a particular decryption are denoted with lower-case letters.

The batching algorithm consists of three phases: an upward-percolation phase, an exponentiation phase, and a downward-percolation phase. In the upward-percolation phase, the individual encrypted messages v_i are combined to form, at the root of the batch tree, the value

$$v = \prod_{i=1}^{b} v_i^{e/e_i}\text{, where } e = \prod_{i=1}^{b} e_i .$$

In preparation, assign to each leaf node a public exponent: $E \leftarrow e_i$. Each inner node then has its E computed as the product of those of its children: $E \leftarrow E_L \cdot E_R$. The root node's E will be equal to e, the product of all the public exponents. Each encrypted message v_i is placed (as v) in the leaf node labeled with its corresponding e_i. The v's are percolated up the tree using the following recursive step, applied at each inner node:

$$v \leftarrow v_L^{E_R} \cdot v_R^{E_L} .$$

At the completion of the upward-percolation phase, the root node contains $v = \prod_{i=1}^{b} v_i^{e/e_i}$. In the exponentiation phase, the e th root of this v is extracted. Here, the knowledge of factorization of N is required. The exponentiation yields $v^{1/e} = \prod_{i=1}^{b} v_i^{1/e_i}$, which is stored as m in the root node.

In the downward-percolation phase, the intent is to break up the product m into its constituent subproducts m_L and m_R, and, eventually, into the decrypted messages m_i at the leaves. At each inner node an X is chosen satisfying the following simultaneous congruencies:

$$X = 0 \pmod{E_L} \qquad\qquad X = 1 \pmod{E_R}.$$

The value X is constructed using the Chinese Remainder Theorem ("CRT"). Two further numbers, X_L and X_R, are defined at each node as follows:

$$X_L = X/E_L \qquad\qquad X_R = (X - 1)/E_R.$$

Both divisions are done over the integers. (There is a slight infelicity in the naming here: X_L and X_R are not the same as the X's of the node's left and right children, as implied by the use of the L and R subscripts, but separate values.)

The values of X, X_L, and X_R are such that, at each inner node, m^X equals $v_L^{X_L} \cdot v_R^{X_R} \cdot m_R$. This immediately suggests the recursive step used in downward-percolation:

$$m_R \leftarrow m^X / \left(v_L^{X_L} \cdot v_R^{X_R}\right) \qquad\qquad m_L \leftarrow m / m_R .$$

At the end of the downward-percolation process, each leaf's m contains the decryption of the v placed there originally. Only one large (full-size) exponentiation is needed, instead of b of them. In addition, the process requires a total of 4 small exponentiations, 2 inversions, and 4 multiplications at each of the b - 1 inner nodes.

Basic batch RSA is fast with very large moduli, but may not provide a significant speed improvement for common sized moduli. This is because batching is essentially a tradeoff. Batching produces more auxiliary operations in exchange for fewer full-strength exponentiations.

Batching in an SSL-enabled web server focuses on key sizes generally employed on the web, e.g., $n = 1024$ bits. Furthermore, this embodiment also limits the batch size b to small numbers, on the order of $b = 4$, since collecting large batches can introduce unacceptable delay. For simplicity of analysis and implementation, the values of b are restricted to powers of 2.

Previous schemes perform two divisions at each internal node, for a total of $2b$ - 2 required modular inversions. Modular inversions are asymptotically faster than large modular exponentiations. In practice, however, modular inversions are costly. Indeed, the first implementation (with $b = 4$ and a 1024-bit modulus) spends more time doing the inversions than doing the large exponentiation at the root. Two embodiments, when combined, require only a single modular inversion throughout the algorithm with the cost of an additional $O(b)$ modular multiplication. This tradeoff gives a substantial running-time improvement.

The first embodiment is referred to herein as delayed division. An important realization about the downward-percolation phase is that the actual value of m for the internal nodes of the tree is consulted only for calculating m_L and m_R. An alternative representation of m that supports the calculation of m_L and m_R, and that can be evaluated at the leaves to yield m would do just as well.

This embodiment converts a modular division a/b to a "promise," $\langle a, b\rangle$. This promise can operate as though it were a number, and, can "force" getting its value by actually computing $b^{-1}a$. Operations on these promises work in a way similar to operations in projective coordinates as follows:

$$\begin{array}{cc} a/b=\langle a,b\rangle & \langle a,b\rangle^c=\langle a^c,b^c\rangle \\ c\cdot\langle a,b\rangle=\langle ac,b\rangle & \langle a,b\rangle\cdot\langle c,d\rangle=\langle ac,bd\rangle \\ \langle a,b\rangle/c=\langle a,bc\rangle & \langle a,b\rangle/\langle c,d\rangle=\langle ad,bc\rangle \end{array}.$$

Multiplication and exponentiation takes twice as much work had these promises not been utilized, but division can be computed without resort to modular inversion.

If, after the exponentiation at the root, the root m is expressed as a promise, $\mathbf{m} \leftarrow \langle m, 1\rangle$, this embodiment can easily convert the downward-percolation step to employ promises:

$$\mathbf{m}_R \leftarrow \mathbf{m}^X/\left(v_L^{X_L}\cdot v_R^{X_R}\right) \qquad \mathbf{m}_L \leftarrow \mathbf{m}/\mathbf{m}_R.$$

No internal inversions are required. The promises can be evaluated at the leaves to yield the decrypted messages.

Batching with promises uses b - 1 additional small exponentiations and b - 1 additional multiplications. This translates to one exponentiation and one multiplication at every inner node, saving $2(b - 1) - b = b - 2$ inversions. To further reduce the number of inversions, another embodiment uses batched divisions. When using delayed inversions one division is needed for every leaf of the batch tree. In the embodiment using batched divisions, these b divisions can be done at the cost of a single inversion with a few more multiplications.

As an example of this embodiment, invert three values x, y, and z. Continue by forming the partial product yz, xz, and xy and then form the total product xyz and invert it, yielding $(xyz)^{-1}$. With these values, calculate all the inverses:

$$x^{-1} = (yz)\cdot(xyz)^{-1} \qquad y^{-1} = (xz)\cdot(xyz)^{-1}$$

$$z^{-1} = (xy)\cdot(xyz)^{-1}$$

Thus the inverses of all three numbers are obtained at the cost of only a single modular inverse along with a number of multiplications. More generally, it can be shown that by letting $x_1, \ldots, x_n \in \mathbb{Z}_N$, all n inverses $x_1^{-1}, \ldots, x_n^{-1}$ can be obtained at the cost of one inversion and 3n - 3 multiplications.

It can be proven that a general batched-inversion algorithm proceeds in three phases. First, set $A_1 \leftarrow x_1$, and $A_i \leftarrow x_i \cdot A_{i-1}$ for $i > 1$. By induction, it can be shown that

$$A_i = \prod_{j=1}^{i} x_j .$$

Next, invert $A_n = \prod x_j$, and store the result in $B_n : B_n \leftarrow (A_n)^{-1} = \prod x_j^{-1}$. Now, set $B_i \leftarrow x_{i+1} \cdot B_{i+1}$ for $i < n$. Again, it can be shown that

$$B_i = \prod_{j=1}^{i} x_j^{-1} .$$

Finally, set $C_1 \leftarrow B_1$, and $C_i \leftarrow A_{i-1} \cdot B_i$ for $i > 1$. Furthermore, $C_1 = B_1 = x_1^{-1}$, and, by combining, $C_i = A_{i-1} \cdot B_i = x_i^{-1}$ for $i > 1$. This embodiment has thus inverted each x_i.

Each phase above requires n - 1 multiplications, since one of the n values is available without recourse to multiplication in each phase. Therefore, the entire algorithm computes the inverses of all the inputs in $3n$ - 3 multiplications and a single inversion.

In another embodiment batched division can be combined with delayed division, wherein promises at the leaves of the batch tree are evaluated using batched division. Consequently, only a single modular inversion is required for the entire batching procedure. Note that the batch division algorithm can be easily modified to conserve memory and store only n intermediate values at any given time.

The Chinese Remainder Theorem is typically used in calculating RSA decryptions. Rather than computing $m \leftarrow v^d \pmod{N}$, the modulo p and q is evaluated:

$$m_p \leftarrow v_p^{d_p} \pmod{p} \qquad m_q \leftarrow v_p^{d_q} \pmod{q}.$$

Here $d_p = d \bmod p - 1$ and $d_q = d \bmod q - 1$. Correspondingly the CRT can calculate m from m_p and m_q. This is approximately 4 times faster than evaluating m directly.

This idea extends naturally to batch decryption. In one embodiment each encrypted message v_i modulo p and q is reduced. Then, instead of using a single batch tree modulo N, two separate, parallel batch trees, modulo p and q, are used and then combined to the final answers from both using the CRT. Batching in each tree takes between a quarter and an eighth as long as in the original, unified tree since the number-theoretical primitives employed, as commonly implemented, take quadratic or cubic time in the bit-length of the modulus. Furthermore, the b CRT steps required to calculate each $m_i \bmod N$ afterwards takes negligible time compared to the accrued savings.

Another embodiment referred to herein as Simultaneous Multiple Exponentiation provides a method for calculating $a^u \cdot b^v \bmod m$ without first evaluating $a^u \cdot b^v$. It requires approximately as many multiplications as does a single exponentiation with the larger of u or v as an exponent.

For example, in the percolate-upward step, $V \leftarrow V_L^{E_R} \cdot V_R^{E_L}$ the entire right-hand side can be computed in a single multi-exponentiation. The percolate-downward step involves the calculation of the quantity $v_L^{X_L} \cdot v_R^{X_R}$,which can be accelerated similarly. These small-exponentiations-and-product calculations are a larger part of the extra bookkeeping work required for batching. Using Simultaneous Multiple Exponentiation reduces the time required to perform them by close to 50% by combining the exponentiation process.

Yet another embodiment involves Node Reordering. Normally there are two factors that determine performance for a particular batch of keys. First, smaller encryption exponents are better. The number of multiplications required for evaluating a small exponentiation is proportional to the number of bits in the exponent. Since upward and downward percolation both use *O(b)* small exponentiations, increasing the value of $e = \prod e_i$ can have a drastic effect on the efficiency of batching.

Second, some exponents work well together. In particular, the number of multiplications required for a Simultaneous Multiple Exponentiation is proportional to the number of bits in the larger of the two exponents. If batch trees are built that have balanced exponents for multiple exponentiation (E_L and E_R, then X_L and X_R, at each inner node), the multi-exponentiation phases can be streamlined.

With $b = 4$, optimal reordering is fairly simple. Given public exponents $e_1 < e_2 < e_3 < e_4$, the arrangement $e_1 - e_4 - e_2 - e_3$ minimizes the disparity between the exponents used in Simultaneous Multiple Exponentiation in both upward and downward percolation. Rearranging is harder for $b > 4$.

Figure 4 is an embodiment of a system 400 for improving secure communications. The system includes multiple client computers 432, 434, 436, 438 and 440 which are coupled to a server system 410 through a network, 430. The network 430 can be any network, such as a local area network, a wide area network, or the Internet. Coupled among the server system 410 and the network 430 is a decryption server. While illustrated as a separate entity in **Figure 4**, the decryption server can be located independent of the server system or in the environment or among any number of server sites 412, 414 and 416. The client computers each include one or more processors and one or more storage devices. Each of the client computers also includes a display device, and one or more input devices. All of the storage devices store various data and software programs. In one embodiment, the method for improving secure communications is carried out on the system 400 by software instructions executing on one or more of the client computers 432 - 440. The software instructions may be stored on the server system 410 any one of the server sites 412 - 416 or on any one of the client computers 432 - 440. For example, one embodiment presents a hosted application where an enterprise requires secure communications with the server. The software instructions to enable the communication are stored on the server and accessed through the network by a client computer operator of the enterprise. In other embodiments, the software instructions may be stored and executed on the client computer. A user of the client computer with the help of a user interface can enter data required for the execution of the software instructions. Data required for the execution of the software instructions can also be accessed via the network and can be stored anywhere on the network.

Building the batch RSA algorithm into real-world systems presents a number of architectural challenges. Batching, by its very nature, requires an aggregation of requests. Unfortunately, commonly-deployed protocols and programs are not designed with RSA aggregation in mind. The solution in one embodiment is to create a batching server process that provides its clients with a decryption oracle, abstracting away the details of the batching procedure.

With this approach modifications to the existing servers are minimized. Moreover, it is possible to simplify the architecture of the batch server itself by freeing it from the vagaries of the SSL protocol. An example of the resulting web server design is shown in **Figure 5**. Note that in batching the web server manages multiple certificates, i.e., multiple public keys, all sharing a common modulus N 510.

One embodiment for managing multiple certificates is the two-tier model. For a protocol that calls for public-key decryption, the presence of a batch-decryption server 520 induces a two-tier model. First is the batch server process that aggregates and performs RSA decryptions. Next are client processes that send decryption requests to the batch server. These client processes implement the higher-level application protocol (e.g., SSL) and interact with end-user agents (e.g., browsers).

Hiding the workings of the decryption server from its clients means that adding support for batch RSA decryption to existing servers engenders the same changes as adding support for hardware-accelerated decryption. The only additional challenge is in assigning the different public keys to the end-users such that there are roughly equal numbers of decryption requests with each e_i. As the end-user response times are highly unpredictable, there is a limit to the flexibility that may be employed in the public key distribution.

If there are k keys each with a corresponding certificate, it is possible to create a web with ck web server processes with a particular key assigned to each. This approach provides that individual server processes need not be aware of the existence of multiple keys. The correct value for c depends on factors such as, but not limited to, the load on the site, the rate at which the batch server can perform decryption, and the latency of the communication with the clients.

Another embodiment accommodates workload unpredictability. The batch server performs a set of related tasks including receiving requests for decryption, each of which is encrypted with a particular public exponent e_i. Having received the requests it aggregates these into batches and performs the batch decryption as described herein. Finally, the server responds to the requests for decryption with the corresponding plain-text messages. The first and last of these tasks are relatively simple I/O problems and the decryption stage is discussed herein. What remains is the scheduling step.

One embodiment possesses scheduling criteria including maximum throughput, minimum turnaround time, and minimum turnaround-time variance. The first two criteria are self-evident and the third is described herein. Lower turnaround-time variance means the server's behavior is more consistent and predictable which helps prevent client timeouts. It also tends to prevent starvation of requests, which is a danger under more exotic scheduling policies.

Under these constraints a batch server's scheduling can implement a queue where older requests are handled first. At each step the server seeks the batch that allows it to service the oldest outstanding requests. It is impossible to compute a batch that includes more than one request encrypted with any particular public exponent e_i. This immediately leads to the central realization about batch scheduling that it makes no sense, in a batch, to service a request that is not the oldest for a particular e_i. However, substituting the oldest request for a key into the batch improves the overall turnaround-time variance and makes the batch server better approximate a perfect queue.

Therefore, in choosing a batch, this embodiment needs only consider the oldest pending request for each e_i. To facilitate this, the batch server keeps k

queues Q_i, or one for each key. When a request arrives, it is placed onto the queue that corresponds to the key with which it was encrypted. This process takes $O(1)$ time. In choosing a batch, the server examines only the heads of each of the queues.

Suppose that there are k keys, with public exponents $e_1, \ldots, e_k$, and that the server decrypts requests in batches of b messages each. The correct requests to batch are the b oldest requests from amongst the k queue heads. If the request queues Q_i are kept in a heap with priority determined by the age of the request at the queue head, then batch selection can be accomplished by extracting the maximum, oldest-head, queue from the heap, de-queue the request at its head, and repeat the process to obtain b requests to batch. After the batch has been selected, the b queues from which requests were taken may be replaced in the heap. The entire process takes $O(b \lg k)$ time.

Another embodiment utilizes multi-batch scheduling. While the process described above picks only a single batch, it is possible, in some cases, to choose several batches at once. For example, with $b = 2$, $k = 3$, and requests for the keys 3–3–5–7 in the queues, the one-step lookahead may choose to do a 5–7 batch first, after which only the unbatchable 3–3 remain. A smarter server could choose to do 3–5 and 3–7 instead. The algorithms for doing lookahead are more complicated than the single-batch algorithms. Additionally, since they take into account factors other than request age, they can worsen turnaround-time variance or lead to request starvation.

A more fundamental objection to multi-batch lookahead is that performing a batch decryption takes a significant amount of time. Accordingly, if the batch server is under load, additional requests will arrive by the time the first chosen batch has been completed. These can make a better batch available than was without the new requests.

But servers are not always under maximal load. Server design must take different load conditions into account. One embodiment reduces latency in a medium-load environment by using k public keys on the web server and allowing batching of any subset of b of them, for some $b < k$. To accomplish this the batches must be pre-constructed and the constants associated with $\binom{k}{b}$ batch trees must be keep in memory one for each set of e's.

However, it is no longer necessary to wait for exactly one request with each e before a batch is possible. For k keys batched b at a time, the expected number of requests required to give a batch is

$$E[\#\text{ requests}] = k \cdot \sum_{i=1}^{b} \frac{1}{k-i+1}.$$

This equation assumes each incoming request uses one of the k keys randomly and independently. With $b = 4$, moving from $k = 4$ to $k = 6$ drops the expected length of the request queue at which a batch is available by more than 31%, from 8.33 to 5.70.

The particular relationship of b and k can be tuned for a particular server. The batch-selection algorithm described herein is time-performance logarithmic in k, so the limiting factor on k is the size of the k^{th} prime, since particularly large values of e degrade the performance of batching.

In low-load situations, requests trickle in slowly, and waiting for a batch to be available can introduce unacceptable latency. A batch server should have some way of falling back on unbatched RSA decryption, and, conversely, if a batch is available and batching is a better use of processor time than unbatched RSA, the servers should be able to exploit these advantages. So, by the considerations given above, the batch server should perform only a single unbatched decryption, then look for new batching opportunities.

Scheduling the unbatched decryptions introduces some complications. Previous techniques in the prior art provide algorithms that when requests arrive, a batch is accomplished if possible, otherwise a single unbatched decryption is done. This type of protocol leads to undesirable real-world behavior. The batch server tends to exhaust its queue quickly. Furthermore it responds immediately to each new request and never accumulates enough requests to batch.

One embodiment chooses a different approach that does not exhibit the performance degradation associated with the prior art. The server waits for new requests to arrive, with a timeout. When new requests arrive, it adds them to its queues. If a batch is available, it evaluates it. The server falls back on unbatched RSA decryptions only when the request-wait times out. This approach increases the server's turnaround-time under light load, but scales gracefully in heavy use. The timeout value is tunable.

Since modular exponentiation is asymptotically more expensive than the other operations involved in batching, the gain from batching approaches a factor-of-b improvement only when the key size is improbably large. With 1024-bit RSA keys the overhead is relatively high and a naive implementation is slower than unbatched RSA. The improvements described herein lower the overhead and improve performance with small batches and standard key-sizes.

Batching provides a sizeable improvement over plain RSA with $b = 8$ and $n = 2048$. More important, even with standard 1024-bit keys, batching significantly improves performance. With $b = 4$, RSA decryption is accelerated by a factor of 2.6; with $b = 8$, by a factor of almost 3.5. These improvements can be leveraged to improve SSL handshake performance.

At small key sizes, for example $n = 512$, an increase in batch size beyond $b = 4$ provides only a modest improvement in RSA performance. Because of the increased latency that large batch sizes impose on SSL handshakes, especially when the web server is not under high load, large batch sizes are of limited utility for real-world deployment.

SSL handshake performance improvements using batching can be demonstrated by writing a simple web server that responds to SSL handshake requests and simple HTTP requests. The server uses the batching architecture described herein. The web server is a pre-forked server, relying on "thundering herd" behavior for scheduling. All pre-forked server processes contact an additional batching server process for all RSA decryptions as described herein.

Batching increases handshake throughput by a factor of 2.0 to 2.5, depending on the batch size. At better than 200 handshakes per second, the batching web server is competitive with hardware-accelerated SSL web servers, without the need for the expensive hardware.

Figure 6 is a flow diagram for improving secure socket layer communication through batching of an embodiment. As in a typical initial handshake between server and client in establishing a secure connection, the client uses the server's public key to encrypt a random string R and then sends the encrypted R to the server 620. The message is then cached 625 and the batching process begins by determining is there is sufficient encrypted messages coming into the server to form a batch 630. If the answer to that query is no, it is determined if the scheduling algorithm has timed out 640. Again if the answer is no the message returns to be held with other cached messages until a batch has been formed or the scheduler has timed out. If the scheduler has timed out 640 then the web server receives the encrypted message from the client containing R 642 The server then employs the private key of the public / private RSA key pair to decrypt the message and determine R 646. With R determined the client and the server use R to secure further communication 685 and establish an encrypted session 690.

Should enough encrypted messages be available to create a batch 630 the method examines the possibility of scheduling multiple batches 650. With the scheduling complete the exponents of the private key are balanced 655 and the e^{th} root of the combined messages is extracted 658 allowing a common root to be determined and utilized 660. The embodiment continues by reducing the number of inversions by conducting delayed division 662 and batched division 668. With the divisions completed, separate parallel batch trees are formed to determine the final inversions that are then combined 670. At this point simultaneous multiple exponents are applied to decrypt the messages 672 which are separated 676 and sent to the server in clear text 680. With the server and client both possessing the session key R 685 a encrypted session can be established 690.

Batching increases the efficiency and reduces the cost of decrypting the cipher-text message containing the session's common key. By combining the decryption of several messages in an optimized and time saving manner the server is capable of processing more messages thus increasing bandwidth and improving the over all effectiveness of the network. While the batching techniques described previously are a dramatic improvement in secure socket layer communication, other techniques can also be employed to improve the handshake protocol.

From the above description and drawings, it will be understood by those of ordinary skill in the art that the particular embodiments shown and described are for purposes of illustration only and are not intended to limit the scope of the claimed invention.

CLAIMS

What is claimed is:

1. A method for secure computer communications, comprising:

generating a Rivest-Shamir-Adleman ("RSA") algorithm public / private key pair at a web server, wherein <N, *e*'>, represents the public key with N being the product of two distinct primes, p and q, and wherein the private key is represented by d;

sending a client hello message to the web server from a client requesting a secure network connection;

responding to the client with a server hello message comprising the RSA public key;

encrypting a random string R at the client using the RSA public key, wherein the resulting cipher-text C includes R;

sending the encrypted cipher-text to the web server;

decrypting the cipher-text at the web server using the RSA private key wherein $d=r_1 \bmod (p-1)$ and $d=r_2 \bmod (q-1)$, and wherein $<r_1, r_2>$ are relatively small numbers on the order of 160 bits in length, wherein R'_1 equals the cipher-text raised to the r_1 power moduli one of the distinct prime numbers and R'_2 equals the cipher-text raised to the r_2 power moduli the remaining prime number;

combining R'_1 and R'_2 to produce R using the Chinese Remainder Theorem wherein finding R'_1 and R'_2 is more efficient than using standard RSA keys; and

establishing a common session key between the web server and client using R.

2. The method of claim 1, wherein the secure communications includes Secure Socket Layer ("SSL") messages.

3. The method of claim 1, wherein the secure communications includes Transport Layer Security ("TLS") messages.

4. The method of claim 1, wherein the secure communications includes internet protocol secure ("IPSec") messages.

5. The method of claim 1, wherein generating a RSA public / private key pair includes;

taking the product of the n-bit primes to produce an arbitrary number N;

picking two random k-bit values r_1 and r_2 such that r_1 and r_2 are on the order of 160 bits and are mathematically related to the n-bit primes and e' is related to N; and

sending the public key to a certificate authority and receiving back from the certificate authority a public key certificate for a public key wherein e' is on the order of N in size.

6. The method of claim 5, wherein the k-bit values are related to the n-bit primes by the equations $\gcd(r_1, p-1) = 1$, $\gcd(r_2, q-1) = 1$, and $r_1 = r_2$ mod w, respectively, wherein gcd represents the greatest common divisor and $w = \gcd(p\text{-}1,q\text{-}1)$.

7. The method of claim 6, wherein the relationship between e' and N is expressed by the equation $e' = d^{-1} \bmod \varphi(N)$.

8. The method of claim 1, wherein decrypting includes:

computing R_1' and R_2' as expressed by the relationship $R_1 = C^{r_1} \bmod p$ and $R_2 = C^{r_2} \bmod q$; and

applying the Chinese Remainder Theorem to produce R, wherein $R = R_1' \bmod p$ and $R = R_2' \bmod q$

9. A method for performing an initial handshake during secure communications in a computer network comprising:

coupling a client to a web server;

generating a Rivest-Shamir-Adleman ("RSA") algorithm public / private key pair at the web server, wherein the RSA public key is a product of two distinct prime numbers and the private key is a function of two random numbers, wherein each random number has a number of bits greater than or equal to 160 bits and less than a number of bits of the RSA key;

sending a client hello message to the web server requesting a secure network connection;

responding to the client with a server hello message containing the RSA public key;

encrypting a random string R at the client using the RSA public key, wherein the resulting cipher-text C includes R;

sending the encrypted cipher-text message to the web server;

separating cipher-text moduli of the two distinct prime numbers;

decrypting the moduli of the two distinct prime numbers individually using the two random numbers, wherein the results are combined using the Chinese Remainder Theorem, wherein computational efficiency is improved; and

establishing a common session key between the web server and the client using R.

10. The method of claim 9, wherein the initial handshake of secure communications includes Secure Socket Layer ("SSL") messages.

11. The method of claim 9, wherein the initial handshake of secure communications includes Transport Layer Security ("TLS") messages.

12. The method of claim 9, wherein the initial handshake of secure communications includes internet protocol secure ("IPSec") messages.

13. The method of claim 9, further comprising:

combining individually encrypted messages into a set of encrypted messages wherein each encrypted message possesses a public key comprising an encryption exponent;

determining a root node of a binary tree containing leaf nodes corresponding to each encryption exponent using a plurality of separate parallel batch trees, wherein the root node of each tree is found and combined to determine the final answer;

minimizing a disparity between sizes of the encryption exponents of the within the set;

using simultaneous multiple exponentiation such that the encryption exponents are combined to reduce the number of exponentiations;

calculating a product of the encrypted messages;

extracting at least one root from the product of the encrypted messages; and

decrypting the encrypted messages by expressing the at least one root as at least one promise and evaluating the at least one promise at the leaf nodes, and multiplying an inversion of a total product of the leaf nodes with a partial product of the leaf nodes forming an inversion of the leaf node, producing a reduced number of modular inversions wherein efficiency of the decryption is increased.

14. The method of claim 9, further comprising keeping the size of N constant while reducing the size of the two distinct prime numbers such that the size of the two distinct prime numbers is on the order of one third of the size of N.

15. A method for secure communications, comprising:

generating a Rivest-Shamir-Adleman ("RSA") algorithm public / private key pair at a web server, wherein the RSA public key is a product of two distinct prime numbers and the private key is a function of two random numbers;

receiving a client hello message from a client requesting a secure socket layer ("SSL") coupling;

responding to the client with a server hello message containing the RSA public key;

encrypting a random string R at the client using the RSA public key, wherein the resulting cipher-text includes R;

receiving the encrypted cipher-text message at the web server;

separating cipher-text moduli of the two distinct prime numbers;

decrypting the moduli of the distinct prime numbers individually using the two random numbers, wherein the results are combined using the Chinese Remainder Theorem; and

establishing a common session key between the web server and client using R.

16. A method for secure computer communications, comprising:

coupling a web server to a client wherein the client requests the formation of a secure network connection;

generating a Rivest-Shamir-Adleman("RSA") algorithm public / private key pair, the public key comprising a root N, wherein N of the RSA public key is the product of two distinct n-bit prime numbers, p and q, wherein an encryption exponent e' of the RSA public key is of the same order in size as the public key root, N

encrypting a plain-text message R using the RSA public key such that the resulting text is cipher-text C;

decrypting the cipher-text C using the RSA private key wherein the RSA private key is a function of two roots r_1 and r_2, wherein the two roots each are on the order of 160 bits in length; and

using the plain-text message R to determine a session encryption key and a session integrity key.

17. A method for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications, comprising:

generating a RSA public/private key pair at a web server, wherein <N, *e*> represents the public key that is mathematically related to two distinct prime numbers;

keeping a size of N constant while reducing a size of the two distinct prime numbers by calculating N from a product of a first distinct prime number raised to the first power and a second distinct prime number wherein the first power is greater than one;

using the public key by a client to encrypt a plain-text message R to form a cipher-text message C;

decrypting the cipher-text C at the web server by using the RSA private key d to determine the plain-text message R by finding R'_1 and R'_2, wherein the private key is a function of two random numbers $<r_1, r_2>$, and wherein an additional R''_1 is constructed by using one of the two distinct prime numbers raised to a power greater than one, wherein efficiency of the decryption is increased in response to the reduced size of the two distinct prime numbers; and

computing the plain-text message using the Chinese Remainder Theorem.

18. The method of claim 17, further comprising;

combining individually encrypted network security protection handshake messages into a set of encrypted messages wherein each encrypted message is derived using a public key containing an encryption exponent;

determining a root node of a binary tree comprising leaf nodes corresponding to each encrypted messages encryption exponent by using a plurality of separate, parallel batch trees finding the root node of each tree and combining the final answers;

minimizing the disparity between the sizes of the encryption exponents of the public keys within the set;

using simultaneous multiple exponentiation such that the encryption exponents are combined to reduce the number of exponentiations;

calculating a product of the encrypted messages;

extracting at least one root from the product of the encrypted messages; and

decrypting the encrypted messages by expressing the at least one root as at least one promise and evaluating the at least one promise at the leaf nodes, and multiplying an inversion of a total product of the leaf nodes with a partial product of the leaf nodes forming an inversion of the leaf node wherein the decryption is increased by reducing the number of modular inversions.

19. The method of claim 17, wherein the *k*-bit values r_1, r_2 are related to the n-bit primes by the greatest common divisor of $(r_1, p-1) = 1$, $(r_2, q-1) = 1$, $r_1 = r_2 \bmod w$ respectively such that $d = r_1$, $\bmod\, p-1$, $d = r_2 \bmod q-1$, and w is equal to the greatest common divisor of $(p\text{-}1, q\text{-}1)$.

20. The method of claim 17, wherein decrypting includes:

computing R_1', R"$_1$, and R_2' as expressed by the relationships

$$R_1' = C^{r_1} \bmod p \;,\; R_2' = C^{r_2} \bmod q \text{, and } R_1'' = R_1' - \frac{(R_1')^e - C}{e(R_1')^{e-1}} \left(\bmod p^2\right).$$

21. A method for generating a Rivest-Shamir-Adleman ("RSA") public / private key pair in secure network couplings, comprising:

generating two n-bit distinct prime numbers;

computing a public key root from a mathematical relationship between two distinct prime numbers;

reducing the size of the two distinct prime numbers while keeping the size of the public key root constant using exponentiation of the two distinct prime numbers;

forming a public RSA key pair by associating the public key root and a standard RSA encryption exponent; and

computing a private RSA key pair by mathematically combining the standard RSA encryption exponent and the n-bit distinct prime numbers.

22. The method of claim 21, wherein computing the public key root includes the product of the square of one n-bit prime number and the second n-bit prime number.

23. The method of claim 21, wherein the public RSA key pair is indistinguishable from a standard RSA pair.

24. The method of claim 21, further comprising:

encrypting a pre-master-secret using the public RSA key pair; and

decrypting the pre-master-secret using the private RSA key pair wherein Hensle lifting compensates for reducing the size of the distinct prime numbers.

25. A method for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications, comprising:

generating a RSA public/private key pair at a web server, wherein <N, *e*> represents a public key that is mathematically related to two distinct prime numbers and d represents a private key that is mathematically related to two random numbers;

keeping a size of N constant while reducing a size of the two distinct prime number by calculating N from a product of a first distinct prime number raised to a power greater than one and the second distinct prime number;

using the public key at a client to encrypt a plain-text message R to form a cipher-text message C;

decrypting the cipher-text C at the web server using the RSA private key d to determine the plain-text message R by finding R'_1 and R'_2, wherein an additional R''_1 is constructed by raising the first of the two distinct prime numbers to a power greater than one, wherein the efficiency of the decryption is increased due to a reduced size of the two distinct prime numbers using the private RSA key pair, wherein Hensle lifting compensates for altering a multiplicity of the distinct prime numbers; and

computing the plain-text message using the Chinese Remainder Theorem.

26. A method for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications, comprising:

generating a RSA public/private key pair at the web server wherein <N,

e> represents the public key that is mathematically related to two distinct prime numbers;

keeping a size of N constant while reducing a size of the two distinct prime numbers such that each of the two distinct prime numbers is on the order of one third of the size of N;

using the public key at a client to encrypt a plain-text message R to form a cipher-text message C;

decrypting the cipher-text C at the web server by using the RSA private key d, to determine the plain-text message R by finding R'_1 and R'_2, wherein an additional R''_1 is constructed by using the one of the two distinct prime numbers raised to a power greater than one, wherein the efficiency of the decryption is increased in response to the reduced size of the two distinct prime numbers using the private RSA key pair wherein Hensle lifting compensates for altering the multiplicity of the distinct prime numbers; and

computing the plain-text message using the Chinese Remainder Theorem.

27. A system for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications, comprising:

a web server generating a RSA public/private key pair wherein <N, *e*> represents a public key that is mathematically related to two distinct prime numbers;

the web server keeping a size of N constant while reducing a size of the two distinct prime numbers by calculating N from the product of a first distinct prime number raised to a power greater than one and a second distinct prime number;

a client using the public key to encrypt a plain-text message R to form a cipher-text message C;

the web server decrypting the cipher-text C by using the RSA private key d to determine the plain-text message R by finding R'_1 and R'_2, wherein an additional R''_1 is constructed by using one of the two distinct prime numbers raised to a power greater than one wherein the efficiency of the decryption is increased in response to the reduced size of the two distinct prime numbers; and

the web server computing the plain-text message using the Chinese Remainder Theorem.

28. A system for using Rivest-Shamir-Adleman ("RSA") decryption of secure network communications in a computer network, comprising:

at least one web server;

at least one client processor coupled to the at least one web server, wherein the at least one web server generates a RSA public/private key pair, <N, *e*>, representing the public key that is mathematically related to two distinct prime numbers, wherein d represents the private key;

the at least one web server keeping a size of N constant while reducing a size of the two distinct prime numbers by calculating N from the product of a first distinct prime number raised to a power greater than one and a second distinct prime number;

the at least one client processor using the public key to encrypt a plain-text message R to form a cipher-text message C;

the at least one web server decrypting the cipher-text message C by using the RSA private key $<r_1, r_2>$ to determine the plain-text message R by finding R'_1 and R'_2, wherein an additional R''_1 is constructed by using one of the two distinct prime numbers raised to a power greater than one wherein the efficiency of the decryption is increased in response to the reduced size of the two distinct prime numbers; and

the at least one web server computing the plain-text message using the Chinese Remainder Theorem.

29. A computer-readable medium, comprising executable instructions for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications which, when executed in a processing system, causes the system to:

couple a web server to a client;

send a client hello message to the web server requesting a secure network connection;

generate a Rivest-Shamir-Adleman ("RSA") algorithm public / private key pair at the web server wherein the RSA public key is the product of two distinct prime numbers wherein the RSA private key is a function of two random numbers wherein each random number has a number of bits greater than or equal to 160 bits and less than a number of bits of the RSA key;

respond to the client with a server hello message containing the RSA public key;

encrypt a random string R at the client using the RSA public key, wherein the resulting cipher-text C includes R;

send the encrypted cipher-text message C to the web server;

separate cipher-text C moduli of the two distinct prime numbers;

decrypt the moduli of the two distinct prime numbers individually using the two random numbers, wherein results are combined using the Chinese Remainder Theorem, wherein computational efficiency is improved and

establish a common session key between the web server and the client using R.

30. An electromagnetic medium, comprising executable instructions for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications which, when executed in a processing system, causes the system to:

couple a web server to a client;

send a client hello message to the web server requesting a secure network connection;

generate a Rivest-Shamir-Adleman ("RSA") algorithm public / private key pair at the web server wherein the RSA public key is the product of two distinct prime numbers, wherein the RSA private key is a function of two random numbers wherein each random number has a number of bits greater than or equal to 160 bits and less than a number of bits of the RSA key;

respond to the client with a server hello message containing the RSA public key;

encrypt a random string R at the client using the RSA public key, wherein the resulting cipher-text C includes R;

send the encrypted cipher-text message C to the web server;

separate cipher-text moduli of the two distinct prime numbers;

decrypt the moduli of the two distinct prime numbers individually using the two random numbers, wherein results are combined using the Chinese Remainder Theorem, wherein computational efficiency is improved; and

establish a common session key between the web server and the client using R.

31. A computer-readable medium, comprising executable instructions for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications which, when executed in a processing system, causes the system to:

generate a RSA public/private key pair at the web server wherein <N, *e*> represents the public key that is mathematically related to two distinct prime numbers;

keep a size of N constant while reducing a size of the two distinct prime numbers such that each of the two distinct prime numbers is on the order of one third of the size of N;

use the public key at client to encrypt a plain-text message R to form a cipher-text message C;

decrypt the cipher-text C at the web server by using the RSA private key d to determine the plain-text message R by finding R'_1 and R'_2, wherein an additional R''_1 is constructed by using one of the two distinct prime numbers raised to a power greater than one, wherein the efficiency of the decryption is increased in response to the reduced size of the two distinct prime numbers using the private RSA key pair wherein Hensle lifting compensates for altering the multiplicity of the distinct prime numbers; and

compute the plain-text message using the Chinese Remainder Theorem.

32. An electromagnetic medium, comprising executable instructions for Rivest-Shamir-Adleman ("RSA") decryption of secure network communications which, when executed in a processing system, causes the system to:

generate a RSA public/private key pair at the web server wherein <N, *e*> represents the public key that is mathematically related to two distinct prime numbers;

keep a size of N constant while reducing a size of the two distinct prime numbers such that each of the two distinct prime numbers is on the order of one third of the size of N;

use the public key at a client to encrypt a plain-text message R to form a cipher-text message C;

decrypt the cipher-text C at the web server by using the RSA private key d to determine the plain-text message R by finding R'_1 and R'_2, wherein an additional R''_1 is constructed by using one of the two distinct prime numbers raised to a power greater than one, wherein the efficiency of the decryption is increased in response to the reduced size of the two distinct prime numbers using the private RSA key pair wherein Hensle lifting compensates for altering the multiplicity of the distinct prime numbers; and

compute the plain-text message using the Chinese Remainder Theorem.

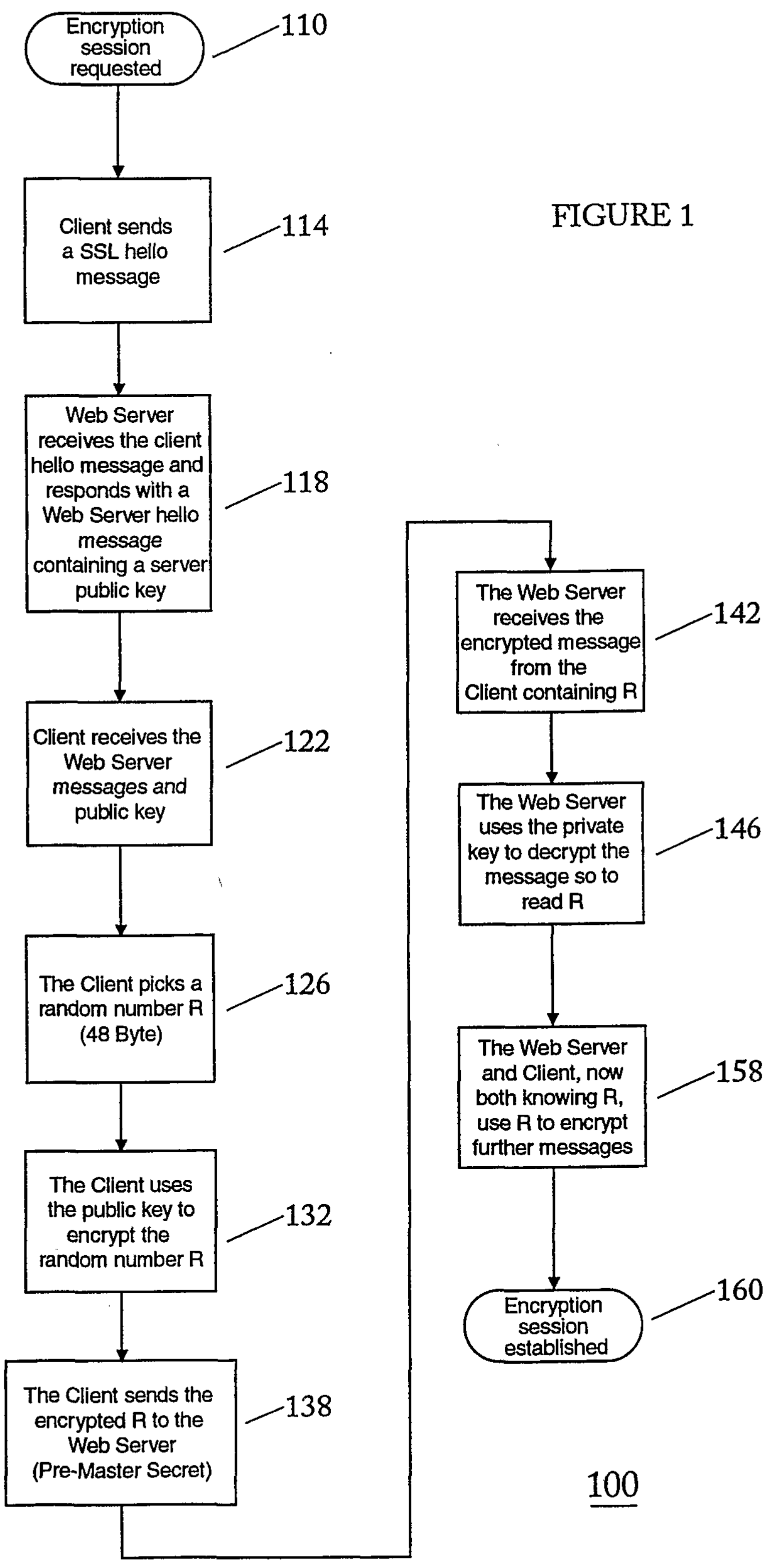
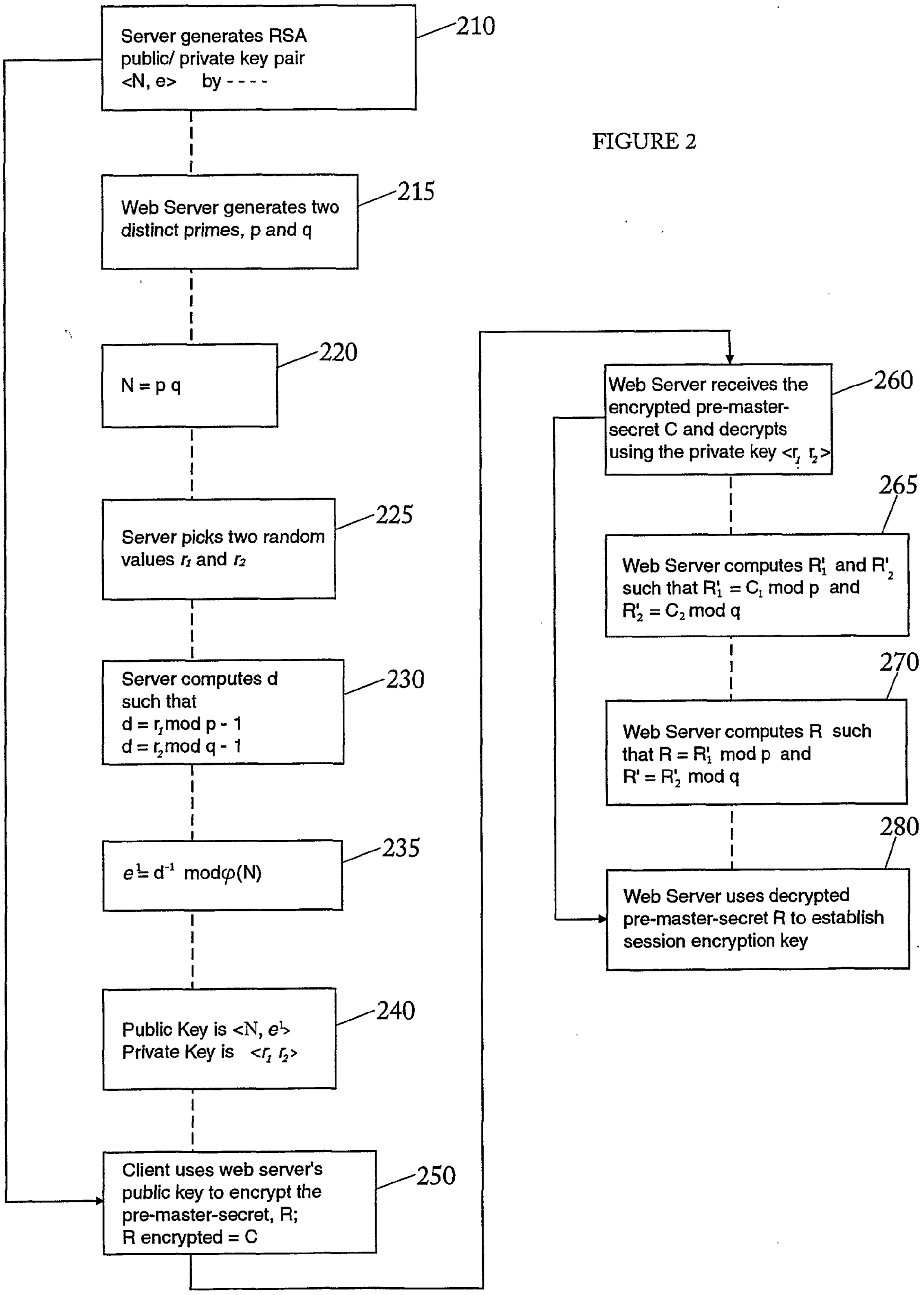

Encryption session requested
110
Client sends a SSL hello message
114
Web Server receives the client hello message and responds with a Web Server hello message containing a server public key
118
Client receives the Web Server messages and public key
122
The Client picks a random number R (48 Byte)
126
The Client uses the public key to encrypt the random number R
132
The Client sends the encrypted R to the Web Server (Pre-Master Secret)
138
The Web Server receives the encrypted message from the Client containing R
142
The Web Server uses the private key to decrypt the message so to read R
146
The Web Server and Client, now both knowing R, use R to encrypt further messages
158
Encryption session established
160
100

FIGURE 1



Server generates RSA public/ private key pair <N, e> by - - - -
210
Web Server generates two distinct primes, p and q
215
N = p q
220
Server picks two random values r_1 and r_2
225
Server computes d such that $d = r_1 \bmod p - 1$ $d = r_2 \bmod q - 1$
230
$e' = d^{-1} \bmod \varphi(N)$
235
Public Key is <N, e'> Private Key is $<r_1\ r_2>$
240
Client uses web server's public key to encrypt the pre-master-secret, R; R encrypted = C
250
Web Server receives the encrypted pre-master-secret C and decrypts using the private key $<r_1\ r_2>$
260
Web Server computes R'_1 and R'_2 such that $R'_1 = C_1 \bmod p$ and $R'_2 = C_2 \bmod q$
265
Web Server computes R such that $R = R'_1 \bmod p$ and $R' = R'_2 \bmod q$
270
Web Server uses decrypted pre-master-secret R to establish session encryption key
280

FIGURE 2

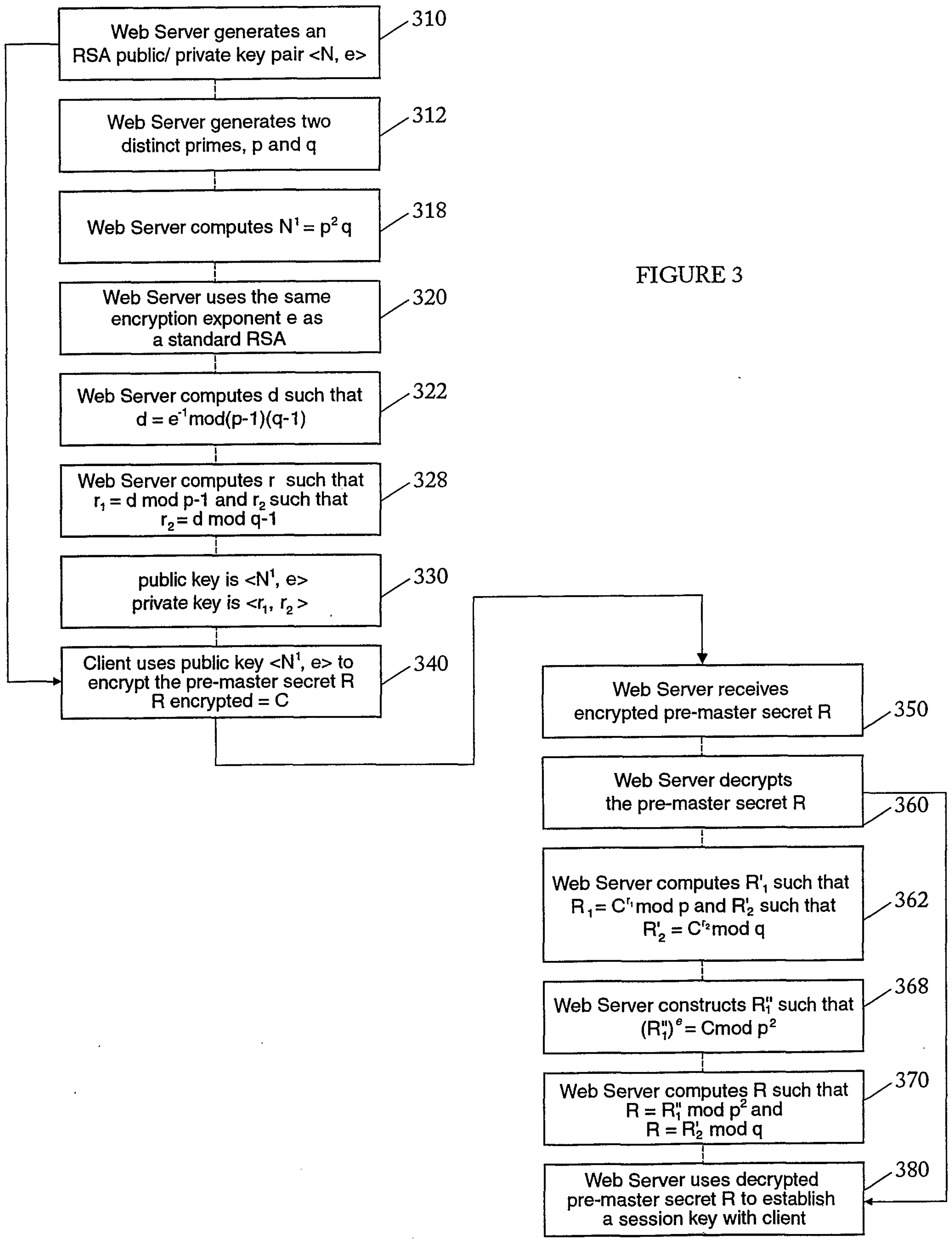

Web Server generates an RSA public/ private key pair <N, e>
310
Web Server generates two distinct primes, p and q
312
Web Server computes $N^1 = p^2 q$
318
Web Server uses the same encryption exponent e as a standard RSA
320
Web Server computes d such that $d = e^{-1} \bmod (p-1)(q-1)$
322
Web Server computes r such that $r_1 = d \bmod p-1$ and r_2 such that $r_2 = d \bmod q-1$
328
public key is $<N^1, e>$ private key is $<r_1, r_2>$
330
Client uses public key $<N^1, e>$ to encrypt the pre-master secret R R encrypted = C
340
Web Server receives encrypted pre-master secret R
350
Web Server decrypts the pre-master secret R
360
Web Server computes R'_1 such that $R_1 = C^{r_1} \bmod p$ and R'_2 such that $R'_2 = C^{r_2} \bmod q$
362
Web Server constructs R''_1 such that $(R''_1)^e = C \bmod p^2$
368
Web Server computes R such that $R = R''_1 \bmod p^2$ and $R = R'_2 \bmod q$
370
Web Server uses decrypted pre-master secret R to establish a session key with client
380

FIGURE 3

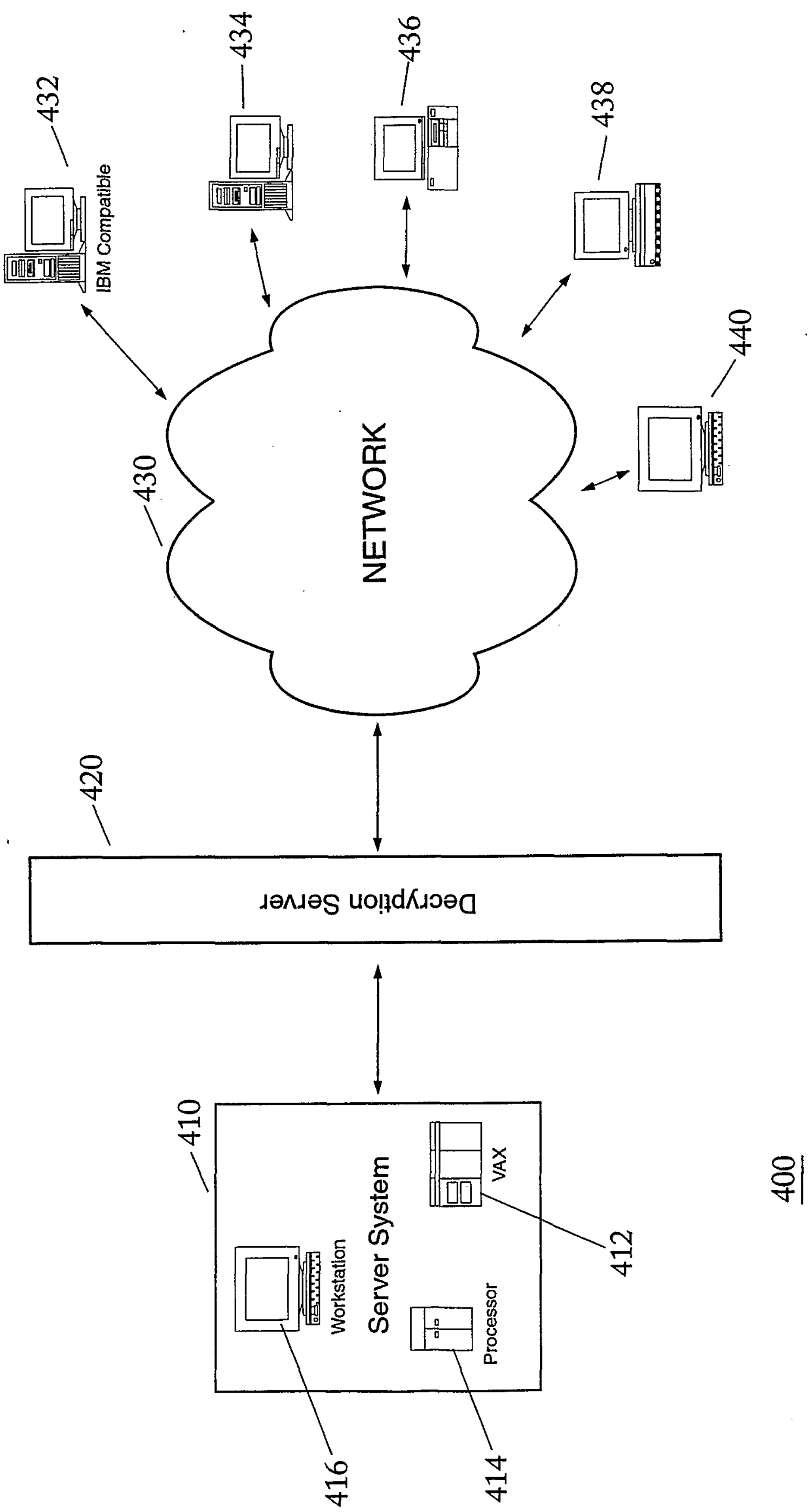

400
410
Server System
Workstation
416
Processor
414
VAX
412
420
Decryption Server
430
NETWORK
432
IBM Compatible
434
436
438
440

FIGURE 4

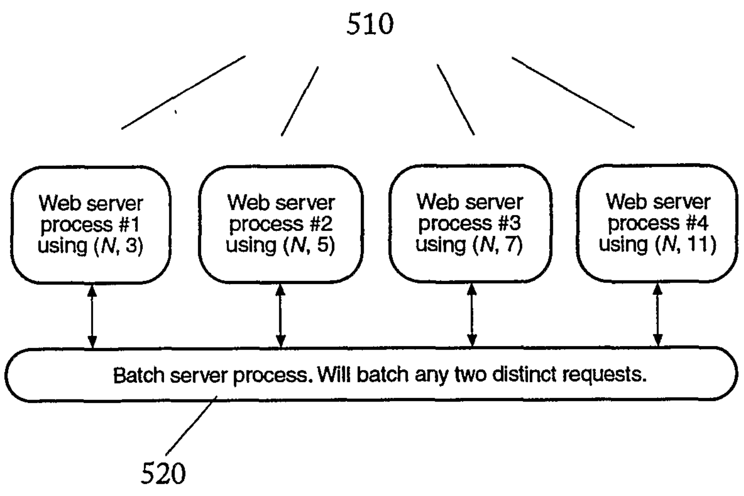
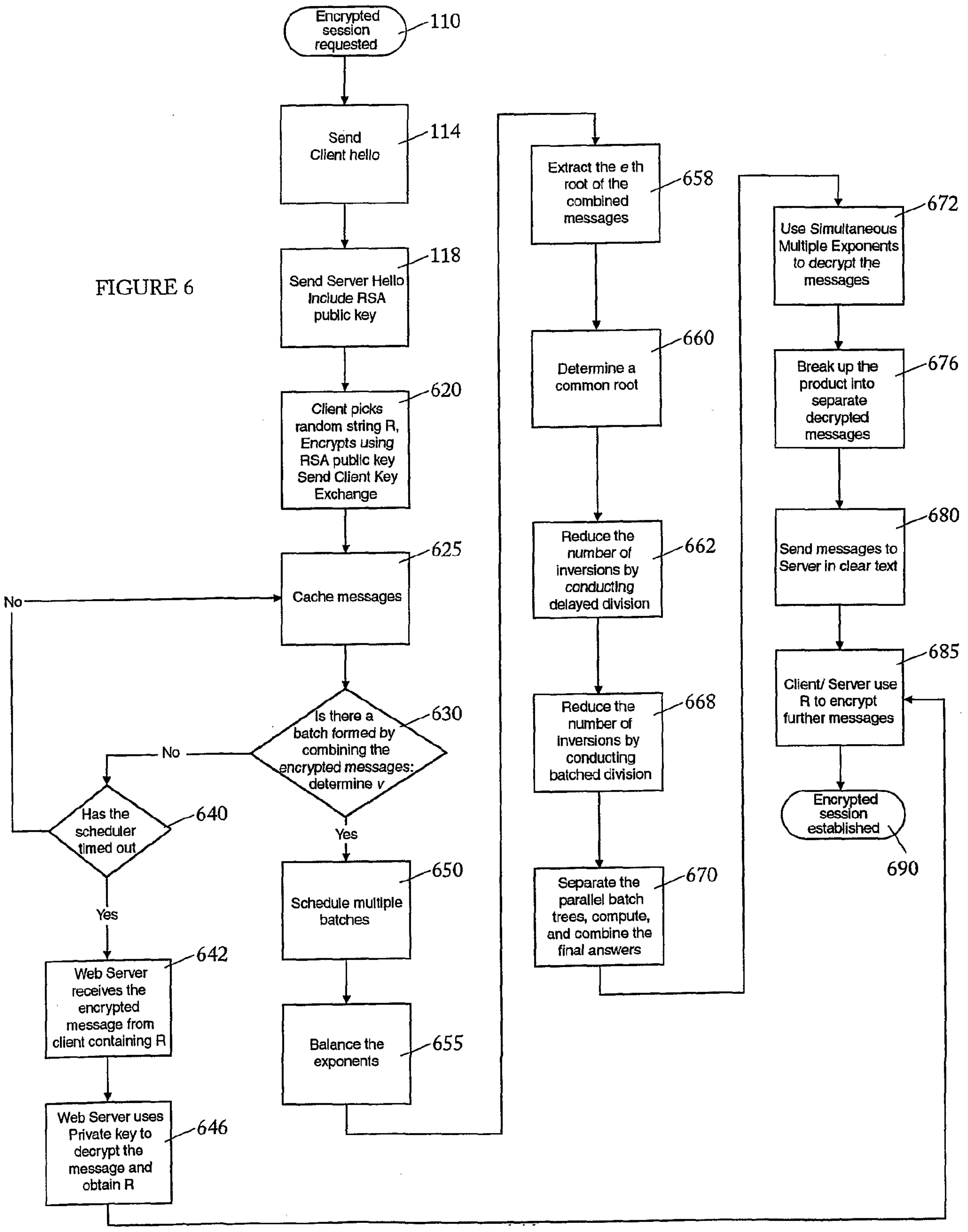

510
Web server process #1 using (N, 3)
Web server process #2 using (N, 5)
Web server process #3 using (N, 7)
Web server process #4 using (N, 11)
Batch server process. Will batch any two distinct requests.
520

FIGURE 5

FIGURE 6



Encrypted session requested
110
Send Client hello
114
Send Server Hello Include RSA public key
118
Client picks random string R, Encrypts using RSA public key Send Client Key Exchange
620
Cache messages
625
No
Is there a batch formed by combining the encrypted messages: determine v
630
No
Yes
Has the scheduler timed out
640
Yes
Web Server receives the encrypted message from client containing R
642
Web Server uses Private key to decrypt the message and obtain R
646
Schedule multiple batches
650
Balance the exponents
655
Extract the e th root of the combined messages
658
Determine a common root
660
Reduce the number of inversions by conducting delayed division
662
Reduce the number of inversions by conducting batched division
668
Separate the parallel batch trees, compute, and combine the final answers
670
Use Simultaneous Multiple Exponents to decrypt the messages
672
Break up the product into separate decrypted messages
676
Send messages to Server in clear text
680
Client/ Server use R to encrypt further messages
685
Encrypted session established
690