J Supercomput (2013) 64:762–776
DOI 10.1007/s11227-010-0546-4

User requirements-aware security ranking in SSL protocol

Fang Qi · Zhe Tang · Guojun Wang · Jie Wu

Published online: 13 January 2011

Abstract The primary goal of the secure socket layer protocol (SSL) is to provide confidentiality and data integrity between two communicating entities. Since the most computationally expensive step in the SSL handshake protocol is the server's RSA decryption, it is introduced that the proposed secret exchange algorithm can be used to speed up the SSL session initialization. This paper first points out that the previous batch method is impractical since it requires multiple certificates. It then proposes a unique certificate scheme to overcome the problem. The optimization strategy, which is based on the constrained model considering the user requirements-aware security ranking, focuses on the optimal result in different public key sizes. It is also introduced that the parameter is optimized when integrating user requirements for Internet QoS, such as the stability of the system and the tolerable response time. Finally, the proposed algorithm is evaluated to be practical and efficient through both analysis and simulation studies.

Keywords Quality of Service (QoS) · SSL handshake · Optimization strategy · Security ranking

1 Introduction

The secure socket layer protocol (SSL) protects communications by encrypting messages with a secret key negotiated in the SSL handshake protocol [1]. How to offer

F. Qi · Z. Tang (✉) · G. Wang
School of Information Science and Engineering, Central South University, 410083 Changsha, PR China
e-mail: zhetang@gmail.com

J. Wu
Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA

some Quality of Service (QoS) that may be satisfied with web users has become a new issue.

Web-based applications rely on the HTTPS protocol to guarantee security and privacy in transactions ranging from e-banking, e-commerce, and e-procurement to those that deal with sensitive data, such as career and identity information [2]. The SSL protocol allows the server and the client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before transmitting and receiving the first byte of data [3]. However, such a protocol needs intensive computational resources due to the cost of public-key operations [4].

In the web context, one of the main factors is the direct consequence of expensive public-key operations performed by servers as part of each SSL handshake. Since most SSL-enabled servers use RSA, the burden of performing many costly decryption operations can be very detrimental to the server performance.

For example, a typical Pentium server (running Linux and Apache) can handle about 322 HTTP connections per second at full capacity, but only 24 SSL connections per second; and a Sun 450 (running Solaris and Apache) fell from 500 to 3 connections per second [5]. There are several ways to improve the performance of the SSL handshake protocol:

1. Hardware: Obviously, a specific circuit can improve the performance. This solution may not be a good solution for small or medium-sized servers [6].
2. Session Caching: The cache allows subsequent connections to resume an earlier TLS session, and thus, to reuse the result of an earlier computation. Research has suggested that, indeed, session caching helps to improve the web server performance [1]. However, the cache technology has no help to speed up the session setup.
3. Software: Many algorithmic approaches for speeding up SSL's performance on a web server are presented in the literature [7–13]. They are designed for heavily-loaded web servers handling many concurrent SSL sessions. However, these schemes ignore the satisfaction of the user requirements for QoS, such as the stability of the system and the tolerable response time.

Being aware of the computational imbalance between clients and server in the SSL handshake protocol, we propose a secret exchange algorithm to overcome the problem. The starting point of the proposed scheme is a technique due to the batch RSA decryption [14, 15]. This paper adapts the certificate mechanism so as to provide the SSL setup with a unique certificate issued by the Certificate Authority (CA). This paper also proposes the constrained model integrating the user-perceived quality into secure web server design [16, 17]. This paper also optimizes the batch size by the constrained model, which meets the user requirements-aware security ranking focusing on the optimal result in different public key sizes. In addition, the proposed scheme uses the approximate analytical solution of the mean response time to optimize the batch size of the server. It is designed for heavily-loaded web servers handling many concurrent SSL sessions. Notice that a preliminary version of this paper appeared in the paper [18].

The rest of the paper is organized as follows: Section 2 describes the secret exchange algorithm in the SSL handshake protocol. The proposed constrained model of the QoS-aware optimization strategy is presented in Sect. 3. The QoS-aware optimization algorithm is presented in Sect. 4. Section 5 validates the proposed solutions through both analysis and simulation studies, and Sect. 6 concludes the paper.

2 Secret exchange algorithm in the SSL handshake protocol

The proposed scheme in this paper focuses on the parameter optimization-based technology, which is a software-only approach for speeding up SSL's performance on a web server. The starting point of the proposed scheme is a technique due to using the batch RSA decryption. Being aware of the computational imbalance between clients and server in the SSL handshake protocol, we proposed algorithms to overcome the problem.

The following Algorithms 1 and 2 are secret exchange algorithms of the SSL handshake at the server side and the client side, respectively. When using small public exponent e_1 and e_2, it is possible to decrypt two cipher texts for approximately the price of one [14]. This technology facilitates more favorable load distribution by requiring the clients to perform more work (as part of encryption) and the server to perform commensurately less work, thus leading to better SSL throughput at the server.

In the standard SSL protocol, each client encrypts a 48-byte pre-master secret as the encryption exponent, and the server decrypts the cipher text independently so as to get the *pre-master secret*. Algorithm 1 obtains the *pre-master secret* from multiple clients and hence, improves the performance significantly.

The parameter b is the size of multi-clients for aggregate decryption. The public key is made of two integers $\langle N, e\rangle$. The value e is called the encryption exponent. Given b, distinct and pairwise relatively prime public keys $e_1, \dots, e_b$, all share a common modulus $N = pq$. The N is the product of two large primes p and q. The security parameter n is the bit length of the public modulus N, and k is the bit length of the bigger of e_i at Steps 1–5 of Algorithm 1. Furthermore, we have b encrypted messages $v_1, \dots, v_b$, and further results $v'_1, \dots, v'_b$, one encrypted with each key at Steps 3–8 of Algorithm 2. An e'_i is defined as each client's sibling's exponent. After we perform Step 10 of Algorithm 1, we call Algorithm 2 and receive the message including cipher text $v_1, \dots, v_b$ and $v'_1, \dots, v'_b$ from each client. At Step 19 of Algorithm 1, quantities subscripted by L or R refer to the corresponding value of the left or right child of the node, respectively.

At Steps 16–21 of Algorithm 1, we combine the individual encrypted messages v_i to form, at the root of the batch tree, the value $v = \prod_{i=1}^{b} v_i^{e/e_i}$, where $e = \prod_{i=1}^{b} e_i$. At Steps 16–20, the first $b/2$ values of "Currentnode.ciphertext" are computed by $v'_1, \dots, v'_b$ because the left child and the right child of "Currentnode" are leaves. The number of external nodes is equal to $b - 1$.

Using the binary tree construction, we are working from the leaves to the root. At every internal node, each encrypted message v_i is placed (as v) in the leaf node labeled with its corresponding e_i. The root node v's value is percolated up the tree using the following recursive step, applied at each internal node at Steps 16–21 of Algorithm 1. E_L, E_R are the left child and the right child of each product of the internal node at Step 17 of Algorithm 1.

Algorithm 1 Secret exchange algorithm at the server side

```
1: Given $b$ distinct and pairwise relatively prime public keys $e_1, \dots, e_b$
2: //all sharing a common modulus $N = pq$, relatively prime to $\phi(N) = (p-1)(q-1)$.
3: //$n$ is the bit length of the public modulus $N$ and $k$ is the bit length of the bigger of $e_i$.
4: Construct a full binary tree $T_d$ which is called the decryption tree with leaves labeled
5: $e_1, \dots, e_b$;
6: **Input**: $e_1, \dots, e_b$
7: **Output**: a full binary tree $T_d$, $m_i$ $1 \le i \le b$
8: Construct a message including $e_i$ and the information $e'_i$ about sibling's exponent for
9: each client, where $1 \le i \le b$;
10: **call** Algorithm 2; upon receiving the message including cipher text $v_i$ and $v'_i$ from each
11: client, where $1 \le i \le b$;
12: //Compute two middle values, *exponent* and *ciphertext*, at each
13: //internal node of $T\_d$ repeating this computation recursively. The number of external
14: //nodes is equal to $b-1$. The computation phase is to generate the product
15: //$v = \prod_{i=1}^{b} v_i^{e/e_i} \quad mod \quad N$, where $e = \prod_{i=1}^{b} e_i$.
16: **for**($j = 1$ to $b-1$)**do**{
17:    $E_L$ ←leftchild.exponent; $E_R$ ←rightchild.exponent; Currentnode.exponent← $E_L \times E_R$;
18:    $L$ ←leftchild.ciphertext; $R$ ←rightchild.ciphertext;
19:   If ($j \le (b/2)$) Currentnode.ciphertext← $v'_{2j-1} \times v'_{2j}$
20:    else Currentnode.ciphertext← $L^{E_R} \times R^{E_L}$;}
21:   $v$ ←rootnode.ciphertext; $e$ ←rootnode.exponent;
22:   //The value of $v$ and $e$ is simply the result associated with the root
23: Compute $m \leftarrow v^{1/e} \quad mod \quad N = \prod_{i=1}^{b} v_i^{1/e_i} \quad mod \quad N$; $e \leftarrow \prod_{i=1}^{b} e_i$
24: //$m$ is the *plaintext* of root node of $T\_d$. $e$ is the *exponent* of root node of $T\_d$.
25: //The next step is to break up the product $m$ to obtain the plaintext $m_i = v_i^{1/e_i}$ with
26: //repeating this computation recursively from the root node.
27: **for**($i = 1$ to $b-1$) **do**{
28:    Compute $X$ while (($X = 0$ mod $E_L$) and ($X = 1$ mod $E_R$)==true);
29:      $X_L \leftarrow X/E_L$; $X_R \leftarrow (X-1)/E_R$;
30:      $m_R \leftarrow m^X/(v_L^{X_L} \cdot v_R^{X_R})$; $m_L \leftarrow m/m_R$;}
31:     // The values of $v_L$ and $v_R$ are simply the results associated with the *ciphertext* of
32:     // a node, which stored at Steps 16–20. Here, $v_L$ and $v_R$ are the ciphertexts of the left
    child and the right child of current node.
33: **return** $m_i$; $1 \le i \le b$
```

Algorithm 2 Secret exchange algorithm at the client side

```
1: **Input**: $e_i, e'_i$
2: **Output**: $v'_i, v_i$
3: Create plaintext $m_i (0 < m_i < N)$;
4: //upon receiving the message including $e_i$;
5: Compute $v_i = m_i^{e_i} \quad \mathrm{mod} \quad N \ 1 \le i \le b$.
6: //upon receiving the message including $e'_i$ which is the sibling's exponent value of $e_i$;
7: Compute $v'_i = v_i^{e'_i} \quad \mathrm{mod} \quad N$;
8: Construct message including $v'_i, v_i$ for server.
9: **return**
```

The root node contains $v = \prod_{i=1}^{b} v_i^{e/e_i}$ at Step 21 of Algorithm 1. The eth root of this v is extracted. We store $v^{1/e} = \prod_{i=1}^{b} v_i^{1/e_i}$ as m in the root node at Step 23 of Algorithm 1. The Steps 27–34 are to break up the product m to obtain the plaintext $m_i = v_i^{1/e_i}$ with repeating this computation recursively from the root node.

3 Unique certificate scheme in the SSL handshake protocol

Our unique certificate method is to reuse the message *ServerHello* in the protocol. For simplicity, we only show the related processes and the modified information in the standard SSL handshake protocol (see Fig. 1).

The following procedure is the unique certificate scheme for the SSL handshake protocol: (1) The clients send a *client hello* massage, which includes cipherspects to the server and create random nonce r_c, respectively. Here, the cipherspects specify the bulk data encryption algorithm, the MAC algorithm, and the cryptographic attributes [3]. (2) The server responds with a *server hello* message that includes server's public-key certificate and a random nonce r_s. In this improvement, e_i and e_i' are actually a part of ServerHello.random. The server only needs to send unique certificates to all the clients. (3) The clients choose a secret random 48-byte *pre-master secret* m_1 and m_2 by inputting values m_1, m_2, r_c, r_s into hash function $f()$. It then encrypts m_i with e_i, which is different from the server's public key, and attaches the cipher text to a *client key exchange* message that is sent to the server. (4) The server decrypts the *pre-master secret* m_1 and m_2 simultaneously using Algorithm 1, and uses it to compute the *shared master secret* s_1 and s_2, respectively. The client will verify the certificate as usual, but encrypt the pre-master secret with received e_i, instead of the public exponent in the certificate. Therefore, no extra charge is required, and it is easy

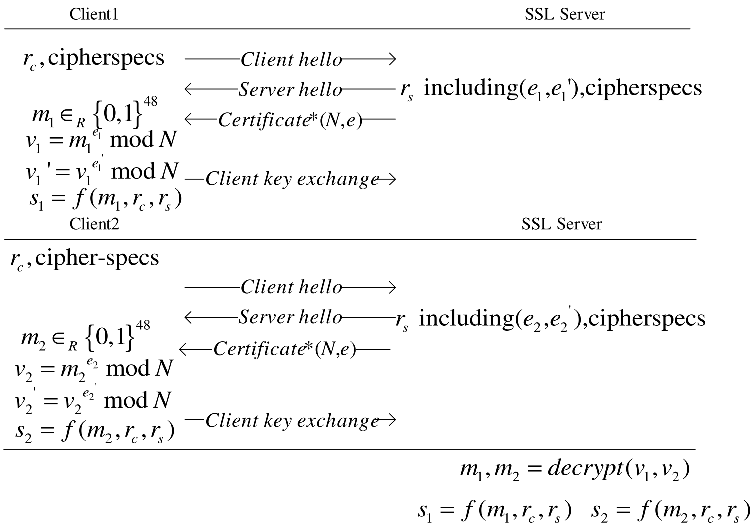


Fig. 1 Unique Certificate Scheme for a partial handshake

to manage the certificate. On the other hand, since the certificate is used to prove the owner who knows the factors of the RSA modulus N only, this adaptation does not undermine the security strength of the SSL protocol.

4 Constrained model of QoS-aware optimization strategy

The optimization strategy, which is based on the constrained model considering the user requirements-aware security ranking, focuses on the optimal result in different public key sizes.

Lemma 1 *Algorithm 1 can generate the b decryption results in $O(\log_2 b \times (\sum_{i=1}^{b} \log_2 e_i) + \log_2 N)$ modular multiplications and $O(b)$ modular divisions.*

Proof According to Step 23 of Algorithm 1, $m \leftarrow v^{1/e} = \prod_{i=1}^{b} v_i^{1/e_i} \bmod N$, with $e = \prod_{i=1}^{b} e_i$, the algorithm can get the result in $O(\log_2 N)$ modular multiplications, which is equivalent to one RSA decryption.

Using the full binary tree as a guide, working from the leaves to the root, for constructing the serial number for every exponent of the leaves, the binary length of the serial number is equal to $\lfloor \log_2 b \rfloor$. In other words, the depth of the leaves is equal to $\lfloor \log_2 b \rfloor$.

According to Step 27 of Algorithm 1, the algorithm takes the recursive result from the left child and right child, and the result associated with this node is $m_R \leftarrow m^X/(v_L^{X_L} \cdot v_R^{X_R})$. The computation phase is to break up the product m to obtain the plaintext $m_i = v_i^{1/e_i}$, which we wish to decrypt simultaneously.

Note that v_l and v_r have already been computed and stored, as the left and right branch values of the root, during the tree-based computation of m at Steps 12–22 of Algorithm 1. By definition, X is the unique solution $((X = 0 \mod E_L)$ and $(X = 1 \bmod E_R) == true)$. Note that $\log_2 X < \log_2 e$ and $X_L \leftarrow X/E_L$; $X_R \leftarrow (X-1)/E_R$, we can get $\log_2 X_L + \log_2 X_R < \log_2 X < \log_2 e$ with $e = \prod_{i=1}^{b} e_i$ [14].

Because the depth of the leaves is equal to $\lfloor \log_2 b \rfloor$, for every plaintext result $m_i = v_i^{1/e_i}$, every node contributes at most $\lfloor \log_2 b \rfloor$ bits to the appropriate exponents in the computation of m^X, $v_L^{X_L}$, and $v_R^{X_R}$ recursive result.

Because the binary length of exponent e_i is $\lceil \log_2 e_i \rceil$, Steps 25–33 of Algorithm 1 can generate the following b results in $O(\log_2 b(\sum_{i=1}^{b} \log_2 e_i))$ modular multiplications or $O(\log_2 b \log_2 e)$ modular multiplications with $e = \prod_{i=1}^{b} e_i$.

To solve for $m_R \leftarrow m^X/(v_L^{X_L} \cdot v_R^{X_R})$, we divide $v_L^{X_L} \cdot v_R^{X_R}$ by m^X, the number of modular divisions required is $O(b)$.

At all, Algorithm 1 can generate the b results:
$m_1^{1/e_1} (\bmod N), m_2^{1/e_2} (\bmod N), \ldots, m_b^{1/e_b} (\bmod N)$
in $O(\log_2 b(\sum_{i=1}^{b} \log_2 e_i) + \log_2 N)$ modular multiplications and $O(b)$ modular divisions.

Then, Lemma 1 is proved. □

Theorem 1 *Choosing the batch size* b, *which satisfies* $2 \leq b \leq \frac{n}{(\log_2 n)^2}$, *and choosing the* e_i *exponents to be polynomial in* n, *we get* $O((\log_2 n)^2 + n)$ *modular multiplications and* $O(\frac{n}{(\log_2 n)^2})$ *modular divisions. Here*, n *is defined as the binary length of modulus* N.

Proof We can easily get $\lceil \log_2 N = n \rceil$, where n is defined as the binary length of modulus N.

We can easily get $\frac{n}{(\log_2 n)^2} < n$, because n is a positive number and $n > 2$.

Also, because the function $\log_2 x$ increases with x, we can get $\log_2 \frac{n}{(\log_2 n)^2} < \log_2 n$. Because of choosing the batch size b, which satisfies $2 \leq b \leq \frac{n}{(\log_2 n)^2}$, we can derive $\log_2 b \leq \log_2 \frac{n}{(\log_2 n)^2}$ and $\log_2 b(\sum_{i=1}^{b} \log_2 e_i) + \log_2 N \leq \log_2 \frac{n}{(\log_2 n)^2}$ $(\sum_{i=1}^{b} \log_2 e_i + \log_2 N)$.

Due to Lemma 1, where $\sum_{i=1}^{b} \log_2 e_i = \log_2 e$ and $\log_2 N = n$, it can be described as:

$\log_2 \frac{n}{(\log_2 n)^2} (\sum_{i=1}^{b} \log_2 e_i + \log_2 N) < \log_2 e(\log_2 n) + n$.

By choosing the e_i exponents to be polynomial in n, thus, $e < n$, the following equation is derived as:

$\log_2 e(\log_2 n) + n < (\log_2 n)^2 + n$.

Then, Theorem 1 is proved. □

The constrained model considering the user requirements-aware security ranking is proposed based on Theorem 1. We optimize the batch size b for a specific modulus size, and obtain better results for smaller batches if the modulus is relatively small. According to Theorem 1, the batch size is optimized as $\log_2 \frac{n}{(\log_2 n)^2}$ in this constrained model.

5 QoS-aware optimization algorithm

Let the decryption time of Algorithm 1 in the SSL handshake time be T_b.

The performance analysis of Algorithm 1 can be divided as multiplication at Steps 16–20, exponentiation at Steps 21–24, and division computation phases at Steps 27–33.

We can estimate the cost of $e = \prod_{i=1}^{b} e_i$ is $(b-1)n^2$. The cost of computing $1/e$ is $(b-1)n^2$. The main computation cost is exponentiation cost $(3k-2)n^2 + o(n^2)$ with the exponent e_i, where the bit length of e_i is k. The whole cost in the multiplication phase is $b((3k/2) + 3) + o(n^2)$.

In the exponentiation computation phase, Steps 21–24 cost $3n^3 + n^2 + o(n^2)$.

In the division phase, Step 31 mainly includes one exponentiation cost $b(3bk - 2)n^2 + o(n^2)$ and two exponentiation cost $(3(b-1)k - 2)n^2 + o(n^2)$. Step 31 also includes one inversion of n-bit integers. Inversion of n-bit integers is equivalent to 20 modular multiplications. The cost of modular multiplication of two n-bit integers can be estimated as $2n^2 + 2n$ [15, 18].

The whole cost in Algorithm 1 is estimated as $3n^3 + n^2(44b + 3b^3 - 1) + o(n^2)$. As a result, the decryption time of Algorithm 1 T_b is estimated as:

$$\left(\frac{3n^3 + n^2(44b + 3b^3 - 1)}{b(3n^3 + n^2)}\right) bT_{rsa} = \left(\frac{3n + (44b + 3b^3k - 1)}{b(3n + 1)}\right) bT_{rsa} \tag{1}$$

Since T_b is the majority of the service time, the batching service time of the server τ is T_b roughly.

Lemma 2 *To satisfy the client's requirement for the stability of the system, the decryption time in SSL handshake T_b is less than the batch size multiplied by the mean Poisson distributed arrival time interval when the time in the Batch Queue Model M/D/*1, *thus*:

$$\tau \approx T_b < b/\lambda \tag{2}$$

Proof Let X_i $(i = 1, 2 \ldots\ldots)$ be the arrival time interval of two consecutive requests, and Y be the time interval of b consecutive requests. Batch Queue Model M/D/1 has been described in our previous work [18].

If the system achieves the stability when the time $t \to \infty$ for the M/D/1 queue model, $T_b < E(Y)$, where $E(Y)$ is the expected value of Y. Because X_i is a random variable with independent identical distribution, the average arrival time interval of b consecutive requests is:

$$E(Y) = E\left(\sum_{i=1}^{b} X_i\right) = bE(X_i) = b/\lambda \tag{3}$$

Then, Lemma 2 is proved. □

Lemma 3 *In the Batch Queue Model M/D/*1, *to satisfy the client's requirement for the stability of the system*, *thus*, $T_q < b/2\lambda$.

Proof In the Batch Queue Model M/D/1, the value of T_q is derived following the equation:

$$\begin{aligned} T_q &= \left(\frac{1 - e^{-\lambda\tau}}{1 - e^{-\lambda\tau} + e^{-1.5\lambda\tau}}\right) T_r = \left(\frac{e^{\lambda\tau} - 1}{e^{-\lambda\tau} - 1 + e^{-0.5\lambda\tau}}\right) T_r \\ &= \left(\frac{1}{1 + \frac{1}{(e^{\lambda\tau} - 1)e^{0.5\lambda\tau}}}\right) T_r, \end{aligned} \tag{4}$$

where $T_r = 0.5\tau$; due to Lemma 2, it can be easily described as:

$$T_q = \left(\frac{0.5\tau}{1 + \frac{1}{(e^{\lambda\tau} - 1)e^{0.5\lambda\tau}}}\right) < \left(\frac{1}{1 + \frac{1}{(e^{b} - 1)e^{0.5b}}}\right)\left(\frac{b}{2\lambda}\right) \tag{5}$$

It can be easily described as when $b \leq 2$,

$$0.944 \approx 1 + \frac{1}{(e^2 - 1)e^{0.5*2}} \leq 1 + \frac{1}{(e^b - 1)e^{0.5b}} < 1. \quad (6)$$

Then, the value bound of the upper limit of T_q is estimated as $[0.944b/2\lambda, b/2\lambda]$.

Then, Lemma 3 is proved. □

Tolerable response time (TRT) is defined as the delay time a client can tolerate between a request for a secure web page and receiving the page [16–19]. The real response time (RRT) is the interval between the receipt of the end of transmission of an SSL-based inquiry message and the beginning of the transmission of a response message to the station originating the SSL handshake.

Theorem 2 *In the Batch Queue Model M/D/*1, *to satisfy the client's requirement for tolerable response time*, $RRT < TRT$, *thus*, $\Rightarrow b < 0.4(\lambda \times TRT + 1)$.

Proof The mean real response time (RRT) is denoted as the sum of T_q, T_c, and the T_b.

In the Batch Queue Model M/D/1, the value bound of the upper limit of T_q is estimated as $b/2\lambda$ derived from Lemma 3 (refer to (6)).

The value bound of the upper limit of T_b is estimated as b/λ derived from Lemma 2 (refer to (2)).

T_c is the mean time of waiting for the other client in the same batching, which is easily derived that the max value of T_c is $(b - 1/\lambda)$.

On the other hand, it is supposed that the solution of b should satisfy the approximate bound, which is derived from Lemma 2, Lemma 3, and described as the following equation:

$$RRT = T_q + T_c + T_b < \frac{b}{2\lambda} + \frac{(b-1)}{\lambda} + \frac{b}{\lambda} < TRT \Rightarrow b < 0.4(\lambda \times TRT + 1) \quad (7)$$

Then, Theorem 2 is proved. □

Combining the user requirements for QoS, such as security ranking, the stability of the system, and tolerable response time, these strategies aim to optimize the parameter b, which is the size of multi-clients for aggregate decryption in Algorithm 1. QoS-aware optimization algorithm satisfies these strategies in this paper.

According to Theorem 2, the initial value of b is estimated at Step 4 of Algorithm 3 with TRT and λ as input values. According to Theorem 1, b is estimated at Step 5 with PKS as input value. If the condition of the optimal batch cannot be satisfied, the algorithm has the ability to fall back on conventional_RSA_decryption(), which is the decryption with plain RSA, which is described at Step 9 and Step 18. The computation of T_b is performed using (1) at Step 12. Step 11 sorts b to satisfy the max solution of $T_b < b/\lambda$ according to Lemma 2 in the descending order from the upper limit computing at Step 7 to two. The computation of T_b_real is performed by executing Algorithm 1.

Algorithm 3 QoS-aware optimization algorithm

```
1: Input: $\lambda$, $TRT$, $PKS$
2: Output: $Optimal\_b$, $T_b$, $T_b$_real, $Speedup$, $Speedup\_real$
3: // Compute the initial value of $b$
4: $initial\_b \leftarrow \int(0.4(\lambda \times TRT + 1))$; (refer to Eq. (7))
5: $n \leftarrow PKS$; $estimate\_b \leftarrow \lfloor \frac{n}{(\log_2 n)^2} \rfloor$
6: if ($estimate\_b < initial\_b$)
7: then{$initial\_b \leftarrow estimate\_b$; Successfind $\leftarrow$ false;}
8: if ($initial\_b <= 1$)
9: then{do conventional_RSA_decryption(); return;}
10: $b \leftarrow initial\_b$;
11: while ($b != 1$) do{
12:   $T_b = (\frac{3n+(44b+3b^3k-1)}{b(3n+1)})bT_{rsa}$ (refer to Eq. (1))
13:   if ($T_b \leq b/\lambda$)
14:   then{$Optimal\_b \leftarrow b$; Successfind $\leftarrow$ true; break; }
15:   elseif($T_b > b/\lambda$)
16:   then $b \leftarrow b - 1$; }
17: if ($initial\_b <= 1$)
18: then {do conventional_RSA_decryption(); return; }
19: call $Algorithm$ 1; Compute $T_b$_ real
20: $Speedup \leftarrow \frac{b(3n+1)}{3n+(44b+3b^3k-1)}$; (refer to Eq. (1))
21: $Speedup\_real \leftarrow bT_{rsa}/T_b$_real;
22: return;
```

6 Validation of analytical models and performance evaluation study

6.1 Validation of analytical models

The analytical results and simulation results are executed on a machine with an Intel Pentium IV processor clocked at 3.20 GHz and 1 GB RAM. Specifically, this paper performs the simulation of the SSL handshake secret exchange algorithm with very small public exponents, namely, $e = 3, 5, 7, 11, 13, 17,$ etc. It is assumed that the value (TRT) is equal to 1 second and 8 seconds as examples both in the analytical model and simulation. It is assumed that the values of the public key sizes (PKS) are equal to 512, 1024, and 2048 bit lengths as examples both in the analytical model and simulation.

Table 1 validates the result of *Optimal_b*, described by the constrained model of the QoS-aware optimization strategy. It is assumed that TRT is equal to 1 s in the analytical model and simulation. Due to small arrival rates, b is almost uniformly calculated by our analytical model. Since arrival rates are small (i.e., $\lambda < 2$), there is very little opportunity to batch, and therefore, the solution of b is relatively small. Even at a higher arrival rate, the analytical result and simulation result are very close to each other. The solution of the optimal batch size is increased with λ, both in the analytical model and simulation when $\lambda < 30$ (i.e., $PKS = 1024$ bits), approximately. Otherwise, RRT is not increased, obviously. The solution of b is decreased with λ when $\lambda > 60$ (i.e., $PKS = 1024$ bits), approximately. The solution of b cannot satisfy the

Table 1 Optimal batch size in constrained model validation

λ/PKS	Optimal_b					
	Analytical model			Simulation results		
	512	1024	2048	512	1024	2048
1	–	–	–	–	–	–
2	2	2	2	2	2	2
3	2	2	2	2	2	2
4	3	3	3	3	3	3
5	4	4	4	4	4	4
10	8	8	6	8	8	6
20	8	10	6	8	10	6
30	8	10	6	8	10	6
40	8	10	–	8	10	–
50	8	10	–	8	10	–
60	8	6	–	8	6	–
70	8	6	–	8	6	–
80	8	6	–	8	6	–
90	6	5	–	6	5	–
100	6	5	–	6	5	–

user requirements for the stability of the system; in other words, the solution of b cannot satisfy $T_b \leq b/\lambda$ according to Lemma 3 when $\lambda > 37.5$, approximately (i.e., $PKS = 2048$ bits).

But, with a non-batching system, it becomes unstable when $\lambda > 1/T_{rsa} = 1/0.16 = 6.25$ for 2048-bit keys due to the fact that a non-batching system becomes unstable when $\lambda > \tau$.

6.2 Performance evaluation

The simulation result of the RSA decryption time T_{rsa} with a larger public exponent, namely $e = 65537$, is about 16, 32, and 130 ms with public modulus N that is 512 bits in length, 1024 bits in length, and 2048 bits in length, respectively, and is tested using reiterative results.

The multi-factor RSA [12] can expect a theoretical speedup of around 2.25 with $n = pqr$ and 3.38 for $n = p^2q$. Experiments show the real speedup to be around 1.73 and 2.3, respectively. Rebalanced RSA offers a theoretical speedup of 3.6, but the actual speedup is 3.2 for 1024-bit keys. Specifically, d is chosen to be close to n such that both d mod $(p-1)$ and d mod $(q-1)$ are small integers [12]. The resulting public exponent e also becomes close to n, which is much larger than typical values (i.e., $e = 3, 17$, or 65537). It is in fact so large that Microsoft Internet Explorer (IE) cannot accept it; SB (Shacham and Boneh) scheme [10] offers a speedup factor of 2.5 for 1024-bit keys. The downside is obvious because CAs charge per certificate regardless of whether the certificate is for the same site or not. It also ignores the satisfaction of the user requirements for QoS, where the batch size is equal to four.

Our algorithm offers a speedup factor of 2.76 (Table 2) for a 1024-bit key, which is used in the SSL handshake protocol frequently. Typically, b is equal to 6 for optimal

Table 2 Speedup of decryption time validation

b/PKS	Speedup of decryption time					
	Speedup			Speedup_real		
	512	1024	2048	512	1024	2048
2	1.81	1.90	1.95	1.78	1.83	1.90
4	3.68	3.05	3.42	3.21	2.93	3.39
6	2.10	2.80	3.82	2.01	2.76	3.80
8	1.28	2.18	3.47	1.14	2.09	3.39

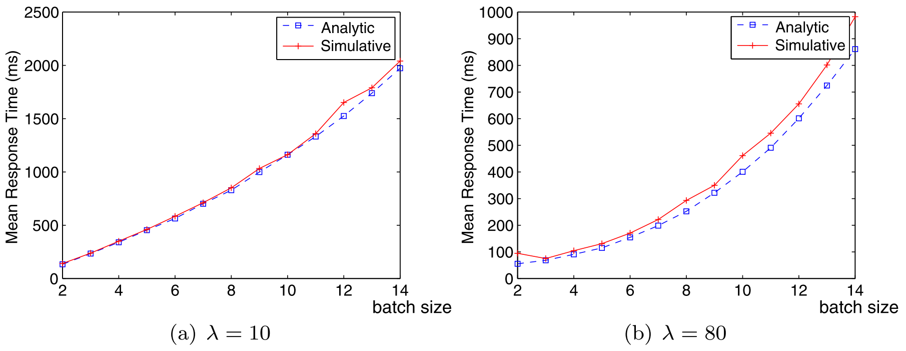


Fig. 2 Mean response time validation over batch size

performance when $60 < \lambda < 80$, approximately (Table 1). Obviously, our scheme not only achieves a better speedup factor, but also overcomes these disadvantages of the previous schemes. All the methods are backward compatible with standard RSA. Also, all speedup methods discussed are based on 1024-bit RSA and are relative to the cost of performing plain RSA decryptions.

It is assumed that TRT is equal to 8 seconds in Fig. 2. Figure 2 shows that RRT is almost linear when λ is relatively small. This is due to the fact that $RRT = T_q + T_c + T_b$ (refer to (7)). When λ is relatively small, the main contribution to RRT is made by T_c (i.e., $\lambda = 10$, $PKS = 1024$ bits). It is evident that the time T_c is increased linearly with b. T_b is also increased with b. Therefore, RRT is also increased with b when λ is relatively large (i.e., $\lambda = 80$, $PKS = 1024$ bits).

Figure 3 illustrates the analytical mean response time RRT, comparing our simulation results and that of the SB (Shacham and Boneh) scheme (i.e. $TRT = 1$ $PKS = 1024$ bits). When λ is approximately equal to 30, RRT reaches its maximum, which is equal to 828 ms, approximately, whereas the value is less than 1 second and decreased with λ when using optimal batch size b (see Fig. 3). In the SB scheme, RRT behaves poorly, especially when λ does not exceed 10 requests/sec (see Fig. 3). When λ is larger than 10 and less than 100, approximately, the performance of our scheme and the SB scheme are all satisfied with the clients' requirement of tolerable response time. However, it is shown that the solutions of b (Table 1) are larger than four. That means the optimal scheme can submit more decryption requests to decryption device than the SB scheme. The analytical and simulation results of RRT shows that our scheme behaves nicely.

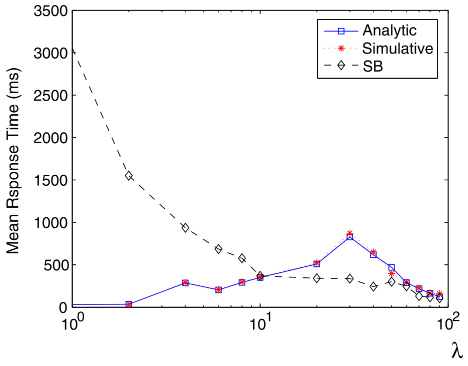
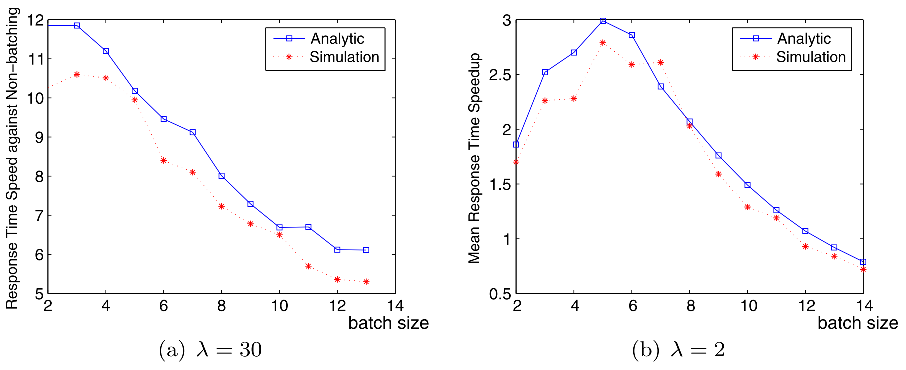


Fig. 3 Mean response time validation over client's tolerable response time



(a) $\lambda = 30$ (b) $\lambda = 2$

Fig. 4 Mean response time speed against non-batching scheme

A non-batching system becomes unstable when $\lambda > 1/T_{rsa} = 1/0.032 = 31.25$ for 1024-bit keys. When the non-batching system is stable, the mean response time T' can be estimated as (8) [18]. The mean service time τ' is deterministic in the non-batching in the M/D/1 queue model. Since T_{rsa} is the majority of the service time, the mean service time τ' of the server is roughly T_{rsa}:

$$T' = \tau' + \tau'\left(\frac{\tau'\lambda}{2(1-\tau'\lambda)} \approx T_{rsa} + T_{rsa}\left(\frac{T_{rsa}\lambda}{2(1-T_{rsa}\lambda)}\right)\right) \tag{8}$$

Figure 4 ($TRT = 8$) illustrates the comparison of the mean response time of our scheme with non-batching scheme. The vertical axis in each graph is the mean response time over the batch size divided by the mean response time with non-batching scheme. The optimal batch size in our scheme is equal to 10 when $\lambda = 30$. The speedup of the mean response time is an optimal one that is equal to 6.69, approximately. The optimal batch size in our scheme is equal to 2 when $\lambda = 2$. The speedup of the mean response time is an optimal one that is equal to 1.94, approximately. It is clear that with the optimal batch size, our scheme has significant advantages while costing less.

7 Conclusion

This paper proposes the secret exchange algorithm in the SSL handshake protocol. This paper also proposes a method of assigning the set of public exponents e_i only using the unique certificate in the SSL handshake protocol. Combining the user requirements for Quality of Service (QoS), such as security ranking, the stability of the system, and the tolerable response time, these strategies aim to optimize the parameter b, which is the size of multi-clients for aggregate decryption. The parameter optimization-based SSL handshake is a viable option for secure communications. Currently, we mainly investigated this work based on the Internet with a high speed client/server computing paradigm. This paper also proposes an optimization strategy based on the constrained model considering the user requirements-aware security ranking. Other optimization strategies are based on the constrained model considering the users' requirements for the stability of the system and the client's requirement for the tolerable time. These optimal strategies focus on the optimal result in different public key sizes. Combining the users' requirements for Quality of Service (QoS), there are many factors which should be considered in real web application development. The evaluation criterion of web service's performance is not only restricted to the response time and the throughput when the server handles SSL requests. In our future work, we will consider establishing more evaluation criterion of web service's performance and propose optimal strategies combing more users' requirements for Quality of Service.

Acknowledgements This work is supported by National Natural Science Foundation of China under Grant Nos. 90718034 and 61073037, Hunan Provincial Natural Science Foundation of China under Grant No. 09JJ4031, Hunan Provincial Science and Technology Program under Grant Nos. 2010GK2003 and 2010GK3005, Central South University Postdoctoral Science Foundation, and Changsha Science and Technology Program under Grant Nos. K1003064-11 and K1003062-11.

References

1. Goldberg I, Wagner D (1996) Randomness and the netscape browser. Dr Dobb's J 21(1):66–70
2. Callegati F, Cerroni W, Ramilli M (2009) Man-in-the-middle attack to the HTTPS protocol. IEEE Secur Priv 7(1):78–81
3. Freier AO, Karlton P, Kocher PC (1996) The SSL protocol V3.0, 1996-11-01
4. Sobh TS, Elgohary A, Zaki M (2008) Performance improvements on the network security protocols. Int J Comput Sci Netw Secur 6(1):103–115
5. Feigenbaum J, Freedman MJ, Sander T, Shostack A (2002) Privacy engineering for digital rights management systems. In: Proceedings of 2001 ACM workshop on security and privacy in digital rights management. LNCS, vol 2320. pp 76–105
6. Rescorla E, Cain A, Korver B (2002) SSLACC: A clustered SSL accelerator. In: Proceedings of the 11th USENIX security conference, San Francisco, August 5–9, pp 229–246
7. Sun HM, Yang CT, Wu ME (2009) Short-exponent RSA, IEICE transactions on fundamentals of electronics. Commun Comput Sci E92-A(3):912–918
8. Kuo FC, Tschofenig H, Meyer F et al (2006) Comparison studies between pre-shared and public key exchange mechanisms for transport layer security. In: Proceedings of IEEE global internet symposium 2006, Spain, pp 1–6
9. Castelluccia C, Mykletun E, Tsudik G (2006) Improving secure server performance by re-balancing SSL/TLS handshakes. In: Proceedings of the 2006 ACM symposium on information, computer and communications security. ACM Press, New York, pp 26–34

10. Shacham H, Boneh D (2001) Improving SSL handshake performance via batching, RSA'2001. In: Lecture notes in computer science. vol 2020. Springer, San Francisco, pp 28–43
11. Takagi T (1997) Fast RSA-type cryptosystems using N-adic expansion. In: Proceedings of crypto '97. Lecture notes in computer science, vol 1294, pp 372–384
12. Boneh D, Shacham H (2002) Fast variants of RSA. RSA Lab Crypt 5(1):1–8
13. Blakey E (2009) Factorizing RSA keys an improved analogue solution. New Gener Comput 27(2):159–176
14. Fiat A (1989) In: Batch RSA, Crypto'89. Springer, Berlin, pp 175–185. See also J Cryptology 10(2):75–88, 1997
15. Menezes A, Van Oorschot P, Vanstone S (1997) Handbook for applied cryptography. CRC Press, Boca Raton
16. Bhatti N, Bouch A, Kuchinsky A (2000) Integrating user-perceived quality into web server design. In: Proceedings of the 9th international world wide web conference, Amsterdam, Netherlands, pp 24–334
17. Bouch A, Kuchinsky A, Bhatti N (2000) Quality is in the eye of the beholder: Meeting user's requirements for internet quality of service. In: Proceedings of the CHI 2000 conference on human factors in computing systems, The Hague, The Netherlands, pp 297–304
18. Qi F, Tang Z, Wang GJ, Wu J (2009) QoS-aware optimization strategy for security ranking in SSL protocol. In: Proceedings of the 2009 IEEE international symposium on trust, security and privacy for pervasive applications (TSP-09), in conjunction with IEEE MASS 2009, Macau SAR, China, pp 842–847
19. Nah F (2004) Study on tolerable waiting time: How long are web users willing to wait? Behav Inf Technol 23:153–163