Reverse SSL: Improved Server Performance and DoS Resistance for SSL Handshakes

Kemal BICAKCI
bicakci@metu.edu.tr
Informatics Institute, Middle East Technical University, Ankara, TURKEY

Bruno Crispo
Computer Science Department, Vrije Universiteit Amsterdam, The Netherlands

Andrew S. Tanenbaum
Computer Science Department, Vrije Universiteit Amsterdam, The Netherlands

Abstract. *Common occurrence of server overload and the threat of denial-of-service (DoS) attacks makes highly desirable to improve the performance and DoS resistance of SSL handshakes. In this paper, we tackle these two related problems by proposing reverse SSL, an extension in which the server is relieved from the heavy public key decryption operation and authenticated by means of a digital signature instead. On the server side, reverse SSL employs online/offline signatures to minimize the online computation required to generate the signature and on the client side, RSA key generation computation can be used as a client puzzle when clients do not have a public key certificate. The preliminary performance results show that reverse SSL is a promising technique for improving the performance and DoS resistance of SSL servers.*

1 Introduction

Although CPU processing power is getting cheaper everyday, today the issue of computational efficiency is still problematic for server machines in contrast to standard personal computers. This is due to common occurrence of server overload in client-server applications.

This performance problem becomes more severe in case when the server is under a denial of service (DoS) attack. Once the clients can request the server to perform computationally expensive operation without doing any work themselves, an adversary can arrange a DoS attack by generating too many requests and exhaust the computational resources of the server.

In our research, we explore these two related problems by focusing on SSL protocol. SSL protocol starts with the SSL handshake which uses public key cryptography so that the server and the client agree on a secret key to be used for securing subsequent communication. SSL handshake protocol comes in two types.

For the first type that supports client authentication, in this paper we propose an extension to SSL called **reverse SSL** so that the methods used to authenticate the client and the server is interchanged. In other words, the server is authenticated by means of generating a digital signature whereas the client is authenticated by public key encryption. This change allows us to utilize the **online/offline signature** primitive in a way that allows the server to perform most of the public key computation offline before the client request. We argue that since the server load is varying with respect to time[1], the underutilized period of time can be used to do

[1] Previous research shows that the workload of a large e-commerce site follows a typical time-of-day pattern. That is the site is busiest during the day and least busy during the early morning [9].

the pre-computation necessary. This leads to a significant improvement in the overall throughput of web servers due to lower online computation requirements of reverse SSL.

The more widely used type of SSL handshake is the one which authenticates only the server hence the client does not need to hold a public key certificate. Reverse SSL is still applicable for this case by asking the client to generate a public key – private key pair on the fly instead of using a long-term certificate issued by a third party. Since the key generation for public key algorithms is expensive, in our protocol with a careful treatment this is used as a **client puzzle,** a countermeasure against DoS attacks. The idea behind client puzzles is simple: ask the client to solve a computational problem before providing resources to that client. By this way, an attacker is prevented to launch the attack without investing considerable resources himself.

The rest of the paper is organized as follows. Section 2 gives an overview of original SSL protocol. Section 3 summarizes the earlier work on online/offline signatures. Section 4 introduces reverse SSL with client authentication. Section 5 summarizes the earlier work on DoS protection and client puzzles. Section 6 introduces reverse SSL with client puzzles as a solution for settings which do not implement client authentication but have a serious concern on DoS attacks. Section 7 provides the results of our experiments conducted to compare the performance of original SSL and reverse SSL in different situations. Section 8 reviews the related work on performance improvement of SSL. Section 9 concludes with some directions for future work.

2 SSL Protocol

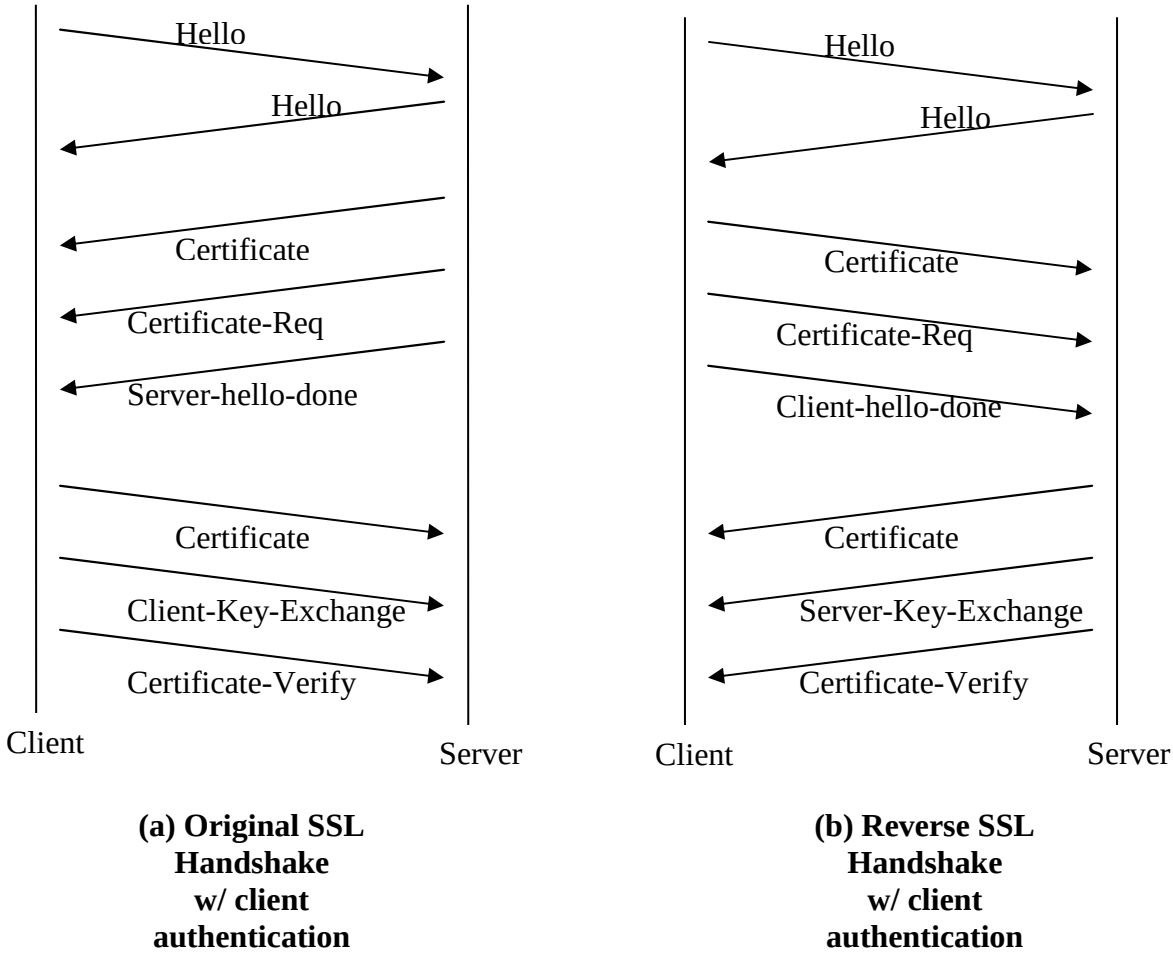
Today, maybe the most widely used security protocol is SSL that becomes the standard way of implementing security for web transactions. Although it was initially designed for web, any network application can use SSL to build an encrypted tunnel for confidential exchange of messages. For doing this, in the SSL handshake protocol, both parties agree on a secret key to be used for subsequent communication. Since the security provided by key exchange without authentication is limited, SSL handshake protocol supports both server and client authentication.

The way SSL implements authentication is by using public-key cryptography which was cited as the most serious performance bottleneck in the SSL operation [1]. Thanks to Moore's law, most client machines today can implement public key cryptography with a reasonable performance (if we exclude very low end devices such as RFIDs, low cost sensors). Contrarily, in spite of doubling CPU speeds for every 18 months, SSL performance of server machines remains an important issue. This is due to the requirement of serving increasingly high number of clients at the same time. An evidence of this problem is the growing demand to specialized hardware devices to accelerate the performance of SSL servers. It was estimated that the revenue for dedicated SSL acceleration hardware and SSL VPN gateways would surpass $1 billion in 2005, a huge jump from 2002 revenue of just $98 million. [2].

Figure 1.a shows the operation of SSL handshake when client authentication is implemented. Note that today in most SSL servers, only server authentication is implemented. We will return to this issue later.

The basic operation of SSL handshake with client authentication is as follows:

After negotiation of some protocol parameters (supported SSL version, encryption algorithms etc.) in the hello messages, the server sends to the client its certificate. Before the server's hello is done, the server also asks to the client his certificate. Now the client finishes the handshake protocol on his side by sending three messages to the server: (1) his certificate (2) a master key encrypted with the server's public key obtained from the server's certificate. (3) all prior communication signed by the client's private key which can be verified using the public key obtained from the client certificate (messages 1 and 3 are empty when there is no



(a) Original SSL Handshake w/ client authentication

(b) Reverse SSL Handshake w/ client authentication

Figure 1. The handshake in original SSL and reverse SSL with client authentication

client authentication). Upon receipt of these messages, the server decrypts the master key with its private key and verifies the certificate and signature to authenticate the identity of the client. In other words, the server does one public-key decryption operation and two or more signature verification operations (one to verify the signature and at least one to verify the certificate).

SSL handshake is completed when the client and server exchange "Finished" messages to verify that the key exchange and authentication processes were successful. Now, both client and server can use the master key to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity.

3. Online/offline Signatures

In 1990, Even et al. introduced the concept of online/offline signatures as a way to get around the fact that most digital signature schemes have high computational requirements [3]. Their online/offline scheme was constructed by joining the general purpose digital signature schemes with the idea of one-time signatures.

Unlike most other signature schemes, one-time signatures can be implemented using only one-way functions therefore they have the advantage of being very fast. On the other hand, these simple schemes have the inherent drawback of signing legitimately only a single message per a given public key (one-time public key). In online/offline signatures, this one-time property does not pose any problem because each one-time public key is signed with the traditional public key signature and this signing can be repeated infinitely. The key point here is that since the one-time public key is independent from the message, it can be signed beforehand. Once the message is ready, the only thing necessary is to sign the message with the ultra-fast one-time signature.

More formally, an online/offline signature scheme consists of the following three algorithms:

> **1. The Key Generation Algorithm (G):** It is as same as the traditional signature scheme. The signer runs the algorithm *G* to the input 1^k for a security parameter *k* and generates signing key *SK* which is kept secret and public key *PK* which should be securely transmitted to the receivers.
>
> **2. The Signing Algorithm (S'):** It consists of two phases.
>
> > **a. Offline Phase:** The signer runs an algorithm *g* on input 1^k to randomly generate a one-time signing key *sk* and the corresponding one-time public key *pk*. He then signs *pk* with SK using the traditional signing algorithm *S* to generate the signature W. The signer stores the triple (*sk,pk,W).*
> >
> > **b. Online Phase:** The signer retrieves *sk* and runs an algorithm *s* to sign the message *M* and generate the one-time signature *w*. The triple *(w,pk,W)* constitutes the signature of *M.*
>
> **3. The Verification Algorithm (V'):** To verify the triple *(w,pk,W),* two verifications are performed. First, the verifier uses the traditional verification algorithm *V* to check *W* is indeed a signature of *pk*. Then, he runs the algorithm *v* to check *w* is indeed a one-time signature of *M*.

For the sake of brevity, we did not include the explanation of three algorithms *g*, *s*, and *v* in the description above. The description of these algorithms in the original one-time signature scheme proposed by Lamport [4] is as follows:

1. **The Key Generation Algorithm (g):**
   a. The one-time signing key *sk* consists of 2*k* elements $x_{i,j}$ generated randomly with $1 \leq i \leq k$ and $j = 0,1$. *(k* is the length of the message in base 2*).*
   b. The one-time public key *pk* is generated by computing $y_{i,j} = f(x_{i,j})$ for all *i,j (f* is a secure one-way function).

2. **The Signing Algorithm (s):** The signature w of a k-bit message $m = m_1\ m_2\ m_3 \ldots m_k$ is $x_{1,m1}\ x_{2,m2}\ x_{3,m3}, \ldots\ x_{k,mk}$.

3. **The Verification Algorithm (v):** The signature is verified by checking if $f(x_{i,mi}) = y_{i,mi}$ holds for all $1 \leq i \leq k$.

The Lamport's scheme is not optimized in terms of signature and public key sizes therefore there are a lot of previous works trying to improve it in various ways. Nevertheless, Lamport's simple scheme remains the most efficient one with respect to signing computation required. The pseudo-code for signing is only four lines of code:

```
for i = 1 to k do begin
        if m_i = 0 then release x_{i,0}
        else release x_{i,1}
end /* for */
```

This means, signing algorithm requires only k binary comparisons for a message of length k. For messages longer than k-bits, just like traditional signatures, a hash function is applied to the message to generate the fingerprint of the message of length k. Thus, for any message length one-time signature generation costs one hash plus k binary comparisons in total.

Other one-time signature schemes succeeded in decreasing the length of signature and/or public key with only a minor increase in the signing computation. For instance in [5], the authors proposed a technique which decreases the signature and public key sizes around their half sizes while keeping the signing computation (excluding hashing the message) less than one hash computation.

Another online/offline signature scheme was proposed by Shamir and Tauman in 2001 [6]. By using a special type of hash function called trapdoor hash function, their scheme reduces the length of the signature in great extent but with an increased online computation requirement (online complexity is equal to 0.1 modular multiplication).

A final note in this section is that some signatures schemes such as DSS and Elgamal can be naturally partitioned into online and offline phases. However the online computation requirement in these signatures is much higher than the online/offline signature scheme that uses one-time signatures.

4. Our Proposal: Reverse SSL

In this section, we introduce our extension to SSL which supports client authentication. Section 6 will show a modification of this extension for the case when client authentication is not required but the server should be protected against DoS attacks.

Today most SSL implementations use RSA algorithm [7] both for encryption and signing tasks. We agree with [8] in believing that this situation will continue in the future because RSA is better understood, explored and documented. The RSA problem is certainly one of the best studied problems in cryptography. Our proposal **Reverse SSL** is built on top of RSA cryptosystem together with a standard hash function which is also in general use in the SSL protocol.

Table 1. A simple performance comparison of different RSA operations

RSA Decryption	Slow
RSA Encryption	Fast
RSA Signature Generation	Slow
RSA Signature Verification	Fast

We will discuss the performance issues in detail later. But for the moment the simple comparison given in table 1 is sufficiently useful and important. For RSA, since public exponent is smaller than the private one, encryption and signature verification can be performed at least an order of magnitude more efficient than decryption and signature generation. Hence in the original SSL protocol given in Figure 1.a, the natural target is to improve the performance of RSA decryption operation. Consequently, a considerable portion of previous work on SSL, summarized in section 8, is on this issue. For instance a recent work [8] claims an RSA decryption speedup by a factor of between 11 and 19. This was shown to be possible by offloading most of server's decryption computation to the client side.

One of our goals in designing reverse SSL is to achieve better than this. Below we briefly explain how this is possible.

First a question: in the table above, RSA decryption and RSA signature verification are cheap. For performance reasons, can we modify SSL so that on the server side only these cheap operations are used?

The answer is unfortunately "no" because if the roles of the client and server are switched and the task of master key decryption is performed by the client not the server, the server is left unauthenticated. If we rely on digital signatures for server authentication, now the server needs to generate a signature which is an expensive operation. So it looks like we do not gain much with this trick.

Fortunately, we have the second trick of online/offline idea that saves our effort. By utilizing online/offline signatures explained in the previous section, most of the computation required for signature generation can be performed offline before the message to be signed is available. Once the message is available, the online computation required for completion of signature is very small (a few microseconds).

The description of reverse SSL illustrated in Figure 1.b is as follows:

Offline Phase: The server executes the offline phase of the signing algorithm (S') given in previous section.

Online Phase: It is composed of following steps:

- Usually in client-server computing the client is the one who instantiates the communication therefore the order of HELLO messages is as same as the original SSL. In these HELLO messages, client and server agrees on which version of SSL will be used. If any party does not support reverse SSL, one of earlier versions will be used instead. Backward compatibility is preserved by this way.

- Now, the roles of client and server are changed. Instead of server, the client sends his certificate and requests the server's certificate. The server verifies the client's certificate and sends three things in return: (1) his certificate (2) a master key encrypted with the client's public key obtained from the client's certificate. (3) the signature of all prior communication.
- For signing the prior communication, the server performs the following:
    - Calculates the hash of prior communication. Here the hash value computed is treated as the message M.
    - Executes the offline phase of the signing algorithm (S') given in previous section.
- The client decrypts the master key in standard way and verifies the server's signature by executing the online/offline verification algorithm V' given in previous section.
- At the end, both parties generate the session key from the master key and the handshake completes.

5. DoS Protection and Client Puzzles

DoS attacks aim at exhausting the server resources by sending a flood of bogus requests so that the server cannot respond timely to the legitimate requests. DoS attacks are simple yet effective attacks against all servers and SSL servers are not an exception. One proposed solution against these attacks is client puzzles which require the client side to perform some computation before his request is satisfied by the server. By this way, the aim is to limit the number of requests that can originate from the attacker. Client puzzles have recently received much attention in the security world in spite of the question mark on how effective client puzzles can be against "distributed DoS attacks" where the attacker breaks onto tens or even hundreds of machines and then uses these "zombie" machines to launch the attack. This is maybe because we do not have a better solution yet to protect against this strong and well-established threat.

Client puzzles were first proposed as a countermeasure against DoS attacks by Juels and Brainard [10]. Then, Dean and Stubblefield used this idea specifically to protect SSL/TLS servers [11]. Broadly speaking, any client puzzle construction should hold at least two properties:

1. The puzzle should be computationally intensive to solve (i.e. no shortcut solutions) so that the attacker must have access to a very large computing resource to respond with sufficient number of puzzle solutions for exhausting the server resources [2].
2. The puzzle verification mechanism should be very lightweight. Otherwise, this mechanism itself will be the target of DoS attack in which the attacker sends to server excessive number of bogus puzzle solutions [12].

6. Reverse SSL with Client Puzzles

Although SSL protocol can authenticate clients with X.509 certificates, most web applications currently in use do not implement this option. This is why from practical point of view, an efficient extension to SSL which works in case when clients do not have certificates is also

[2] Together with this requirement, the previous studies [11] also noted that the computation should be finished in a reasonable time in order not to harm legitimate clients with a slow machine. However it is left open how to determine the optimum point with respect to these two conflicting requirements.

highly desirable. Reverse SSL however requires the clients to be able decrypt the master key encrypted by the server. When the clients do not have a long-term certificate, there are two options left:

1. Before any SSL transaction occurs, the client generates a self-signed public key instead of a third party issued certificate. However we do not recommend using the same RSA key pair for multiple SSL sessions due to the interception risk of corresponding private key when stored on the client machine. Note that original SSL protocol without client authentication does not have this kind of a security problem.

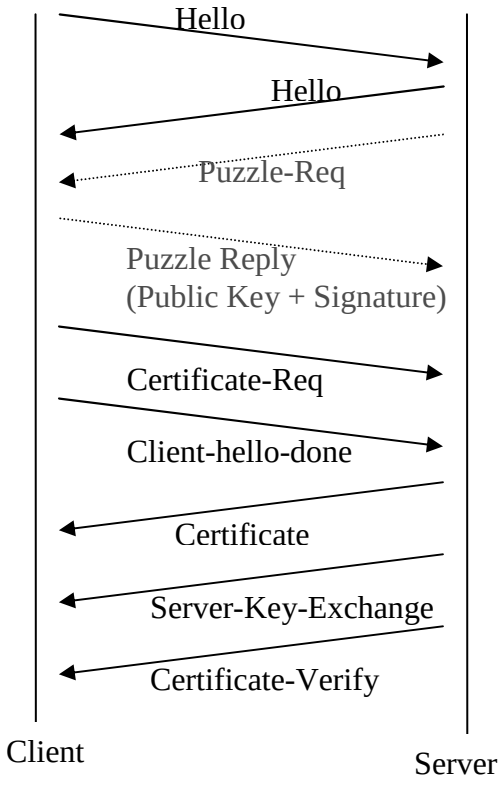


Figure 2. Reverse SSL Handshake with Client Puzzles

2. Client can generate a public key on the fly for each SSL session. Before the SSL session is finished, the private key is destroyed.

In the rest of this section, we will investigate the second option since it does not have a security risk like the first. More specifically, we will show that the computation required for key generation can be used as a client puzzle to safeguard the server against DoS attacks.

Figure 2 demonstrates the operation of reverse SSL with client puzzles. The basic operation of SSL handshake with client puzzles is as follows:

The client notifies the server that he supports reverse SSL with client puzzles in his HELLO message. After its HELLO message, the server sends the client a puzzle and requests a solution. The client's reply to this puzzle has a dual purpose. First, it is a way to show that the client himself has performed a considerable amount of computation before the server responds to his request. Second, the reply includes the client's public key that can be used by

the server to encrypt the master key. The rest of the handshake in as same as the reverse SSL protocol already described.

Next, we provide the description of two different puzzle constructions in reverse SSL. We start with the easier one.

First Puzzle Construction:

- The server generates a big random number (e.g. 1024 bits) *m* ($m \in_R \{0,1\}^{1024}$). He sends *m* and another puzzle parameter *K* (e.g. K = 81*)* to the client. The tuple *(m,K)* constitutes the puzzle request. The meaning of this puzzle request is that the server wants the client to generate an RSA key pair having a modulus in the range $m-2^K$ and $m+2^K$.
- Upon receipt of puzzle request, the client generates two prime numbers *p* and *q* satisfying the following inequalities:

$$\sqrt{\frac{m-2^K}{2}} < p < \sqrt{\frac{m+2^K}{2}}$$

$$\frac{m-2^K}{p} < q < \frac{m+2^K}{p}$$

  The RSA modulus *n* is calculated as n = p * q. The inequalities above guarantee that *n* is between $m-2^K$ and $m+2^K$.
- The client determines the RSA public exponent *e* (or it can have a default value of e = 3) and calculates the private key exponent *d*.
- The client self-signs the public key he has generated and generates the signature *s.*
- The client sends to the server the triple *(n,e,s).*

The client needs to send also a signature otherwise an attacker can generate a fake public key with a modulus *n* in the range $m-2^K$ and $m+2^K$ without taking the trouble to generate the public key - private key pair. Avoiding such a shortcut is possible by asking the client to sign the public key because fake public keys cannot have a signature verifiable by the public key.

However this does not mean that the server should always verify the signature. Later the server sends to the client an encrypted master key. The client's ability to decrypt the master key encrypted by his public key also proves that the public key has been generated legitimately. Recall that in reverse SSL server generates a signature before the client decrypts the master key however this is a very fast operation thanks to online/offline idea. Only if there is not any signature ready for use on the server with a completed offline phase, signature generation computation can be a DoS target. Consequently, the server's decision for verifying the client's signature depends on the availability of online/offline signatures ready to use.

The standard way of generating RSA keys involves two steps: (1) Generating two big prime numbers (2) Computing modulus and exponents. The first step uses a probabilistic primality test (such as Miller-Rabin test [13]) to see whether a candidate chosen odd number passes the test or not. If a number passes the test, we conclude that the number is "probable prime", which is enough for our purposes. The only modification necessary to solve the client puzzle is to choose the candidate odd numbers in the specified range. The rest is exactly the same as the standard RSA key generation.

The reason why the client is forced to generate the modulus in the specified range is to avoid a situation where an attacker uses the same RSA key for multiple SSL sessions. Keeping the information of RSA keys that have been used and refusing a request with an old key is another alternative for the server but has the burden of storing and comparing each RSA key used.

The rationale of choosing the parameter K as 81 is as follows: Breaking a 1024-bit RSA modulus requires approximately 2^{80} operations [14]. The client puzzle we employ should not introduce an attack requiring less than 2^{80} operations. However, an alternative method to find p and therefore break the scheme is to search the space $\left[\sqrt{\frac{m-2^K}{2}}, \sqrt{\frac{m+2^K}{2}}\right]$ for all odd numbers to see whether it is factor of modulus n or not. This requires on average 2^{80} operations.

Another security note about our first client puzzle is that the parameter K is chosen big enough to avoid the problem above but it is small enough not to let previous RSA keys to be valid and in the range specified. The probability of using an old RSA key as a valid solution to a new client puzzle is only $2^{82}/2^{1024}$, a negligible quantity.

Second Puzzle Construction:

If we assess our first client puzzle using the criteria given in the previous section, we see that solving the puzzle requires RSA key generation and RSA signature generation computation whereas puzzle verification is just RSA signature verification. As discussed in section 7, for RSA cryptosystem key generation is much more expensive as compared to RSA signature verification (In the same machine for 1024 bit RSA, key generation plus signature generation takes 318.3 msecs on average whereas signature verification takes only 0.5 msecs).

The second puzzle construction we will explain improves the first one in two aspects:

- Adjustable Solving Time: Our first puzzle is not flexible in the sense that the time required to solve the puzzle cannot be fine-tuned. As we will show, by incorporating a hash-based puzzle into the first one, we can construct puzzles with adjustable solving times.
- Two-Level Verifiability: In the first puzzle construction, although the time required for verifying the puzzle is very small as compared to puzzle generation (as well as compared to the public key computation done by the server in the original SSL protocol), this might still be a target for a DoS attack. Note that the puzzle solving time is not an issue for the attacker sending bogus puzzle replies because he does not spend his time for solving the puzzles. Employing a hash-based puzzle also helps to solve this problem because it enables the server to verify the hash-based puzzle in a much shorter time. Now, the puzzle verification works in two steps. First, hash-based puzzle solution is verified. Only if this puzzle has been verified correctly, the server attempts to verify the RSA signature.

The operation of our second puzzle construction is described as follows:

- The server generates a big random number (e.g. 1024 bits) *m* ($m \in_R \{0,1\}^{1024}$) and determines the puzzle parameter *p*. Let *H()* is a pre-agreed pre-image resistant hash function, highest *p-bits* of m is called as *x* and the rest of *m* is called as *y* (i.e. m = x||y). The server computes *H(m)*. The triple *(H(m),y,K)* constitutes the puzzle request. *K* is a puzzle parameter with the same purpose as the first puzzle.
- Upon receipt of the puzzle, the client tries all possible values of *x* until he finds a match H(x||y) = H(m).
- After finding out what the *m* is, the client goes through the same procedure for RSA key generation given earlier to generate key parameters *(p,q,n,e,d).*
- The client self-signs the public key he has generated and generates a signature *s* and sends to the server the triple *(n,e,s).*
- The server attempts to verify the signature *s* only after verifying that the first *p-bits* of *n* is equal to *x.*

In this new puzzle, the server's computation is only increased by one hash computation. On the other hand the puzzle solution requires on the average 2^{p-1} hash computations in addition to the computation for RSA key generation.

There is one smart thing an attacker can do about this two-level puzzle. After correctly solving the hash-based puzzle, he can generate a bogus public key. We argue that by adjusting *p*, the difficulty of the hash-based puzzle, it is possible to stop attackers to launch this kind of attack, too.

Figure 3 shows a simple flowchart describing the order of operations the server performs after obtaining the client's puzzle reply.

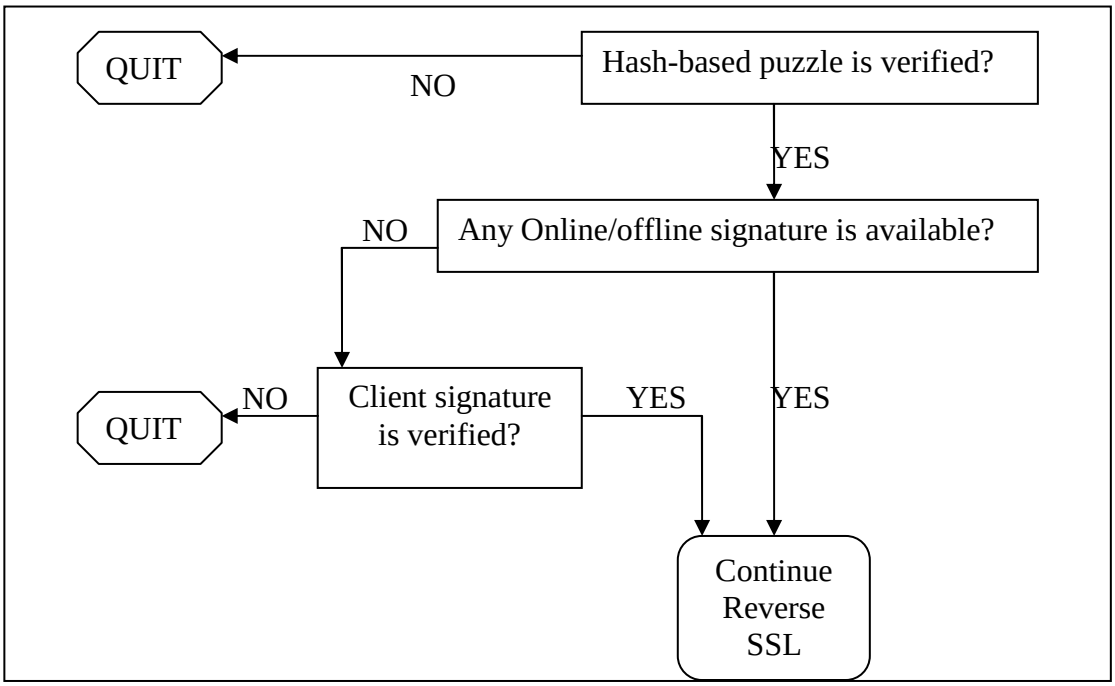


Figure 3. Puzzle Verification Flowchart

7. Experiments and Performance Evaluation

To evaluate the performance of reverse SSL, we did some preliminary experiments by running *openssl speed* command [15] in a server machine (A system with four 1 GHz Sun UltraSPARC CPUs and 8 GB main memory). Computing the hash function took less than 5 microseconds. Since the online requirements for signature generation is less than one hash, we ignore it together with hash computation as they are not significant compared to public key operations.

Next, we measure the performance of public key and private key operations in RSA with various key lengths (*e* is chosen as 65537). Table 2 shows the results of this experiment. If we exclude the computation to verify the client certificate, in the original SSL the server needs to perform one public key and one private key operation. In reverse SSL both with client authentication and client puzzle (by assuming that the server chooses not to verify the signature), the online requirement for the server is just doing one public key operation. Table 3 shows the comparison of computation requirements in these two protocols and the speedup possible using reverse SSL.

Table 2. Performance of RSA public and private key operations

	Private key operation	**Public key operation**	**Ratio**
RSA 512-bits	1.7 ms	0.2 ms	8.5
RSA 1024-bits	8.1 ms	0.5 ms	16.2
RSA 2048-bits	50.0 ms	1.5 ms	33.3
RSA 4096-bits	339.7 ms	5.3 ms	64.1

Table 3. Comparison of Original and Reverse SSL with respect to online computational requirements

	Original SSL	**Reverse SSL**	**Speedup**
RSA 512-bits	1.9 ms	0.2 ms	9.5
RSA 1024-bits	8.6 ms	0.5 ms	17.2
RSA 2048-bits	51.5 ms	1.5 ms	34.3
RSA 4096-bits	345.0 ms	5.3 ms	65.1

The work by Castelluccia et al. [8] has claimed a speedup of 11 for 1024-bit key and 19 for 2048-bit key. For these key lengths, the speedups reverse SSL provides are 17.2 and 34.3 which means at least 50% more improvement [3]. However note that the comparison we provide here is just to give you a basic idea because these two protocols do not have the same functionality i.e. reverse SSL is either for mutual authentication or server authentication with client puzzles whereas early work concentrated on SSL with only server authentication.

Finally, Table 4 provides the results of our experiments regarding RSA key generation. Note that for a client machine with a slower CPU, these numbers are even higher. An interesting note is that unlike other performance values, the results we obtained for key generation has a

[3] *openssl speed* command supports only tests with e = 65537. Choosing e as 3 can provide more performance improvement. It was cited in [18] that 1024-bit RSA public key operation with e=3 is 7.36 times faster than the one with e=65537.

large variance (Table 4 gives the average value computed over 10 different tests). This is due to probabilistic nature of primality test openssl library uses.

Table 4. Performance of RSA key generation

	RSA Key Generation
RSA 512-bits	91 ms
RSA 1024-bits	310 ms
RSA 2048-bits	2924 ms
RSA 4096-bits	24332 ms

8. Related Work

In section 5, we have already summarized the previous work on using client puzzles to protect SSL. The second issue, performance improvement of SSL, has been investigated extensively by security researchers whose main target is the public key decryption operation since SSL is purely CPU bounded [1]. It would be easier to review this previous work by categorizing it into four groups:

1. Using specialized hardware: Offloading RSA computation of SSL to faster dedicated accelerators is a popular but an expensive solution for enhancing SSL performance.
2. Batching: The idea is to perform multiple RSA exponentiations simultaneously. The completion time is up to 2.5 times faster than doing each handshake separately [16].

Unlike the others, the great advantage of these two solutions is the ability to use them without any change on the client side. In other words, they are fully transparent to the client machine.

3. Using different crypto primitives: Instead of RSA cryptosystem, the authors in [16] used ECC (Elliptic Curve Cryptography) and achieve a speedup factor of 2.4. The downside of ECC is that the optimizations that give ECC its performance advantage have patent-related uncertainities [4] [20].
4. Shifting the workload to the client: Although, speeding up SSL handshakes 2.4 or 2.5 times is a notable contribution, one might need even more improvement for heavily loaded web servers. One way for achieving even a better speedup has recently been shown in [8]. The method they use is to ask the clients to do more work that allows the server to do less work. With this so called "client aided RSA" approach, a speedup of 11 for 1024-bit key and 19 for 2048-bit key is possible.

Using the results of our performance evaluation work given in previous section, we can say that our proposal reverse SSL performs considerably better than the previous work (at least 50% more improvement than client-aided RSA).

Online/offline signature idea reverse SSL uses is similar to the recently proposed client puzzle technique in [12]. In this puzzle that uses public key cryptography, the server pre-computes the modular exponentiation in order to reduce the online verification computation only to a table lookup operation.

[4] On the other hand, the patent on online/offline signatures will expire on May 14, 2008 J [19].

9. Conclusion and Future Work

Built op top of online/offline signature idea, reverse SSL is a novel design for improving the performance of heavily loaded SSL servers. By using the asymmetry property of RSA (i.e. small public exponent and big private exponent) and switching the public key operation done by the client and the server, reverse SSL shows how it becomes possible to perform the slower RSA operation totally offline.

Reverse SSL comes in two flavors:

1. Reverse SSL with client authentication: Today, the use of SSL has evolved from simple browser based web transactions to the de-facto transport security, securing corporate communications in the form of SSL VPNs and connections between front-office and back-office applications [17]. Client authentication is required in these new applications and upgrading to a new SSL version is relatively easy for them, therefore reverse SSL is a viable approach to provide the efficiency required by the servers.

2. Reverse SSL with client puzzles: Client puzzle techniques were recently proposed as a countermeasure against denial-of-service attacks. In previous constructions, the puzzle solution has only one purpose and it is dropped after verified. Our new protocol is novel in the sense that the puzzle solution has a second goal and is used additionally as the client's public key required for subsequent encryption. As a result, DoS resistance and performance improvement can be achieved together by using reverse SSL.

Although our initial experiments show that our proposal is attractive, there are a couple of issues that makes essential to support our claim with a full implementation and performance evaluation work that involves the following steps:

1. Implementation of reverse SSL with client authentication using the openssl library [15]. Alternatively, the implementation might be based not on online/offline signature idea but on DSA which is currently available in the openssl library. Recall that generation of DSA signatures can be naturally partitioned into online and offline phases.
2. Implementation of online/offline signatures and incorporating this into the operation of reverse SSL. There are at least two alternatives here but the one based on one-time signatures is straightforward to implement using openssl library.
3. Extending the reverse SSL so that it supports clients without a certificate. To achieve this, client puzzle constructions should be implemented and incorporated into the operation of reverse SSL.
4. Performance comparison of original SSL and reverse SSL both with client authentication and client puzzle and comparison of these results with theoretical expectations. Here, for more realistic results choosing an appropriate statistical model to characterize the client request is needed.

References

[1] Cristian Coarfa, Peter Druschel, Dan S. Wallach, Performance Analysis of TLS Web Servers. *Proceedings of the Network and Distributed System Security Symposium, NDSS 2002*, San Diego, California, USA. The Internet Society 2002.

[2] Neil Osipuk, Secure Socket Layer In an Insecure World. Available at http://www.varbusiness.com/sections/strategy/strategy.jhtml?articleId=18822452, Last access December 7th 2005.

[3] Shimon Even, Oded Goldreich, Silvio Micali, On-Line/Off-Line Digital Signature Schemes. *Advances in Cryptology - CRYPTO '89 Proceedings.* Lecture Notes in Computer Science 435 Springer 1990.

[4] Leslie Lamport, Constructing Digital Signatures from a One Way Function. *SRI International Technical Report CSL-98,* (October 1979).

[5] Kemal Bicakci, Gene Tsudik, Brian Tung, How to construct optimal one-time signatures, *Computer Networks (Elsevier),* Vol43(3), pp. 339-349, October 2003

[6] Adi Shamir, Yael Tauman, Improved Online/Offline Signature Schemes**.** *Advances in Cryptology - CRYPTO 2001 Proceedings.* Lecture Notes in Computer Science 2139 Springer 2001.

[7] Ron L. Rivest, Adi Shamir, and Leonard M. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, 1978.

[8] Claude Castelluccia and Einar Mykletun and Gene Tsudik, Improving Secure Server Performance by Re-balancing SSL/TLS Handshakes, Accepted to *AsiaCCS 2006*. Also Available in *Cryptology ePrint Archive*: Report 2005/037, http://eprint.iacr.org/2005/037

[9] Martin F. Arlitt, Diwakar Krishnamurthy, Jerry Rolia, Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology (TOIT),* Volume 1, Number 1, August 2001.

[10] Ari Juels and John Brainard, Client Puzzles: A Cryptographic Defense Against Connection Depletion. *Proceedings of 5th Network and Distributed Systems Security Symposium,* 1999.

[11] Drew Dean and Adam Stubblefield, Using Client Puzzles to Protect TLS. *In proceedings of 10th USENIX Security Symposium,* 2001.

[12] Brent Waters, Ari Juels, J. Alex Halderman, Edward W. Felten, New client puzzle outsourcing techniques for DoS resistance. *In Proceedings of ACM Conference on Computer and Communications Security,* 2004.

[13] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Fifth Printing (August 2001).

[14] Recommendation for Key Management, NIST Special Publication 800-57 Draft, 08/2005.

[15] OpenSSL Library, Available at http://www.openssl.org.

[16] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, Hans Eberle, Speeding up Secure Web Transactions Using Elliptic Curve Cryptography. *In proceedings of Network and Distributed System Security Symposium NDSS 2004,* Internet Society 2004.

[17] Ncipher's SSL Accelerators, http://www.ncipher.com/ssl/, Last access January 18th 2006.

[18] MIRACL Library, Available at http://indigo.ie/~mscott/.

[19] Micali et al., U.S Patent No. 5,016,274, May 14, 1991.

[20] ECC Patents, http://en.wikipedia.org/wiki/ECC_patents