Reducing the Cost of Session Key Establishment

Bezawada Bruhadeshwar and Kishore Kothapalli and Maddi Sree Deepya
Center for Security Theory and Algorithmic Research
International Institute of Information Technology, Gachibowli
Hyderabad, India 50003.
Email: bezawada@iiit.ac.in kkishore@iiit.ac.in mdeepya@gmail.com

***Abstract*—Scenarios such as online banking, mobile payment systems, stock trading, selling merchandise, and a host of other applications that need a high level of security have moved from the research domain to real world. Moreover, the nature of clients has been changing from traditional desktops to mobile and handheld devices. Protocols like SSL, SSH are the present standard for establishing secure channels. However, the drawback in these protocols is that both the server and the client need to perform computationally expensive public-key operations for secure channel establishment. In this paper, we present simple constructions that spread the cost of secure channel establishment over several sessions. Our constructions are incrementally deployable and can operate with existing protocols such as SSL and SSH. Experimental results indicate that our constructions are practical and efficient in reducing the computational load at the server as well as the client side.**

Key Words. Session Key Establishment, Energy efficiency, Symmetric Key Protocols, SSL, SSH

I. Introduction

A majority of current Internet applications use secure protocols like SSL, IPSec, SSH which are based on public-key cryptosystems for establishing a secure channel between a server and its clients. Due to the tremendous growth in the number of clients using online servers, to maintain performance and scalability there is a need to reduce the computational overhead on the server during the process of secure channel establishment. However, public-key cryptosystems e.g., the RSA algorithm, require considerable computational resources from both the server and the client. Due to performance concerns in heavily loaded servers and the varied nature of clients i.e., desktops to mobile devices, there is a need to reduce the overhead due to public-key cryptosystems for resource constrained clients without compromising security. Thus, reducing the overhead of secure channel establishment in Internet applications is an important problem.

Reducing cryptographic overhead has been the focus of active research for quite some time. The approaches used by existing solutions fall in select categories. One category of approaches [1] studies the overhead of different cryptographic primitives and recommends the best combination of these primitives for a given application or device. Another category of approaches [2], [3] looks at hardware implementation of some or all cryptographic primitives to exploit the speed and efficiency of hardware components. These approaches are important and aid a system designer to go for the best possible solution. Though these schemes are useful, they have some drawbacks. These solutions give a case-by-case study of performance of cryptographic primitives with respect to current hardware/software and do not provide guarantees of performance or address future trends. Moreover, the current results are only relevant with respect to current hardware/software and cannot predict future trends Some of them may also require extensive modification for integrating these approaches with existing protocols.

One way to achieve better performance is to use only symmetric key primitives that are known to be orders of magnitude faster than public-key cryptosystems. Symmetric key primitives are known to be orders of magnitude faster than public-key based primitives. However, it is not possible to use a purely symmetric key based solution as the basic assumption of symmetric key primitives is the existence of a shared secret between the communicating parties. This implies that public-key cryptosystems are indispensable at the current state-of-the-art knowledge. But it is not possible to use a purely symmetric key based solution as we currently require public-key cryptosystems to establish the initial symmetric key between the communicating parties. Note that, this approach is already in vogue in current secure protocols i.e., public-key cryptosystems are initially used to establish the shared secret and the rest of communication use symmetric key primitives. The main drawback in the current protocols is that the process of using public-key cryptosystems is repeated for every session thereby resulting in a high computational overhead at the server and the client. However, we note that, an appropriate combination of public-key cryptosystems and symmetric key primitives can improve the performance of the application as a whole.

In this paper, we propose extensions to the existing model of secure channel establishment to cater to highly dynamic and heterogeneous environments. Our model considers several sessions of a specific client and spreads the cost of secure channel establishment over these sessions. Our approach is stateful i.e., the server keeps a negligible amount of client state that enables the server to achieve the desired efficiency in computation. The nature of our model makes it a good design choice for developing secure Internet applications and protocols in the future. Specifically, our contributions are as follows.

- We describe constructions that provide interesting trade-offs with respect to the following properties: *complexity of computation*, *storage overhead*, *scalability*, and *security*. Our approaches rely on the security of techniques

like symmetric key encryption, integer factorization, one-way hash chains, and key management to achieve the goal of efficient secure channel establishment. Our main contribution is that our model can be implemented by many such constructions and hence, can further improve the performance of Internet applications.
- Our model supports scenarios like client mobility and shared computers. A mobile client uses multiple devices to access the server. The shared computing model is complementary to the mobility model i.e., many clients use the same device to access the server. Current protocols do not provide support for these scenarios due to security concerns.
- We demonstrate the practicality of our schemes using experimental analysis. Towards this, we have chosen the SSL protocol and illustrated the improvements that can be achieved. From our experiments, we observe that our schemes are widely applicable for a variety of server and client environments. Due to the simplicity of our constructions, they are incrementally deployable and inter-operable with existing protocols such as SSL, SSH etc.

Organization. In Section II we describe the problem and the extant literature. Our constructions are described in Section III. This is followed by experimental validation of our constructions in Section IV. We end with some concluding remarks and future work in Section V.

II. Problem Description

We address the problem of reducing the cost of establishing a secure channel between a client and a server especially, when the client needs to establish multiple independent sessions over a period of time. For example, consider a stock investor who needs to connect to his online stock broking account several times during the day. Clearly, in such scenarios, the current approach to establish a session key using public-key cryptography is computationally intensive for both the client and the server. For clients using portable devices this approach is a major drawback. Similarly, servers suffer from scalability issues if the number of such clients is large and dynamic. Hence, there is a need to investigate ways to reduce the cost of session key establishment over multiple sessions. Currently, protocols use public-key cryptography to establish a shared secret between the client and the server. In this case, for each session the client establishes a new session key by using public-key techniques such as Diffie-Hellman key exchange [4]. However, public key based encryption and decryption are computationally expensive and slow when compared to symmetric key operations which are orders of magnitude faster.

A straightforward solution for this problem is for the client and the server to exchange several session keys at the time of initialization and use a different key for every new session. This scheme is computationally effective as the server and the client need only perform the public-key operations during initialization and no additional computation is required for subsequent sessions. This approach also enables the server to perform client authentication e.g., by verifying a message authentication code generated by the client using the current session key. This feature is useful for clients in mobile and shared environments. For example, mobile clients can carry the keys with them in a secure storage and can use them from any location. Also, clients in shared environments can use biometric key generation approaches to generate the keys at runtime and thus, avoid the need to store the keys on the shared device. However, this scheme suffers from two drawbacks, the amount of storage required at the server and the amount of public-key encryption/decryption that needs to be performed at the initialization stage.

From the above discussion, we make the following observations. First, maintaining some amount of client information allows for an efficient session key establishment process. This information allows the server to perform efficient client authentication, a feature that can prevent denial-of-service attacks that cause resource depletion at the server. Second, the storage or computation required at the client and the server depends on the number of sessions for which the session key establishment is desired. Finally, that it is possible to devise similar solutions where the client need not be transfixed to a particular device to benefit from such solutions.

These observations allow us to develop solutions that preserve the above mentioned benefits. Towards this, all our solutions have the following model: the client and the server exchange some additional secret information during the public-key exchange and use this information to establish future session keys. We describe solutions, in an incremental fashion, which provide interesting trade-offs among the parameters of interest: client storage/computation, server storage/computation and number of sessions that can be supported. Given the complexities involved in taking the protocols to implementation, we describe solutions that are efficient, incrementally deployable, and are backward compatible.

Related Work. Several studies [1], [2], [5] have studied cryptographic overhead by varying the combination of cryptographic primitives used and provide appropriate recommendations based on the performance desired. The results of [5] show that the current generation devices have enough compute power to bring cryptography to the realm of the feasible. However, as they point out, no cross-comparison can be done as such studies are highly platform and implementation dependent. In [1]–[3], the authors propose several optimizations for implementing cryptographic algorithms on wireless devices. We note that, such studies complement our work and can be useful in reducing the cryptographic overhead further.

Works based on pre-shared keys [6], [7] have been proposed to enable client authentication as well as to improve the efficiency of session key establishment. In these approaches, a pre-shared key is assumed to be established in advance using an off-line technique. In [6], [7], three such mechanisms that utilize pre-shared keys are proposed and key establishment techniques are discussed. However, these works do not address the problem of establishing multiple session keys in an automated manner as is possible in our approaches. In [8], a mechanism based on modular exponentiation, similar to the Diffie-Hellman key exchange, is described to establish

passwords remotely and securely. However, our goal is to move away from schemes involving expensive operations like modular exponentiation.

Formal analysis of existing protocols has been done in [9], [10]. The authors identify several possible attacks and solutions. Specifically, the authors of [9] identify pitfalls of using the "is-resumable" option in SSL. This option allows the client and the server to reuse existing keys to generate future session keys. The authors prove that this option suffers from cipher-suite rollback attack which compromises the security of the connection. In [11], the authors present techniques that allow the server to authenticate users based on the session information. Our solutions can leverage the strong authentication guarantees provided by such approaches and reduce the need for strong user authentication for every session. Finally, we note that, our approaches are generic in nature and can benefit any public-key based protocol implementation.

III. Constructions for Session Key Establishment

We describe four constructions, in an incremental fashion i.e., each solution provides interesting tradeoffs among the parameters of interest: client storage/computation, server storage/computation and number of sessions that can be supported. Our solutions use existing cryptographic primitives and hence, the security of our solutions are equivalent to the security of the primitives used. In the following description, for all practical purposes, the terms *shared secret* and *session key* are interchangeable i.e., it means that the shared secret between the client and the server can be used as the session key or the shared secret can be used by both the client and the server to derive a symmetric session key. Also, we assume that the session key for each session is unique.

A. Construction A: Using Symmetric Keys

In the simplest construction, the client uses the current session key to encrypt the next session key and sends it to the server. The client chooses a random value, K_{next}, as the next session key and encrypts K_{next} using the current session key, K_{curr}, as follows: $E_{K_{\text{curr}}}(K_{\text{next}})$. Now, the client communicates this value to the server. However, in this construction, the forward secrecy i.e., security of future sessions is compromised if the current session key is broken by an attacker.

We note that this construction can be protected against the afore-mentioned attack by using random cryptographic salts. Towards this, during the initial public key exchange process, along with the current session key, the client sends two salt values s_1, s_2 where s_1 is the salt for the encrypting key and s_2 is the salt for the new session key being sent. Now, the technique to establish the next session key is as follows: the new session key, $K_{\text{next}} \oplus s_2$ is encrypted using $K_{\text{curr}} \oplus s_1$ and is sent to the server. This technique, although is quite secure, is still vulnerable to the following attack. Consider the case where the attacker is successful in compromising the encrypting keys for two consecutive session key exchanges, say, $K_0 \oplus s_1$ and $K_1 \oplus s_1$ respectively. Clearly, the attacker also has access to the encrypted values, $K_1 \oplus s_2$ and $K_2 \oplus s_2$. From these values, using simple XOR operations, the encrypting key for the next session key exchange can be derived as follows: $(K_1 \oplus s_2) \oplus (K_1 \oplus s_1) \oplus (K_2 \oplus s_2)$ which is equal to $K_2 \oplus s_1$. This is the encrypting key used by the client to communicate the next session key. Hence, the attacker can compromise all future sessions. We note that, however, due the presence of the random salts, this attack is more difficult than the attack on the simple approach as the attacker needs to compromise two consecutive sessions. Thus, this technique trades-off security for reduced storage and performance.

B. Construction B: Using One-way Functions

We consider a construction using one-way functions i.e., a function that is difficult to invert. Consider, for example, a one-way function based on integer factorization. which is given by function $f(x, y) = x \cdot y$. Given $x \cdot y$ for two sufficiently large primes x, y, it is believed to be computationally difficult to determine x and y. The construction using integer factorization is as follows. For session key establishment over n sessions, the client generates n primes $p_1, p_2, \cdots, p_n$ and computes $N = p_1 \cdot p_2 \cdots p_n$. Now, the client sends N and p_1 to the server using the initial public-key exchange process. The value p_1 serves as the shared secret for the first session. For the next session, the client sends the value p_2 to the server by encrypting p_2 with a key generated using p_1. The server can verify the validity of p_2 by checking if p_2 divides N. This is repeated for all n sessions i.e., for the i^{th} session p_i is revealed by the client. This solution can also provide cross-verification which further improves the security. For example, the server can request the client to reveal additional primes, say p_3, p_6 etc, to protect against guessing attacks and replay attacks.

We note that, there are certain drawbacks to this solution. First, the client needs to choose large primes to prevent brute-force attacks. For low powered devices this cost may be prohibitive. Second, the initial cost of setup is high as a large composite number needs to be communicated to the server. This also requires a high server storage per client. However, a different choice of the one-way function may reduce the computational and storage overhead. Thus, this solution provides good security while trading-off performance, storage and initialization cost.

C. Construction C: Using One-way Hash Chain

In [12], Lamport proposed one-way hash chains for one-time password authentication. We use this important cryptographic primitive to describe our next construction. First, the client chooses a random seed r and computes $h^n(r)$ where h is a hash function, say SHA, that is repeatedly applied n times over the seed r. Next, using the public-key exchange, the client communicates $h^n(r)$ to the server. This value serves as the shared secret for the first session. The shared secret for the $(n-k+1)^{th}$ session is given by $h^k(r)$ where $1 \leq k \leq n$. For example, the shared secret for the second session is given by $h^{n-1}(r)$. Furthermore, to communicate the session key for the $(n-k)^{th}$ session i.e., $h^{k-1}(r)$, the client computes $h^{k-1}(r) \oplus h^{k+1}(r)$, encrypts this value using a symmetric key generated from $h^k(r) \oplus h^{2k+1}(r)$ and sends it to the

server. The server can decrypt the client's message as it has all the necessary information and also, can verify that this value originated from the client by hashing $h^{k-1}(r)$ and comparing it with $h^k(r)$. This is because it is known to be computationally difficult to get $h^{k-1}(r)$ from $h^k(r)$.

We note that, due to the choice of our parameters for the hashing, this scheme is secure against the attack that was possible in Construction A. Using experimental results, we show that the computational cost at the client and the server are small enough to make the scheme very practical. The storage cost at the client and the server is small which makes this scheme applicable to resource-constrained systems However, for large values of n, the computational savings achieved may not be significant and can actually exceed the computation under public-key based solutions. Thus, this solution provides good security, small storage but trades-off performance for large number of sessions.

D. Construction D: Using Logarithmic Keying

The previous constructions provided varying trade-offs among storage, computation and number of sessions supported without providing a balanced solution. For this construction, we use key distribution similar to [13], and describe a technique that provides an appropriate balance among these parameters. The cost advantage using this scheme is exponential when compared to the previous constructions.

During the initial public-key exchange with the server, for n sessions, the client generates $2\log n$ symmetric keys along with a pre-determined message M (of small size say 64 bytes) and sends these values to the server. Each session is represented by a $\log n$-bit binary string. Also, each bit-position is associated with two unique keys, one each for bit values 0 and 1, which are chosen from the set of the $2\log n$ keys. Formally, if i denotes the bit-position then the two keys corresponding to this bit-position can be denoted by K_i^v where $v \in \{0, 1\}$ i.e., the key corresponding to bit-value 0 is K_i^0 and the key corresponding to bit-value 1 is K_i^1. Since there are $\log n$ bits, $2\log n$ keys would suffice for this key distribution. By construction, the ith session S_i, $1 \leq i \leq n$, is represented by a unique $\log n$-bit identifier, $i_1 i_2 \ldots i_{\log n}$ where $i_j \in \{0, 1\}$ for $1 \leq j \leq \log n$. Given this encoding, the session key S_i is computed as follows. For each bit value i_j in the session identifier for S_i, the client chooses the key $K_j^{i_j}$, $1 \leq j \leq \log n$, and uses these keys to compute $K^{(i)} = \oplus_{j=1}^{\log n} K_j^{i_j}$ which will be used as the current session key. The client then computes a secure hash of M using $K^{(i)}$ and communicates this value to the server. The server, upon receiving this value, computes $K^{(i)}$, and verifies the hash sent by the client.

We note that, by design, using the above construction, the pool of keys for supporting n sessions cannot be more than $2\log n$. Furthermore, the construction can be generalized as follows: from a pool of keys, a unique subset can be chosen and used to compute the session key for a particular session. It can be trivially shown that the number of subsets of size k i.e., $\binom{2\log n}{k}$, for some $k < \log n$, is greater than n. The value of k can be appropriately chosen using Stirling's approximation. Thus, a larger number of sessions can be supported by only slightly increasing the number of keys.

This construction has several advantages. First, computing the session key is simple and fast as it only involves XOR operations which are much faster than even computing a hash. Second, the storage required at the server is small enough to make it practical. Third, the security of the construction is tied to the ability of an attacker to invert hash function such as SHA-1. Moreover, even if the hash function is inverted, the attacker can only obtain the pre-determined message M but not the keys $(K^{(i)})$ under which the hash is computed. The initial set up cost is also quite small and can be reduced further as follows. Instead of using the public-key to encrypt the set S and the message M, the client can use an initial session key to send these values. However, this comes with the disadvantage that if this symmetric session key is compromised, then all the n sessions are also compromised.

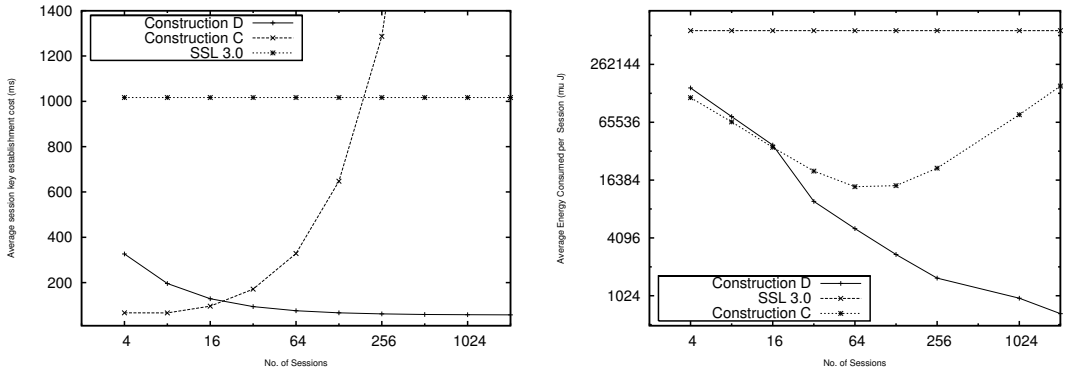
IV. Experiments

In this section, we evaluate the performance of our constructions using experimental analysis. We chose the SSL/TLS protocol [14], which is the de facto protocol for secure communication using the Internet, and compared our results with this protocol.

Experimental Methodology. We focused our experiments on measuring the computational savings and the energy savings using Constructions C and D. To this effect, we analyzed the operations that take place during the SSL handshake protocol and obtained estimates on the cost of operations such as computing a hash using SHA-1, RSA encryption/decryption, computing the pre-master secret and deriving the session key. To obtain the times for cryptographic operations we used the results reported in [15] and for energy measurements we used the results reported in [3].

Results. As can be seen in Figure 1(a), the average session key establishment time can be drastically reduced even for 1024 sessions using Construction D. The reason for this is that by initially exchanging a small number of keys, say 20 symmetric keys, using public-key handshake, about 1024 session keys can be established in a light-weight manner. On average, this implies that the cost of 1 public-key exchange is spread over 50 sessions. Construction C exhibits other interesting phenomenon. As a large number of hashes have to be computed to establish a large number of sessions, the cost advantage compared to SSL 3.0 [14] diminishes around 128 sessions. These results suggest for very dynamic environments, Construction D is more practical whereas for a few sessions, say 20, Construction C is preferable as it can avoid public-key operations within the given time frame.

In Figure 1(b), we show the average energy consumption per session for our constructions in comparison with SSL 3.0 [14]. For construction C, the energy consumed in the first session following SSL protocol dominates the average initially. For about 32 sessions or higher, this effect starts to fade off. However, for more sessions, as the number of hashes to be computed increases, the average energy consumed witnesses a raise but still continues to be less than public-key operations.



(a) (b)

Fig. 1. Average Cost of (a) Session Key Establishment (ms) (b) Energy Consumption (micro Joules)

Although, at first glance, it appears that Construction C requires computing lot of hashes, it is however possible to reduce this cost by storing intermediate hash values [16] at the expense of storage. Finally, for Construction D, the average energy required is quite small due to its simplicity. From these results, we note that, a system designer can choose an appropriate construction based on the parameters that need to be optimized.

V. Conclusion

In this paper, we focused on the problem of reducing session key establishment over multiple sessions for large, dynamic and heterogeneous environment and proposed a model for designing solutions for this problem. Using a mix of public-key and symmetric key primitives, we described several constructions that offered varying tradeoffs among different parameters. We have given an empirical analysis of our constructions which show that these constructions are practical. In future, we plan to investigate ways to integrate some of our constructions into the existing protocols.

References

[1] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha. Securing electronic commerce: reducing the SSL overhead. *IEEE Network*, 14(4):8–16, 2000.

[2] N. Potlapally, S. Ravi, A. Raghunathan, and G. Lakshminarayana. Optimizing public-key encryption for wireless clients, 2002.

[3] N. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Trans. Mob. Comp.*, 5(2), 2006.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Th.*, IT-22:644–654, 1976.

[5] P. G. Argyroudis, R. Verma, H. Tewari, and D. O'Mahony. Performance analysis of cryptographic protocols on handheld devices. In *Proc. IEEE Intl. Symp. on Net. Comp. and App.*, pages 169–174, 2004.

[6] P. Eronen and H. Tschofenig. Pre-shared key ciphersuites for Transport Layer Security. IETF, RFC 4279, 2005.

[7] F.-C. Kuo, H. Tschofenig, F. Meyer, and X. Fu. Comparison studies between pre-shared and public key exchange mechanisms for transport layer security. *IEEE INFOCOMM*, pages 1–6, 2006.

[8] T. Wu. The secure remote password protocol. IETF RFC 2945, 2000.

[9] J. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proc. of USENIX Sec. Symp.*, pages 16–16, 1998.

[10] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proc. USENIX W. on Elec. Commerce*, 1996.

[11] R. Oppliger, R. Hauser, and D. Basin. Ssl/tls session-aware user authentication. *IEEE Computer*, 41(3):59–65, 2008.

[12] Leslie Lamport. Password authentication with insecure communication, 1981.

[13] M. G. Gouda, S. Kulkarni, and S. Elmallah. Logarithmic keying of communication networks. In *Proc. of SSS*, 2006.

[14] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0. draft-ietf-tls-ssl-version3-00.txt, 1996.

[15] A. Kahate. *Cryptography and Network Security*. Tata McGraw Hill, 2005.

[16] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proc. of Conf. on Fin. Crypt.*