

IAuth: An Authentication System for Internet Applications

Suan-Suan Chew, Kok-Leong Ng, Chye-Lin Chee
Department of Information Systems and Computer Science
National University of Singapore

Abstract

With the advent of the Internet, an era of distributed computing is rapidly taking hold. The potential offered by distributed computing increases opportunities for security breaches. This emphasizes the importance of security practices, such as authentication. We have designed a system, IAuth, which provides secure distribution of cryptographic keys while establishing authenticity between a user and a Web-based application. The strength of IAuth in the context of the Internet is that there is no need for a user to possess a cryptographic key if the application requires data encryption or digital signing. Once verified authentic, the user can securely obtain his cryptographic key from the Web application server for data encryption or digital signing.

Keywords: Distributed computing, client-server, security, authentication, cryptography, Internet, world-wide-web.

1. Motivation and Objectives

With the advent of the Internet, an era of distributed computing is rapidly taking hold. Distributed development architectures allow applications to be divided into components, each of which can exist in different locations. This new development paradigm increases opportunities for security breaches. Hence, the potential offered by distributed computing and the Internet carries with it the necessity to exercise security practices.

Good security practices include a number of important attributes - authentication, authorization, confidentiality, containment, auditing and non-repudiation.

The aim of this project is to develop an authentication system for secure Web-based applications. The system should perform *authentication* and provide support for *confidentiality* and *non-repudiation*. The first task of the system is to authenticate remote users with the Web application server. The system needs to assure users that the server is genuine, and to assure the server that a user requesting its service is authentic. Support for confidentiality requires cryptographic means to ensure that data exchanged remains private. Support for non-repudiation also requires cryptographic means to

ensure that communicating principals cannot disavow knowledge of a transaction. To support both confidentiality and non-repudiation, the system must establish a cryptographic key while establishing authenticity.

The objective of this project is two-fold. Our primary goal is to design the authentication system by incorporating good protocol design principles used in various authentication systems. Our secondary objective is to demonstrate the usage of the authentication system by implementing a prototype and testing it with a simple Web application. The prototype will serve as a model for future improvements on the system.

2. Types of Authentication

Authentication is the process of reliably verifying the identity of an entity [4]. There are various forms of authentication - password-based authentication, address-based authentication and cryptographic authentication. When we use password-based authentication, we share a secret quality, the password, with someone to whom we send the password, proving that we know it. The problem with password-based authentication is eavesdropping and impersonation. Address-based authentication does not rely on sending passwords, but assumes that the identity of the source can be inferred based on the network address from which packets arrive. Address-based authentication shares the same weakness as password-based authentication. Cryptographic authentication protocols are more secure than password-based or address-based authentication. The basic idea is that we prove our identity to someone by performing a cryptographic operation on a quantity supplied by that someone. The cryptographic operation is performed based on some shared secret.

Password-based and address-based authentication is unsafe for Web-based applications since IP spoofing and intercepting passwords on the Internet is easy. Authentication schemes based on cryptography are more desirable. When using cryptography in authentication, an intruder listening to the network gains no information that would enable it to falsely claim another person's identity.

3. Cryptographic Authentication Protocols

For discussing protocols in this paper, we represent two communicating principals requiring authentication as Alice and Bob. The cipher notation used is $E\{M/K_{AB}\}$ which reads as message M encrypted with cipher function E using key K_{AB} .

3.1. Kerberos

The Kerberos [1][2][3][4] authentication system performs both authentication and key distribution. The entities involved in a Kerberos authentication process are the client application, the server application and the Kerberos *Authentication Server* or *Key Distribution Center* (AS/KDC). Each client and server application shares with the Kerberos authentication server, a unique secret encryption key which each uses to communicate with the authentication server. The authentication server provides both the client and server applications with a secret session key for communicating with each other.

When a client application wants to communicate with a server application, it requests for a Kerberos *ticket* from the Kerberos authentication server. The authentication server authenticates the client application indirectly through a user password. Once the client is verified to be authentic, the authentication server sends a Kerberos ticket to the server application through the client application. The Kerberos ticket is a certificate which contains a secret session key, the name of the principal to whom the session key was issued, and an expiration time after which the session key is no longer valid. Before the client application communicates with the server application using the secret session key, it has to present an *authenticator*. An authenticator includes current time and checksum fields encrypted with the session key, which are checked for validity. The client is authentic if the authenticator is valid.

A security weakness of Kerberos is that the initial request message from the client application to the authentication server is unencrypted. This allows an intruder to listen for and collect tickets issued by the authentication server for message replay attacks. Although Kerberos tickets are timestamped to counter message replay attacks, a clock-based scheme offers doubtful security since it relies on the performance and security of clock synchronization protocols used. Also, it is unwise to underestimate the capability or the readiness of an intruder to replay a ticket within the ticket acceptance window.

Kerberos is inappropriate in the context of the Internet due to extensive interprocess messaging required for authentication. Also, the client application needs to obtain a ticket from the authentication server in order to communicate with the server application. This does not make sense since clients initiate requests directly to a server application on the Web server through the server's URL. But the concept of a KDC, which serves as a trusted third-party for secure storage and distribution of encryption keys is a desirable feature, especially in the context of key management.

The security flaw in the first unencrypted message of Kerberos can be overcome by using pre-authentication and ensuring that the initial authentication request is encrypted. To counter message replay attacks, instead of using a clock-based scheme, a better way is to use a challenge-response scheme.

3.2. KryptoKnight

KryptoKnight [6] is a secret-key based authentication and key distribution system similar to Kerberos. KryptoKnight differs from Kerberos in the following ways:

1. KryptoKnight uses message digest (hash) functions for authentication and encryption of tickets. This offers performance advantages.
2. In Kerberos, a client application must obtain a ticket before initiating communication with the server application. In KryptoKnight, there is an option for the client application to request communication with the server application directly, and have the server do the handshaking with the KDC. This is advantageous where it is less expensive for the server than for the client application to communicate with the KDC.
3. KryptoKnight does not rely on synchronized clocks, but uses random number challenges instead.
4. KryptoKnight uses more messages as it requires an extra message to transmit a challenge.
5. Other than pre-authentication, KryptoKnight does not complicate its design with the other features added to Kerberos in version 5.

The concept of a *server-to-KDC* handshake is applicable to our system since it is less expensive and more convenient for the Web application server, rather than the client application, to communicate with the KDC. The use of hash functions, random challenge schemes and pre-authentication are also good design principles to incorporate into our design.

3.3. Bellovin-Merritt Protocol

In most authentication protocols, keys are derived from a password. Whenever a key is derived from a password, there is the potential for off-line password-guessing attacks. In this section, we discuss two protocols by Bellovin and Merritt [5] designed to prevent off-line password-guessing attacks. We are not aware of any systems based on these as they are relatively recent. Our system closely models the Bellovin and Merritt protocols.

3.3.1. Bellovin-Merritt-1

The basic idea is that Alice and Bob do a Diffie-Hellman key exchange [7], but encrypt the values they send to each other with a shared password. The Diffie-Hellman key exchange is a key generation and agreement scheme

whereby two principals without any prior arrangements can agree upon a secret key that is known only to them, and, in particular, not known to an eavesdropper listening to the dialogue by which the principals agree on the key. This secret key can then be used, for example, to encrypt further communication between the principals. The Diffie-Hellman key exchange can be applied in protocols for establishing secure connections.

Diffie-Hellman works in the following way. Assume that the communicating principals are Alice and Bob. To start out, Alice and Bob agree on numbers p , a large prime, and g , which is a primitive root modulus p . p and g are pre-established and known to Alice and Bob. Each of them chooses a secret random number. Let $R-A$ represent Alice's secret random number, and $R-B$ represent Bob's secret random number. Each raises g to their secret number modulus p . The result is that Alice computes some number T_A and Bob computes some number T_B . They exchange their T 's. Finally, each raises the received T to their secret number. After the Diffie-Hellman exchange, Alice and Bob will agree on a Diffie-Hellman key, K .

Alice selects R-A at random.	Bob selects R-B at random.
Alice computes $T_A = g^{R-A} \text{ mod } p$.	Bob computes $T_B = g^{R-B} \text{ mod } p$.
They exchange $T_A \leftrightarrow T_B$	
Each raises the number they receive to their secret random number.	
Alice computes $T_B^{R-A} \text{ mod } p$.	Bob computes $T_A^{R-B} \text{ mod } p$.
Both come up with the same number, K .	
$K = T_B^{R-A} = (g^{R-B})^{R-A} = g^{R-BR-A} = g^{R-AR-B} = (g^{R-A})^{R-B} = T_A^{R-B} \text{ mod } p$	

Figure 1. Diffie-Hellman Key Exchange

In Bellovin-Merritt-1, Alice and Bob do a Diffie-Hellman key exchange, but encrypt the messages exchanged using a shared key, K_{AB} . Then, they do a standard mutual authentication exchange proving to each other that they both know K .

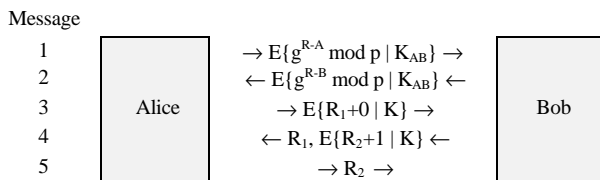


Figure 2. Bellovin-Merritt-1 Message Flow

Messages 1 and 2 are the Diffie-Hellman key exchange encrypted with a user password, K_{AB} . After messages 1 and 2, assuming that Alice and Bob shares the same password, they would have established and agreed on a secret key, K . Messages 3, 4 and 5 is the standard mutual authentication three-way handshake

extended to prevent a reflection attack¹ by concatenating sequence numbers 0 to R_1 and 1 to R_2 .

The Diffie-Hellman key exchange is encrypted so that an eavesdropper would be unable to find the shared user password or the Diffie-Hellman key exchanged based on seeing unencrypted versions of messages 1 and 2. If Diffie-Hellman is not encrypted, an attacker can gain information for doing password-guessing attacks by either impersonating Alice or impersonating Bob. The Diffie-Hellman exchange would have only established a secret key at that point and not authenticated both parties.

Can an eavesdropper attack? There is no way to do a password-guessing attack based on $E\{g^R \text{ mod } p \mid K_{AB}\}$ because if the eavesdropper were to guess a password and use that password to decrypt the transmissions, there is no way to verify that the result after decryption is indeed $g^{R-A} \text{ mod } p$ and $g^{R-B} \text{ mod } p$, since those would be indistinguishable from random numbers. There is no way to do a password-guessing attack based on messages 3 to 5 because they are based on a key that has nothing to do with the shared user password (unless Diffie-Hellman can be broken).

3.3.2. Bellovin-Merritt-2

Bellovin-Merritt-1 has a weakness in that both Alice and Bob must share a user password, K_{AB} . Bellovin-Merritt-2 was designed to overcome this, in which Alice holds the user password but Bob only knows a hash of the user password. In this way, the hashed password is used by Bob to verify that Alice knows the password, but cannot be used by an attacker who stole it from Bob's database to impersonate Alice to Bob.

Bellovin-Merritt-2 extends the key-establishing property of Bellovin-Merritt-1, by providing a secure means to communicate a secret component (a private key for instance) from one principal to the other. The objective of Bellovin-Merritt-2 is to secure the exchange of a secret component, by encrypting the exchange of this secret component using a Diffie-Hellman key, which is also secret. By using the encrypted Diffie-Hellman, authentication is also established in the process.

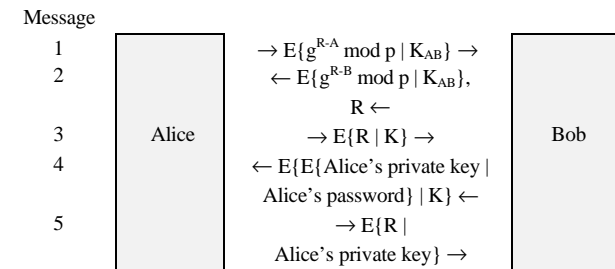
In essence, a principal starts out with zero knowledge of a secret component, the encryption key for instance. It only knows its user password which has been pre-established with another principal which holds its encryption key. Using its password, a principal authenticates itself to the principal holding its encryption key, which it then obtains securely. To prevent stealing, the principal providing the encryption key, only stores a

¹ Reflection attacks afflict multi-session environments which use challenges. The attacker who does not hold a key (and thus cannot encrypt a challenge) can respond to the challenge of the first authentication session by obtaining an encrypted form of the challenge through a second session.

hash of the user password and the encryption key encrypted with the user password.

This protocol can be applied to simplify both data encryption and digital signing, since there is no requirement for a principal to securely possess its private key or secret encryption key. To sign a message, a principal only needs to supply its user password to securely obtain its private key for digital signing. To decrypt a message, a principal can obtain its decryption key securely using its user password.

Bellovin-Merritt-2 can be illustrated as follows. Alice has a private key which Bob stores encrypted under Alice's password and which he gives to Alice after she proves that she knows a hash of her password. She then authenticates again using her private key to encrypt challenges from Bob, proving that she knows the password itself.



where Bob knows $K_{AB} = \text{hash}(\text{Alice's password})$ and $E\{\text{Alice's private key} \mid \text{Alice's password}\}$.

Figure 3. Bellovin-Merritt-2 Message Flow

In messages 1 and 2, Alice and Bob do a Diffie-Hellman exchange encrypted with a hash of Alice's password. Piggy-backed onto the second message of the Diffie-Hellman exchange is a challenge R , generated by Bob. At this stage, Alice and Bob have established a shared key $K = g^{R \cdot AR^B} \bmod p$ and Alice knows Bob's challenge R .

In message 3, Alice proves to Bob that she knows K by encrypting R . This means that she knows K_{AB} , because the Diffie-Hellman exchange was encrypted using K_{AB} . However, this does not prove that Alice is Alice - it only proves that the initiator knows a hash of Alice's password, which could be stolen from Bob's database. So, the protocol continues. In message 4, Bob transmits the encrypted private key, encrypted again with K . The purpose of encrypting with K is so that an eavesdropper will not be able to see Alice's encrypted private key, which could be used for password-guessing attacks. Only one who actually knows Alice's password can eventually decrypt Alice's private key, so the fact that Alice signs R with her private key in message 5 proves to Bob that it is really her. Bob verifies her signature on R using her public key (which can be stored in Bob's database).

Alice knows it is Bob because he sends her $E\{\text{private key}_{Alice} \mid \text{password}_{Alice}\}$ in message 4. However, how does Alice know that she has actually extracted her private key? To be careful in this situation, Bellovin-Merritt propose that the *private key*_{Alice} be a concatenation of her identification and the actual key.

This protocol illustrates one of the principles of protocol design in which one needs to prove one's authenticity and knowledge of a low-quality secret (user password) before being given the high-quality secret (private key).

4. Design Issues

The IAuth system is designed, based on the Bellovin-Merritt protocol. It incorporates the concept of a *Key Distribution Center* (KDC) from Kerberos, and also applies the principle of a *server-to-KDC* handshake of KryptoKnight. In this section, we justify the basis of our design on these protocols and concepts.

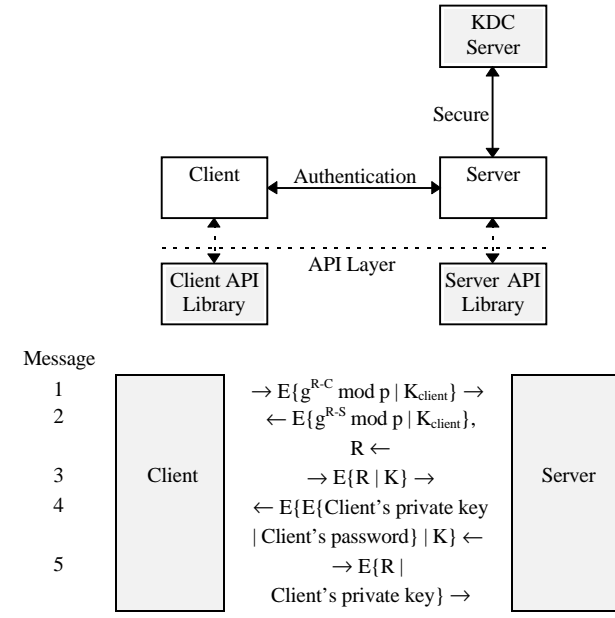
Web-based applications are characterized by the following features:

1. A Web-based application typically models the client-server architecture of network computing. There is a client process requesting a service from a server process.
2. A client obtains the desired service on a Web browser through the server's URL which is public.
3. There is no way for a client to ensure that the server is genuine, and for the server to ensure that a client is authentic and allowed to use its services.
4. Any provision of data encryption and digital signing schemes should not burden the user by requiring the user to possess secret components (keys), other than a user password.
5. The transactions of such applications are usually small and should be simple to use.

These features determine the basic requirements of our system.

If Web-based applications model that of a client-server application, how do we achieve authentication? Surely, there must be another trusted entity involved in the authentication process. The Kerberos concept of a trusted third-party, the *Authentication Server* or *Key Distribution Center* (AS/KDC), applies to our system, but in a different context. The use of an authentication server from which client applications obtain authentication services does not make sense since clients initiate requests directly to a server application on the Web server through the server's URL. But the concept of a KDC, which serves as a trusted third-party for secure storage and distribution of encryption keys is desirable, especially in the context of key management. A KDC as a back-end server can serve multiple Web application servers requiring authentication.

The principle of a *server-to-KDC* handshake of KryptoKnight applies to our system, since clients should not be able to request encryption keys directly from the KDC over the WAN. The Web application server obtains encryption keys from the KDC, which it then securely transmits to the clients. The communication channel between the Web server and the KDC can be protected from the Internet using firewalls and filtering routers.



where Server obtains $K_{Client} = hash(\text{Client's password})$ and $E\{\text{Client's private key} \mid \text{Client's password}\}$ from the KDC.

Figure 4. IAAuth Components and Message Flow

Since a Web application server is open and there is no way to authenticate both clients and server, we need an authentication service that is effective and simple to use. There must be no additional requirement for a user to possess secret components (keys), other than a user password. If data encryption or digital signing is to be provided, it must be easy and secure to obtain the cryptographic key required. The Bellovin-Merritt protocol satisfies these requirements. With this protocol, a user can securely obtain his private key or secret encryption key from the Web application server while being authenticated in the same process. The Bellovin-Merritt protocol has also ruled out most types of attacks through the use of an encrypted Diffie-Hellman key exchange and a challenge-response scheme. To prevent stealing, it ensures that keys and passwords are not stored plain. Also, Bellovin-Merritt is an authentication and key exchange protocol; it does not impose the choice of cipher algorithms to use with the keys exchanged.

5. IAAuth System Architecture and Design

The IAAuth system comprises of a KDC server and API libraries for the client application and the server application. The KDC server is a back-end daemon process which has a secure database that stores two items for each user - a hash of the user password, $K_{Client} = hash(\text{Client's password})$, and the user's private key encrypted with his password, $E\{\text{Client's private key} \mid \text{Client's password}\}$. Server applications obtain hashed passwords and encrypted keys from the KDC process to authenticate a user on the client application, and to provide the user with his private key. To do this, both the client and server application modules call authentication routines provided in the API libraries. The authentication routines run message flows 1 to 5, modeling the Bellovin-Merritt protocol.

Figure 5 illustrates the entities involved in a Web application service, and an example of how the KDC server may interact with a Web application server. We assume the presence of defensive mechanisms such as firewalls and filtering routers, and the protection of the KDC server from other security threats, such as stealing or leaking.

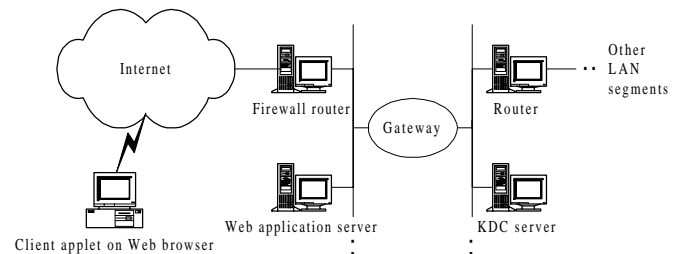


Figure 5. Components of a Web Application Service

6. Implementation

The IAAuth API libraries are implemented in Java (JDK1.1.1). The JDK1.1.1 packages *java.security*, *java.security.interfaces*, and *sun.security.provider* are used for implementing the cipher and mathematical functions required.

To demonstrate the API libraries, a client Java applet and a server Java application are implemented, in which the server requires a user to be authentic before accepting a signed message from the user through the applet. The application works this way. The server authenticates a user and provides him with his private key so that the user can proceed to sign a message for the server. The server then verifies the signature on the message before accepting it. In this application, authenticity, message integrity and non-repudiation is ensured.

To scale down the context of implementation, the client applet, the server application and the KDC process are run on a single machine. The client applet connects to

the server application which is implemented as a daemon process. The server daemon process, in turn, interacts with the KDC daemon process.

7. Conclusion

We have designed an effective and simple authentication system for secure Web-based applications, which performs authentication and provides support for both confidentiality and non-repudiation. We have also implemented a prototype of the system in Java, and tested it with a simple client-server application.

The IAuth system is suitable for all types of distributed applications, especially Web-based applications, since it is password-guessing secure and hence protected from Web spoofing. Also, the strength of the IAuth system in the context of the Internet, is that there is no need for a user to possess a cryptographic key if the application requires data encryption or digital signing. The user only needs to supply a password through a client applet on his Web browser in order to authenticate himself with the desired service on the Web server. Once verified authentic, the user can securely obtain his cryptographic key (private key or secret encryption key) from the Web application server for data encryption or digital signing.

We now summarize the important features of IAuth. The IAuth protocol implements pre-authentication, which is a good protocol design concept. In principle, an entity should never prove its identity to another until the other one does. Since this will not work, we assume that the entity that initiates connection is more likely to be the bad guy. Hence, a user that initiates communication should be the first to prove his identity by demonstrating that he knows a password. In IAuth, this is the hashed password since we do not want intruders to listen for and collect responses to authentication requests.

The IAuth protocol also illustrates another good protocol design concept which is that someone needs to prove that they know a low-quality secret (user password) before being given the high-quality secret (private key).

There is no need for a user to possess cryptographic keys for data encryption and digital signing. The user only needs to know his password in order to securely obtain his private key or secret encryption key for data encryption or digital signing.

An essential requirement of the IAuth system is the protection of the KDC server from other security threats, such as stealing or leaking. Although the KDC database only stores hashed passwords and encrypted private keys, these information needs to be protected because intruders can still mount known-plaintext

attacks if the plain equivalent of a hashed password or an encrypted key is somehow intercepted or known.

Some possible future work on the IAuth system:

1. Multiple and/or replicated KDC servers, to manage authentication load or key distribution across WAN.
2. To improve performance, cipher functions can be implemented using native code instead of Java bytecode which needs to be interpreted. This represents a tradeoff between performance and platform-transparency.
3. To improve performance, we can use a shorter random key to encrypt the Diffie-Hellman exchange and then use the hashed password to encrypt the random key. This may generate time savings especially if the hashed password is long.

Given that we have taken possible security threats into account in the protocol, the fact that no usable system is impenetrable still holds true. Therefore, the objective of any security strategy is to strike a balance between the costs and benefits of the security mechanism and the level of protection required.

We hope that the IAuth system provides a good example of a practical security mechanism for Web-based applications based on the research efforts of the designers of the authentication protocols - Bellovin-Merritt, KryptoKnight and Kerberos.

Bibliography

1. C. Neuman and T. Ts'o. "Kerberos: An Authentication Service for Computer Networks", *IEEE Communications*, 32(9), 33-38 (September 1994).
2. J. T. Kohl, B. C. Neuman, and T. Ts'o. "The Evolution of Kerberos Authentication System In Distributed Open Systems", 78-94. IEEE Computer Society Press (1994).
3. S.M. Bellovin and M. Merritt. "Limitations of the Kerberos Authentication System" *Computer Communication Review*, 20(5), 119-132, Oct 1990.
4. B. Lampson, M. Abadi, M. Burrows, E. Wobber. "Authentication in Distributed Systems: Theory and Practice", *ACM Transactions on Computer Systems*, 10(4), 265-310, (Nov 1992).
5. Bellovin S. M. and Merritt M. "Augmented Encrypted Key Exchange", *Proc. of the First ACM Conf. on Computer and Communications Security*, 244-250 (Nov 1993).
6. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M.Yung. "The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution", *IEEE Transactions on Networking*, Vol 3, No. 1, pp. 31-42, February 1995.
7. PKCS#3: Diffie-Hellman Key-Agreement Standard. *An RSA Lab. Tech. Note*. Ver. 1.4.