Protocol for Secure Submissions into Learning Management Systems

Manjunath Mattam

MSIT Division, International Institute of Information Technology,
Gachibowli, Hyderabad 500 032, Andhra pradesh, India
manjunath.m@iiit.ac.in

Abstract. This paper proposes a model (architecture and protocol) that will help in securing assignment submissions into learning management systems. A client server architecture that uses cryptography is proposed to transform a regular assignment deliverable into a secure deliverable. A protocol is devised between client and server such that faculty and students can securely obtain symmetric keys to either encrypt or decrypt an assignment deliverable.

Keywords: application specific protocols, learning management systems, secure protocol, symmetric key exchange mechanism, client server architecture, plagiarism check, confidentiality in transit, confidentiality in storage, integrity of deliverables.

1 Introduction

Information and communication technologies have transformed the way we teach and way we learn. These technologies opened avenues for remote learning, digital learning (e-learning) and automation of processes in conventional education systems. In this context, deliverables (worked out assignment answers that are submitted after completion) play major role in student assessments. For managing the deliverables and course content most universities use learning management systems (example Moodle, other university specific applications). Learning management systems (LMS) provide users with options to upload the deliverables. Majority of the learning management systems operate on HTTP (web based - plain text).

Plagiarism is one of the primary concerns in student assessment, although most universities operate tools to detect copied deliverables majority of these cases go undetected. My assumption here is reputed universities give assignments that are designed specifically for enrolled students and therefore solutions are not available on the internet. Copied deliverables can be classified based on whether student deliberately shared his work output, or the deliverable is copied without authors notice.

Consider two given scenarios: (1) A student shares his worked out deliverable after uploading it in LMS, by giving away his login details. Ultimately claiming my account is hacked, if he gets caught. (2) Other students can use network traffic sniffers (like wireshark, libcap) to read ongoing traffic, and pick up deliverables without authors

S.M. Thampi et al. (Eds.): SNDS 2012, CCIS 335, pp. 45–53, 2012.

consent. Because most LMS are not HTTPS enabled (reasons being certificates are expensive, and performance intense). It is possible to pick up deliverables that are transferred in regular HTTP traffic (plain text). In scenario 2 students gets punishment without his/her fault, if he/she gets caught.

This paper proposes an approach, which minimizes problems created by above mentioned two scenarios. Over all goals are, if an assignment deliverable is copied then it is due to willful sharing of students (not on transit or at server), making students accountable. A client server architectural approach & application specific protocol is proposed that will ensure secure deliverable submissions (capable of confidentiality, integrity, authentication, and non repudiation of each deliverable).

2 Crypto Primitives

This application uses cryptographic primitives to achieve secure deliverable submissions. Each (secure) deliverable submitted by a student is encrypted with industry adopted symmetric key crypto algorithms that have received substantial public review and have been proven to work effectively like AES with substantial key size (256 bits). When encrypted deliverable is submitted to LMS over open network, it is hard for packet sniffers to retrieve plain text (high level crypto with large key size makes it almost impossible).

Managing the symmetric key is the challenging part of these kinds of applications. Alternative approaches with asymmetric key cryptography are available using public key infrastructure (proven with other electronic commerce applications), which is not recommended in this scenario because (1) it involves issuing certificates to all the users including students (2) public key algorithms need funding and they are expensive (3) computationally performance intense, and consume lot of time.

To open the decrypt deliverables faculty will need same symmetric key used by students for encryption. Transferring the symmetric key unprotected is almost equivalent to no encryption. Stand alone client server program is proposed for secure key transfer and key management. Server acts as trusted third party for all the users (in this case students and faculty). Server generates, issues, stores, and manages symmetric keys. All communication between server and client software is protected by protocol designed specifically for this kind of applications.

The proposed model does not involve asymmetric key cryptography hence the computation required for encrypting or decrypting a deliverable is relatively lower. Client program is not responsible for creation of symmetric keys hence at user computer this computation is saved.

3 Client Server Program

Client server programming model is most appropriate for managing keys; socket programming is best suited for sending/receiving messages between client and server. All communication is encapsulated in application layer, rest of the internet layers are

constructed by network sockets. Server waits for client's request on a designated port number. Server program is capable of understanding application specific pre-defined codes that corresponds to type of client requests (for example: request for creation of symmetric key, request for using existing symmetric key, change of user password etc. all are pre-defined constants). Server is also capable of performing role based operations, (for example: if client software is logged-in by student it generates symmetric key and if client software is logged-in by faculty it retrieves already generated symmetric key).

Given below is an image that illustrates (at high level) how client and server program is used for this process. Server interface in client program does the socket communication. Client interface in client program is responsible for encrypting and decrypting the deliverables. Server program is responsible for key management.

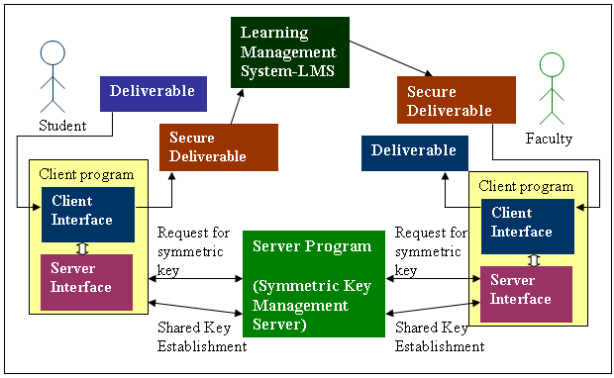

Fig. 1. Model architecture for secure assignments submission. Assignment submission in LMS is as usual. Client server program will help convert regular derliverable into secure deliverable.

3.1 The Server Program

Server is programmed to store, search, retrieve, respond based on user request type. Server is programmed to concurrently handle multiple client programs connected at one point in time. Server stores application details with-in, this storage can be made either in database or inside regular text files (additional security measures are required for data that is stored inside flat files implementation). Users of this application are registered with the server therefore login details like user id, password, role in organization, login status etc. are stored and maintained by the server. Passwords used for authenticating a user must adhere with server specified size, usage of alpha numerals etc.

All communications between clients and server are encrypted, symmetric key that is required to encrypt or decrypt communication is called shared key between client and server. Server stores all the shared keys along with time stamp, corresponding user identification in its data store, as represented in table 1 below. Server will need to

Table 1. Example data structure at the server to store user shared keys

User Identifier	Time Stamp	Shared key between client and server

define time threshold, indicating how long a shared key is valid. If this time threshold expires, client will need to request for a new shared key.

Apart from above mentioned details, server will also store all the symmetric keys required to encrypt assignment deliverables. These symmetric keys are generated by the server upon request from user with a student role on client program. These symmetric keys are used by client programs to encrypt a deliverable. Because there is no time limitation for how long a deliverable is maintained in LMS, server may have to stored these keys for longer period of time like till the end of academic year. For faster retrieval of these keys, each of these symmetric keys is uniquely identified with an ID number. Server maintains key details corresponding to each user, that is User id, message digest of the file, IP/MAC address from which deliverable is uploaded, unique id given for the key, the symmetric key required for the deliverable. Fixed size symmetric keys are generated using simple programming logic for faster execution (for example: Message Digest of (Random Number || User ID || File message digest || Unique file identifier || Time stamp)). Higher the key sizes better the security measures.

Table 2. Example data structure at the server to store symmetric keys to encrypt or decrypt deliverables

Symmetric key unique identifier	Symmetric key	User identifier	Message Digest	MAC address

If the unique identifier for symmetric key are sequential in nature then data is stored in sorted order, hence it is faster to retrieve specific symmetric key using key identifier with search algorithms. If all the values are stored inside data base management systems, then symmetric keys can be retrieved even faster as values are indexed by DBMS software that can scale up to billions of deliverable symmetric keys.

Connection time required for client server communication is low, as maximum number of message exchanges is 4. This way server wait queue is processed quickly. With help of socket programming concurrency techniques like threads server can scale up to thousands of users or more.

3.2 The Client Program

Client program must be installed by all the users (faculty and students). Client program always initiates the connection, this request starts with a user log-in. Client program provides interface for login with username and password. Client software is configured to accept a strong password that includes alpha numeral characters and special symbols with minimum number of letters for password. Client programs can also be configured to bring users to assigned/specific work stations, such that a student can upload deliverables from his/her designated computer only (however this functionality is optional). Client program can send workstation specific information like IP address or MAC address to know location from where user is uploading the deliverable. Client program provides interface to browse and select a file from the resident computer. Once a file is selected client program is capable of encrypting or decrypting that file using symmetric key assigned by server. Based on the user role like (1) students will use client software for encrypting deliverables. (2) Faculty will use client software for decrypting deliverables.

For a student after selecting the deliverable that is choosing the file from local computer he will need to use encryption options, output of the program is saved in same directory with server sent symmetric key unique identifier as file extension (for example: rollno2_program3_dotc.2342313, here 2342313 is symmetric key identifier). This is the output file, and it can be uploaded in learning management systems (LMS), as deliverable. Uploading into the LMS is out of scope for the proposed system, this is the usual procedure a student follows to submit their assignments / deliverables.

Faculty can download student's deliverable from LMS with the usual procedure. Faculty also uses client program because he needs to evaluate the encrypted deliverable. After login a faculty can select downloaded deliverable using client program browse interface. To obtain the plain text (un-encrypted) original file (in our previous example: rollno2_program3_dotc) faculty end client program must send symmetric key identifier to the server. If symmetric key identifier is not present on the deliverable file name, offline communication is needed to obtain the key ID from the student. Client program at faculty must recognize key ID from the deliverable or provide interface for the user to input. Faculty end client program receives symmetric key to decrypt secure deliverable from the server that is symmetric key corresponding to unique key identifier of the deliverable.

Client software must be capable of calculating message digests or hash functions before and after cryptographic process. Deliverable integrity checks are done by comparing (string compare) fixed length hash function output strings.

4 The Protocol

Client and server exchange messages based on rules and formats mentioned below. Application specific protocol messages are explained here using formal methods as specified in reference [1]. First step in this protocol is shared key establishment; in this phase client and server arrive upon a shared key which is used as encryption key for all subsequent communication.

4.1 Shared Key Establishment

Given here is the formal method, please note enclosed in curly brackets { } means encrypted, with a key that is mentioned outside brackets, T means time stamp, N represents nonce, MAC represents media access control address used for identifying a network interface, U represents username.

C --> S: U, $\{U, \text{Mac Address}_C, Nc, T\}_{Password}$
S --> C: $\{U, \text{Mac Address}_S, Nc, Ns, Kcs, T\}_{Password}$
C --> S: $\{U, Ns, Nc^1, T\}_{Kcs}$
S --> C: $\{Nc^1\}_{Kcs}$

Client initiates the connection by sending username (in plain text), encrypted authentication information, the user id, client MAC address, a random number (nonce), and time stamp, all this information is encrypted with user password. Here password is used as encryption key. Server decrypts received information with password corresponding to username. This is an attempt to authenticate user without actually having to transfer password (on wire) in plain text. Decryption can only happen when user password stored in server is equal to password client program used for encryption, if server can decrypt information it only mean username password are correct. If user password stored in server data store does not match with password used as key, generated output will have all garbage values. Server can respond authentication failure and close the connection.

Once authenticated, server responds to client with Username, server MAC address, nonce sent by client, server random number (nonce), shared key, server time stamp all this information is again encrypted by user password. Client software can decrypt this communication with user password. Nonce is a random number used for freshness of message and for verification. By observing these nonce, client and server can be sure the messages are corresponding to current session, not the once stored from previous client and server communications (this ensures protection against replay attacks). After this step client has the shared key sent by the server therefore all further client server messages are encrypted with the shared key.

Next two messages are used for confirmation, which is agreeing upon the shared key. Client sends to the server username, server nonce, a new client nonce, time stamp all this information encrypted with the shared key. This is confirmation that client has agreed to use server given shared key. Server sends back the new client nonce (encrypted with shared key) indicating successful establishment of shared key.

If client or server nonce does not match then shared key establishment is failure or incomplete. Similarly if client program does not demonstrate the knowledge of share key then server does not engage / communicate with the client. Server also returns pre defined code for in correct shared key usage.

4.2 Symmetric Key Management

Learning management systems (LMS) usually provide students option for uploading single or multiple files. Where ever necessary, students will have to zip (compress or

combine) all files before uploading into one submit-able file, the deliverable. These deliverables are encrypted with symmetric key algorithms. In cryptography, symmetric key means same (one) key for both encryption and decryption process.

4.2.1 At Student, the Symmetric Key Is Requested for Encrypting

Client program requests server for symmetric key and server program responds with unique identifier and the symmetric key. Given here is a formal method for clients requesting symmetric key to encrypt a deliverable:

Message 1 C --> S: U, $\{CODE, U, H(X), T\}_{Kcs}$
Message 2 S --> C :$\{ Id, K, \{Ns\}_{K, T}\}_{Kcs}$
Message 3 C --> S :$\{ Ns\}_{Kcs}$

Here U represents user id, CODE is pre-defined number indicating client is requesting for symmetric key (for example: 511525), H(X) represents hash function output of given deliverable, T represents timestamp, K represents symmetric key, N represents nonce, and Kcs is shared key between client and server.

In message 1, Client communicates with the server by sending user name in plain text, and remaining sent information is encrypted with the shared key, that is request code for symmetric key, user name, message digest of deliverable, time stamp. Server can decrypt this client communication with user's corresponding shared key. In message 2, server generates the fixed length symmetric key, unique symmetric key identifier, a nonce (encrypted with symmetric key for verification), and time stamp. All this information is encrypted with shared key before sending to client. In message 3, client obtains the symmetric key, stores key identifier and decrypts server nonce (that was previously encrypted and sent in message 2). This decrypted nonce is sent back to server, which is now verified (if Ns match - client has got the right symmetric key).

4.2.2 At Faculty, the Symmetric Key Is Requested for Decrypting

Server is programmed to retrieve symmetric key only if key identifier is provided. Server responds to key requests based on user name, and its role. Request for decryption key (with key identifier) is possible only for owner of the deliverable and users with faculty role. Given here is a formal method for clients requesting symmetric key to decrypt a deliverable:

Message 1 C --> S: U, $\{CODE, U, Id, T\}_{Kcs}$
Message 2 S --> C: $\{Id, K, \{Ns\}_{K}, T\}_{Kcs}$
Message 3 C --> S: $\{Ns\}_{Kcs}$

Here U represents user id, CODE is pre-defined number indicating client is requesting for symmetric key (for example: 411525), Id is the identifier of symmetric key, T represents timestamp, K represents symmetric key, N represents nonce, and Kcs is shared key between client and server.

In message 1, client (at faculty end) requests to server by sending user name in plain text, and remaining sent information is encrypted with shared key, that is request code for symmetric key, user name, unique id of the symmetric key, time stamp.

Server can decrypt this client communication with corresponding shared key. In message 2, server retrieves symmetric key corresponding to unique key identifier. Server sends to client identifier, key, a nonce (encrypted with this symmetric key for verification), and time stamp. All this information is encrypted with shared key before sending to client. In message 3, client obtains the symmetric key, stores key identifier and decrypts server nonce (that was previously encrypted and sent in message 2). This decrypted nonce is sent back to server, which is now verified (if Ns match - client has got the right symmetric key).

Faculty can now use this symmetric key to decrypt student's deliverables.

5 The Integrity Check

Message integrity checks are required to verify completeness and consistency. Many hash functions like message digest are used to generate message integrity codes. Client software will verify if the intended communication is complete. It is important for the faculty to know, that a deliverable is not altered during communication or storage.

Client software applies hash function before a deliverable goes through encryption process. A deliverable, and its message digest are part of the plain text. After decryption, the deliverables and message digest are saved in the faculty computer. Client software verifies file message digest with decrypted message digest. If they are exactly same this means deliverable is not altered in transit or storage.

6 Other Applications

Security modules in existing learning management systems are either SSL/TLS based or LMS specific proprietary security (lock-in) mechanisms. With SSL / TLS based encryptions all educational institutions need digital certificates for assignments purpose (and will need effective certificates management) which is expensive as specified in section 2 (crypto primitives). Learning management system specific security solutions work only in that LMS environment and may be difficult to migrate (to other LMS) later on.

Architecture discussed in this paper will work in assignment submissions scenario and can be applied to all group (1 to many) based circulation of confidential information like securing digital notice boards, securing office documents where one party acts at encryption end while other party obtains key using role based authentication at decryption end. For the submissions that do not require secure transfer, proposed model can be bypassed and users can continue with the usual / regular mode of submissions.

7 Conclusion

Secure assignment (deliverables) submission architecture & protocol will help in achieving confidentiality of student deliverables on wire & LMS storage through

cryptography, authentication is achieved through passwords and machine specific information like MAC address, integrity of the deliverable is achieved using hash functions and non repudiation is achieved using shared keys & client software. This process will rule out chances of copying deliverables on wire and from learning management systems.

Long passwords must be used to avoid dictionary attacks before establishing shared key. Hashing is used by student client program while uploading the deliverable this is used in avoiding corrupted files occasionally created by network transfers. This is because network protocols verify packet level integrity not at file/deliverable level.

Existing learning management systems need not be changed or modified to incorporate this solution. This solution can also avoid writing numerous plug-ins for different kinds of Learning Management Systems, secure transfers. Simple applications like these can enhance universities deterrence toward plagiarism check.

References

1. Abadi, M., Needham, R.: Prudent Engineering Practice for Cryptographic Protocols. In: IEEE Computer Society Symposium on Research in Security and Privacy (1994)
2. Anderson, R., Needham, R.: Programming Satan's Computer. Cambridge University Computer Laboratory
3. Bellovin, S.M., Merrit, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: 1992 IEEE Computer Society Symposium on Research in Security and Privacy (1992)
4. Richard Stevens, W.: UNIX Network Programming: Networking APIs: Sockets and XTI, 2nd edn., vol. 1. Prentice Hall (1998) ISBN 0-13-490012-X
5. Bellare, M., Canetti, R., Krawczyk, H.: A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In: Proc. of the 30th STOC. ACM Press, New York (1998)
6. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
7. Morris, R., Thompson, K.: Password security: a case history. Communications of the ACM 22, 594–597 (1979)
8. Glass, E.: The NTLM authentication protocol (2003)