make exercise from this. don't change the exercise, this is text from many snapshots of exercises. But i want the full exercises. fill in the gaps

21:03 | 1,6KB/s

✔ucilnica.fri.uni-lj.si/mod/assi 69

=

79

\*

\*

\*\*

Computes the MAC tag over the message.

\* @param payload the message

\*

@param password the password form which M \* @param salt the salt used to strength \* @return the computed tag

public static byte[] mac(byte[] payload, Str return null;

}

/\*\*

\* Verifies the MAC tag.

@param payload the message

\*

\*

\* @param tag

the MAC tag

\*

\*

@param password the password form which M

\* @param salt

the salt used to strength

\* @return true iff. the verification succee

public static boolean verify(byte[] payload, return false;

}

[10 points] Implement the token-based

access-control in Lock

Alice tries to open the lock by sending a token. Lock verifies the received token and if the verification

succeeds, the Lock prints SUCCESS and otherwise it prints FAILURE.

Stanje oddaje prispevka

↑

21:02 | 4,7KB/s

ucilnica.fri.uni-lj.si/mod/assi 69

=

79

Lock should receive the token from Server, and verify it. If the verification succeeds, the token is stored to a local variable.

Implement the following auxiliary functions and use them in the protocol.

\*\*

\* Computes the MAC tag over the message.

\*

\* @param payload the message

\*

@param password the password form which N \* @param salt the salt used to strength \* @return the computed tag

public static byte[] mac(byte[] payload, Str

return null;

}

/\*\*

\* Verifies the MAC tag.

@param payload the message

\*

\*

\* @param tag

the MAC tag

\*

\* @param password the password form which N

\*

@param salt

the salt used to strength

\* @return true iff. the verification succee

public static boolean verify(byte[] payload, return false;

}

[10 points] Implement the token-based access-control in Lock

↑

21:02 | 5,6KB/s\*\*\*

W

Q

79

[15 points] Transfer the token from Server

to Lock

Then as the Server, forward that token to the Lock. When forwarding the token, the Server must compute the MAC tag and send it alongside the token, otherwise the Lock will reject it. The tag is computed with HMAC- SHA256, while the key is derived from a shared password that is known to Server and the Lock; you may define that password globally in the main method. As the password-based key-derivation function use PBKDF2 with HMAC-SHA256, and set the iteration counter to 1000. Thus when forwarding the token, the Server must also send the tag and the salt that is used to verify the tag.

Lock should receive the token from Server, and verify it. If the verification succeeds, the token is stored to a local variable.

Implement the following auxiliary functions and use them in the protocol.

/\*\*

\*

\* Computes the MAC tag over the message.

\*

\*

@param payload the message

\*

\*

@param password the password form which N @param salt

the salt used to strength

\* @return the computed tag

public static byte[] mac(byte[] payload, Str return null;

}

/\*\*

\* Verifies the MAC tag

↑

21:02 | 7,5KB/s\*\*\*

W

Q

79

[20 points] Provide symmetric

confidentiality and integrity and produce a token

Use symmetric cryptographic primitives negotiated above to provide confidentiality and integrity to the communication channel Alice-Server. Then as Alice, implement the procedure that generates 1000 tokens and upload the last token in the chain to the security Server over the secured channel.

Implement the following auxiliary function and use it in the protocol.

/\*\*

\* Hashes the given payload multiple times.

\*

\*

@param times the number of times the va \* @param payload the initial value to be ha \* @return the final hash value

\*/

public static byte[] hash(int times, byte[] return null;

}

[15 points] Transfer the token from Server to Lock

Then as the Server, forward that token to the Lock. When forwarding the token, the Server must compute the MAC tag and send it alongside the token, otherwise the Lock will reject it. The tag is computed with HMAC- SHA256, while the key is derived from a shared password that is known to Server and the Lock; you may define that password globally in the main method. As the password-based key-derivation function use ↑

DRKDE? with HMAC SHA256 and set the iteration cou

21:02 | 0,0KB/s\*\*\*

79

The programming assignment

We have three agents: a user Alice, a security Server and an electronic Lock; and three communication channels: Alice-Server, Server-Lock and Alice-Lock. Your assignments are the following.

[15+ 5 points] Mutually authenticate channel Alice-Server and generate a shared secret

[15 points] Authenticate the channel between Alice and Server. They both use RSA public-private key pairs and their public keys are known to both of them: Alice knows Servers' and Server knows Alices'. Your task is to mutually authenticate the peers and generate a shared secret: Alice needs to be sure she is talking to the Server and vice-versa. Define the public-secret key pairs globally in the main method.

[5 additional points] Implement the key-agreement and key negotiation in a manner that is forward-secure: so even if the adversary was to record all message exchanges between Alice and the Server and then later would manage to steal their key-pairs (secret keys in particular), he still would be unable to decrypt the recorded messages.

You may decide to skip this step and assume that Alice and Server already have a shared symmetric secret. But in this case, you will not be awarded points for this assignment.

[20 points] Provide symmetric confidentiality and integrity and produce

Token

21:02 | 4,1KB/s -

=

Q

79

Token verification

1. When trying to open the lock, the user places her phone next to the lock and a token is transmitted. If we generated 1000 tokens, we would transmit SHA256999 (s) when opening the

token t

=

lock for the first time.

2. The electronic lock would then apply another

round of SHA256 to the received token t and then compare the result to the stored value. In the case of first access, the stored value would be the token the user uploaded to the security Server: SHA2561000 (s).

3. If the computed value and the stored value

match, the access is granted and the currently

stored value is replaced with token t. Otherwise access is rejected.

The token verification is run every time a user attempts to open the electronic lock. Note that the user has to submit a different token for each access. Thus for the next access, the user would use a token that is obtained by hashing the initial secret 998 times. The user can repeat this until she runs out of tokens. At that time, she has to generate a new secret, compute new tokens and upload them to the security server.

1a) Generate tokens

1b) Send the token over a secure channel

(TCP/IP)

↑

21:02 | 13,6KB/s

79

=

Q

3. In the third step, as user tries to gain lock access by using her NFC-capable phone. When standing next to the door, she places her phone next to the lock and the phone transmits a token over the NFC link. The electronic lock then verifies the

token using the Lamport's procedure described below.

To further simplify the example, we assume the company uses only one electronic lock and there is only one user that is allowed to open it. (In general, one would have to generate different tokens for different locks and keep tokens between users separate, but you do not have to handle such cases.)

Token generation

1. User chooses a secret s.

2. She then repeatedly applies the hash function SHA256 to the secret s, giving a value of SHA256(SHA256(... SHA256(s)...)). For instance, we could run the hash function 1000 times.

3. The final value, denoted as SHA2561000 (s), is

then securely transmitted to the security server

and the latter then transmits it to the electronic lock. The lock stores this value.

Token verification

1. When trying to open the lock, the user places her phone next to the lock and a token is transmitted. If we generated 1000 tokens, we would transmit token t SHA256999 (s) when opening the

lock for the first time.

2. The electronic lock would then apply another

round of CII 1956 to.

taken to

21:02 | 7,1KB/s.

=

Q

79

Protocol overview

To prevent unauthorized access to remote areas, a control system is designed, where users unlock electronic locks with their NFC-capable phones. Since such locks have constrained processing capabilities (slow CPU, limited power and memory), they are unable to perform asymmetric cryptographic operations. To alleviate this, you'll implement a version of Leslie Lamport's one-time-password scheme that is based on one-way hash functions. We'll be working with SHA256. The entire procedure roughly consists of three steps.

1. In the first step, the user generates a chain of tokes (hashes) and securely uploads the last token in the chain to the company security server. In practice, such transmission would occur over a secured TCP/IP socket. The user would upload a new token when existing tokens would run out; typically once or twice a year. The details of token. generation are provided below.

2. In the second step, the security server transfers tokens to electronic locks. This transmission is performed via an out-of-band channel; imagine tokens being transmitted through a serial protocol by a technician that is physically next to the lock. However, in order to store the tokens, the technician must authenticate herself and this is achieved with the help of a password-based MAC scheme whose details are specified below. Typically, such procedure would be performed once per month.

3. In the third step, as user tries to gain lock access

by using her NFC-capable phone. When standing

21:02 | 84,8KB/s

✔ucilnica.fri.uni-lj.si/mod/assi 69

=

79

Information Security & Privacy: Midterm 1

General rules

Implement the following communication protocol with appropriate security primitives in Java using the Environment class. First carefully read the instructions and then implement the solution.

You are allowed to use all resources, including the Internet, but communication with anyone except supervising teaching assistants is forbidden.

1. Clone project https://github.com/lem- course/isp-handson.git and code your solution

in file Midterm.java.

2. You have 75 minutes to complete the midterm. 3. Afterwords submit a single file named

Midterm.java.

Protocol overview

To prevent unauthorized access to remote areas, a control system is designed, where users unlock electronic locks with their NFC-capable phones. Since such locks have constrained processing capabilities (slow CPU, limited power and memory), they are unable to perform asymmetric cryptographic operations. To alleviate this, you'll implement a version of Leslie Lamport's one-time-password scheme that is based on one-way hash functions. We'll be working with

SHA256. The entire procedure roughly consists of

three stens

21:03

=

|||

FRI

Učilnica FRI 21/22 Slovenščina (sl)

Server

1a) Generate tokens

Alice

1b) Send the token over

a secure channel

(TCP/IP)

3) Try to gain access using a token

(NEC)

0,1KB/sill 79

7

Q

ANŽE HABJAN

2) Securely deliver the token to the lock

(out-of-band)

↑