Team member: Shrinivass Av , Chao-Hsuan Shen

Implementation of Challenge response system and comparison with other protocols

Introduction

We have learned lots of ideas about how to securely exchange important, sensitive informations in the class. Our team wants to implement a modified challenge response system and compare it with some popular security protocols to quantify its effectiveness based on a simulation. The point is to get familiar with a range of protocols and gain insights about how to make trade-offs under different circumstances because there is no single best protocol, only the proper one in terms of the user context.

Language in use & why

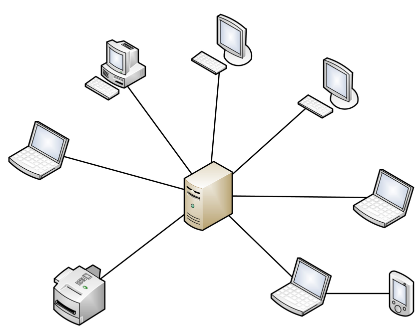
We choose to use Scala with Akka library to build our system due to the following reasons:

1. Scala[1] is Java compatible. We can use existing Scala and Java libraries on the market.
2. Akka library[2] greatly supports many models we need to simulatie in our project. Like:
    a. Client-Server
    b. P2P
3. Akka library takes care of low level details about message passing and support remote actors, which is closer to the modern user scenarios.

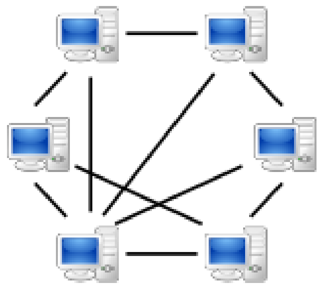
Architecture design

Two major part of architecture design

1. Organization of elements :
    ○ Client-Server Model →



[1] "The Scala Programming Language." 2007. 29 Nov. 2013 <http://www.scala-lang.org/>
[2] "Akka." 2010. 29 Nov. 2013 <http://akka.io/>
[3] <https://communities.intel.com/servlet/JiveServlet/showImage/38-14293-78999/client_net.jpg>



- P2P Model → [4]

2. Communication framework :
   - akka actor is role based message passing system, and it is hierarchical by default, which make it the best framework to use.

Cryptographic implementation

We mainly use two important en/decryption algorithms in the implementations of these protocols.

AES

AES.scala contains the basic implementation of AES algorithm which uses the crypto package of java[5].

- generatekey,genIntkey: These functions create a 128 bit key.
- encrypt: This function takes two arguments, a plaintext string and a key string and encrypts blocks of 128 text following PKCS5 padding. Then it returns an Array of Bytes.
- decrypt: This function takes the encrypted array of bytes, the key string and decrypts to derive the original message

RSA

RSA.scala uses the basic implementation of key generation which utilizes security and math[6] package.

- RSA(): It selects the number n, p,q, $\phi(n)$,all required to implement RSA. The public and private keys are calculated using these variables and function.
- encryptRSA : It takes three arguments, message, key and the modulus to encrypt the given message.
- decryptRSA: It takes three arguments, encrypted message, key and the modulus to decrypt the given message.

We have implemented 5 different types of protocols.Each protocol has its own pros and cons to serve for the security purposes in an insecure network.

---

[4] <http://upload.wikimedia.org/wikipedia/commons/thumb/3/3f/P2P-network.svg/200px-P2P-network.svg.png>

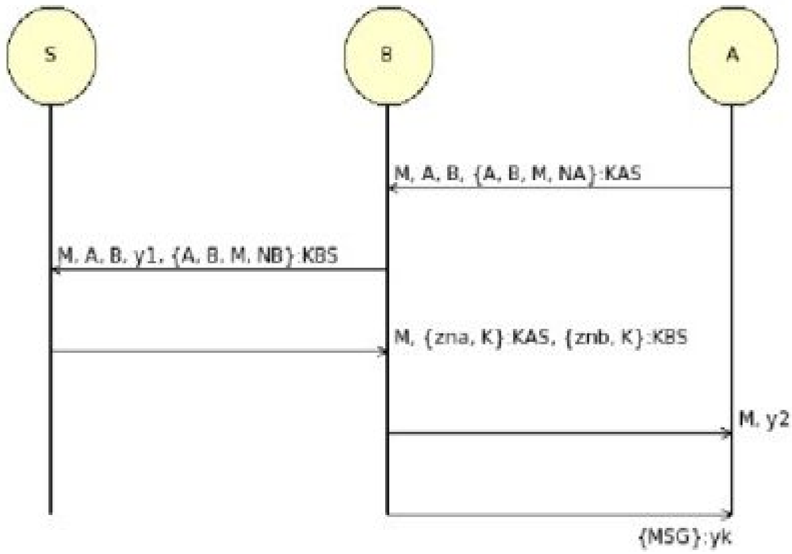
[5] "Encrypting and decrypting a string with AES in java - Stack Overflow." 2013. 29 Nov. 2013 <http://stackoverflow.com/questions/17322002/encrypting-and-decrypting-a-string-with-aes-in-java>

[6] "Rsa implementation using java - Stack Overflow." 2013. 29 Nov. 2013 <http://stackoverflow.com/questions/15920739/rsa-implementation-using-java>

Team member: Shrinivass Av , Chao-Hsuan Shen

1. Otway Rees protocol[7]

- ## Basic Idea[8]:



A and B try to establish a connection via third-party Key Distribution Center.

- In this protocol A sends a message to B. S represents the Key Distribution Center to share the session key. The message contains a session identifier M and a nonce NA to verify whether the message is fresh and is not subjected to replay attacks. The message also contains an encrypted version of A, B, M, NA which is enciphered using a pre-shared key KAS which is shared between A and S by using the AES encryption function.
- B gets the message and adds its own version of A, B, M, NB encrypted with KBS and sends it to S. In the above figure y1 represents the enciphered version A, B, M, NA using KAS.
- S verifies the veracity of A and B by decrypting the message. It sends a message back to B which contains two parts NA , session key K encrypted with KAS and NB, session key encrypted with KBS
- B decrypts the part which is encrypted with KBS and sends the rest of the message to A.
- A decrypts the message key. Thus at the end of the protocol A and B share a session key.

- ## Implementation Detail

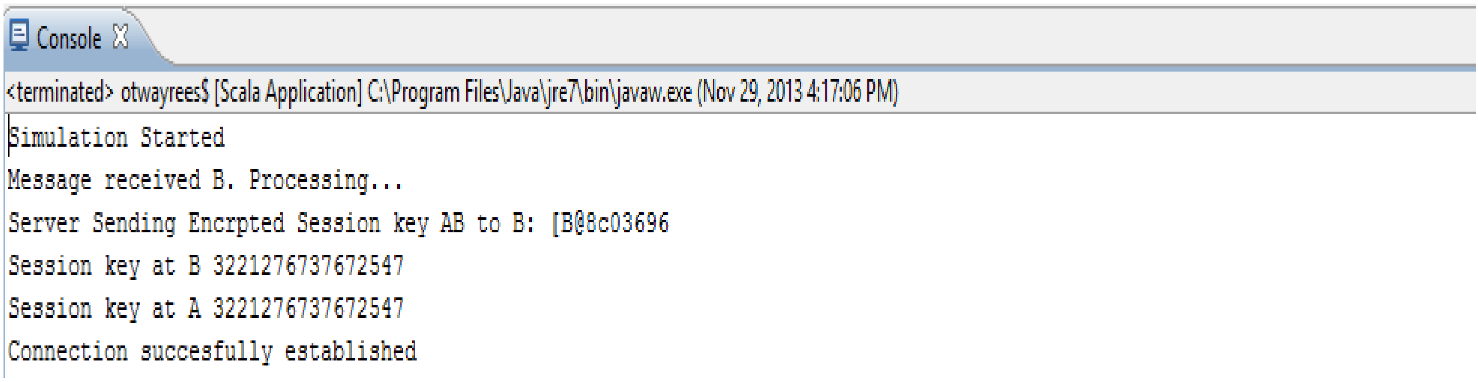
- Start_SimulationOta() and EstCon() implement the 1st step of the protocol. The message is encrypted using the KAS as the key in AES.encrypt() function.
- Once the message is received by B it further appends the message with its own encrypted message with key KBS and sends it to KDC.

---

[7] "Otway–Rees protocol - Wikipedia, the free encyclopedia." 2011. 29 Nov. 2013 <http://en.wikipedia.org/wiki/Otway%E2%80%93Rees_protocol>
[8] "Lecture 62: The Otway-Rees Protocol - Department of Computer ..." 2011. 29 Nov. 2013 <http://www.cs.utexas.edu/~byoung/cs361/lecture62.pdf>

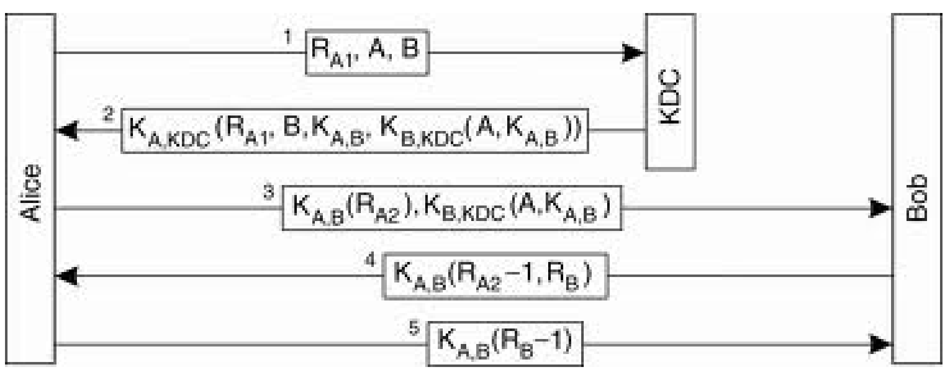
- KDC decrypts the message using the keys KAS, KBS and AES.decrypt(). After verifying the sender and receiver it sends a message back to B.
- B decrypts its message in the function CheckedOta() and obtains the session key. It sends the rest of the message to A.
- A decrypts the to verify its Nonce and then assigns the session key from the decrypted message.

Output:



2. Needham-Schroeder protocol[9]

Basic Idea



The Needham Schroeder protocol also uses a 3rd party system to establish communication link between A and B. The Needham Schroeder protocol we implemented is not the classical version but the improved version to counter replay attacks.

- Initially A sends a message to the Key Distribution Center(KDC) with a nonce R_{a1}
- KDC verifies the sender and the target sends the session key encrypted with both K_{akdc} and K_{bkdc} to send the session key
- A then decrypts the message to obtain the session key verifies its nonce and sends the other part of the encrypted message to B.

[9] "Needham–Schroeder protocol - Wikipedia, the free encyclopedia." 2011. 29 Nov. 2013 <http://en.wikipedia.org/wiki/Needham%E2%80%93Schroeder_protocol>

- B decrypts the message and sends A the nonce calculation to A along with its own value of nonce R_b to avoid replay attacks.
- A decrypts and then encrypts B's nonce calculation with session key to prove its claim and sends it back to B thus establishing connection.

Implementation

i. Start_Simulationneed() starts the process with A (represented in the program as Aneed) sending a message to KDC to obtain the session key.

ii. KDC (represented as KDCneed) encrypts the session key with KAS and KBS(preshared AES keys) and sends it back to A.

iii. A then sends the message to B through function ConSend() with a new nonc encrypted with the session-key.

iv. B decrypts the message sent by A which contains the session key encrypted by KBS aand obtains it. It then decrypts the nonce of A subtracts and 1 and sends it to A along with its own nonce.

v. A verifies whether the nonce value and decrypts the message and sends B's nonce back to B, after performing the subtract function.

vi. B receives the message checks the nonce after decryption and a connection is established between them.

Output

```
Console
<terminated> needhamschroder$ [Scala Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 29, 2013 7:01:51 PM)
Started
Starting Simulaiton for Needham
2186484674831872
A sending message to B
Nonce Value at A  -2051320793
Nonce Value at B after subtracting 1  -2051320794
Nonce value of B  538621547
Nonce Value at A after subtraction  538621546
Log In successful
```

3. Yahalom protocol[10]

Basic Idea and Implementation

The protocol is the revamped version of the Wide mouth frog protocol.

i. A sends a message with a nonce to B requesting communication with B.

ii. B then sends the credentials of A to KDC along with its nonce.

[10] "Yahalom (protocol) - Wikipedia, the free encyclopedia." 2009. 29 Nov. 2013 <http://en.wikipedia.org/wiki/Yahalom_(protocol)>

iii. KDC sends a message to A with session key K_{ab} with the nonce of B and A encrypted with keys KAS and KBS(AES).

iv. A decrypts with AES.decrypt() and then sends this the nonce of B encrypted with session key and session key encrypted by KDC to B..

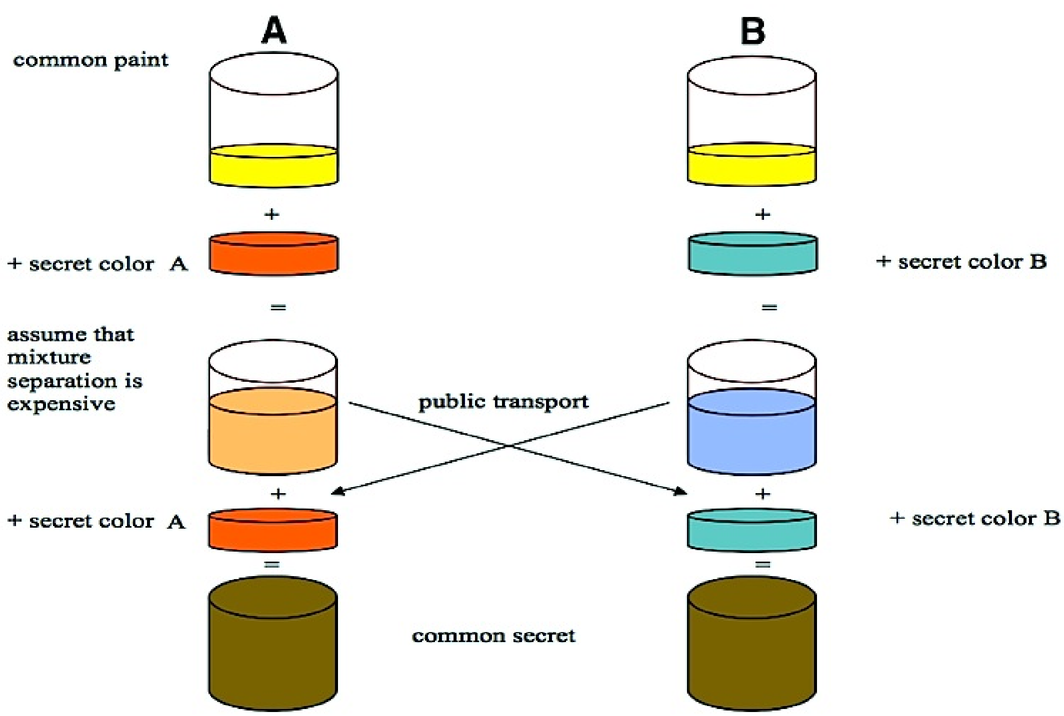
v. Thus B obtains the session key from A with its nonce to confirm the authenticity of the message

- **Output**

```
Console
<terminated> Yahalom$ [Scala Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 29, 2013 7:34:50 PM)
Started
Yahalom
Nonce of A -1397121316
Nonce of B 521631809
Sending message from erver to A
Nonce of A in msg -1397121316
Session Key at A 5282171312128222
Nonce of B after receiving message 521631809
Session key at B  5282171312128222
Connection Established
```

4. Diffie-Hellman protocol[11]

- **Basic Idea:**[12]



[11] "Diffie–Hellman key exchange - Wikipedia, the free encyclopedia." 2009. 29 Nov. 2013 <http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange>

[12] Brit Cruise. "Diffie-hellman key exchange | Intro to Modern Cryptography | Khan ..." 2013. 29 Nov. 2013 <http://www.khanacademy.org/math/applied-math/cryptography/modern-crypt/v/diffie-hellman-key-exchange--part-2>

i. The basic idea is related to the Public-Key cryptography and some common sense of mixing colors.
   1. Both parties public agree on the base color.
   2. Each one has its own public color, and they exchange with each other.
   3. Each one has its own private color, never being exchanged.
   4. Each one mix base color, other's public color and its own private color to get the secrete color, which can't be attained by 3rd party without knowing private color.

- **Implementation Detail:**

i. A generates its randomly chose private key, PR_A, send a pair of number =(Generator, Prime Modular),(g,p), and its randomly chose public key, PU_A, to B

ii. B replies its public key number, $PU_B = g^{PRbob} \% p$, to A
    and the session key can be calculated as, Session = $PU_A^{PRb} \% p$

iii. A can calculate the session key = $PU_b^{PRa} \% p$

iv. A and B establish a consensus of a secret session key

- **Output:**

```
Console
<terminated> Main$ [Scala Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 29, 2013 10:43:01 PM)
Init message to Bob, generator = 2 , primeModular = 17 PUalice = 4
Bob received PUalice, session key= 1
Alice received PUbob = 1, session key = 1
```

- **After Thought**

i. The assumption is that we can't trust the communication medium, so every message being sent is assumed to be eavesdropped.

ii. What we have sent is:
   1. (g,p) → this is like encrypt and decrypt algorithm
   2. Public key

   The session key is computed locally with its own private key, that is never being sent.

iii. So all we do here is to transfer the security pressure from establishing a safe key distribution channel to securely protect its own private key. Would it be better? It depends on the cost of two options
   1. to create a relatively secure key exchanging channel
   2. to securely protect the private key storing locally.

5. Modified Challenge Response mechanism

- **Motivation and Implementation:**

The challenge response system we implemented here is the improvement on the traditional system. This system uses asymmetric cryptography. Here both A and B knows the public keys of each other. The public key provided here is derived using RSA algorithm making it mathematically improbable to derive the private key from the public key.

$$A \rightarrow B:\{A\}_{KSA,KPB}$$

$$B \rightarrow A:\{B, K_{AB}\}_{KSB,KPA}$$

$$A \rightarrow B:\{K_{AB}-1\}_{KAB}$$

K_{SA} and K_{SB} represents the private keys of A and B respectively. K_{PA} and K_{PB} represent the public keys of A and B. K_{AB} is the session key which is produced randomly using the key generation function of AES. Thus the key produced is 128bits long. This system is simplistic where only three messages are passed between A and B to secure a connection between them. It can be seen that the key generated is also used as the nonce. This simple system is also robust to various attacks to which the network is subjected to. Even if one of the system is compromised the performance of the other system will not be hampered as only the public key of the other will be known therefore the incentive to crack one system's encryption is very less in comparison with a third party system protocol where if KDC is compromised the private keys of all the system will be known. Finally this system can be extended.

The use of a nonce by A is intentionally removed because even if the B is subjected to a replay attack by C, C only obtains a session key which has been encrypted with the public key of A. Therefore this system proves to be an efficient, quick and simplistic mechanism for the establishment of a secure communication channel.

ackRegistration() starts the simulation requesting connection with the B (server) with the id of A (client). The server heeds the request and sends an encrypted key to A which is decrypted with its private key and a subtraction response similar to that of Needham Schroeder is computed and sent back to B. Thus a connection is established in the system.

- **Output:**

```
Console
<terminated> challengeresponse$ [Scala Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 29, 2013 10:22:38 PM)
A initiating login
Challenge produced 5442288712323773
Challenge acceptance  5442288712323773
CHallenge resolution 5442288712323773
LogIn Sucessful
```

Team member: Shrinivass Av , Chao-Hsuan Shen

Attacking scenarios → reactions & features

	Otway Rees	Needham-Schroeder	Yahalom	Diffie-Hellman	Challenge Response
Eavesdropping on communication channel	Requires any one of the keys to cause an impact	Requires any one of the keys to cause an impact	Requires any one of the keys to cause an impact	No effect. Attacker can't derive session key only by public keys	
Server got hacked, user_name and password get public	Single point failure all the symmetric keys have to reseted	Single point failure all the symmetric keys have to reseted	Single point failure all the symmetric keys have to reseted	not involve with server.	Only public keys will be obtained.
Imposter	Verifies for both entities	Verifies for both entities	Assumes one identity is authentic	assumes two parties have already know each other	Verifies implicitly
user computer got hacked and its private key is retrieved	Security Breached	Security Breached	Security Breached	security breached	Security Breached
Replay Attack	Counters this threat	Causes Problem[13]	Counters this threat	this protocol doesn't care	Counters this threat
Man in the Middle Attack	Protected	Protected	Protected	no role verification	Protected
Number Of messages exchanged	4	5	4	2	3

Who does what?

- **Shrinivass AV:**
  - Implement AES, RSA classes for decryption and encryption.
  - Implement Otway Rees, Needham-Schroeder, Yahalom, Challenge Response protocols.

[13] "info | Needham-Schroeder - Xklsv.org." 2012. 30 Nov. 2013 <http://www.xklsv.org/viewwiki.php?title=Needham-Schroeder>

Team member: Shrinivass Av , Chao-Hsuan Shen

- **Chao-Hsuan Shen:**
  - Design and implement the backbone architecture and message passing interface. Based on that all protocols can be expanded.
  - Implement Diffie-Hellman protocol.
- **Both**
  - The paper report is compiled by both of us.

Compilation Instructions:

1. Unzip Project.rar
2. Compile the classes present in Project/project/src/ using scalac or sbt

Conclusion

The modified challenge response system mechanism provides better efficiency and security than corresponding protocols that were compared with it. This type of system can be implemented as a Client-server communication model and will enhance the security of the network system.

Self-evaluation	Likert scale
Completeness	4
Proper use of language	4
Organization	4
Clarity	4
Creativity	4
Technical Correctness	4
Level of Difficulty	4