PS4 – DESIGN OVERVIEW

CS 6740 – NETWORK SECURITY

FALL 2017

Soumya Mohanty
Suraj Bhatia

CONTENTS

- THE SETUP
  - Assumptions
  - Architecture
  - Keys
- PROTOCOLS
  - Client <-> Server Authentication
  - Client <-> Client Authentication
  - Client <-> Client Message exchange
  - Client <-> Server Logout
- PROTECTION

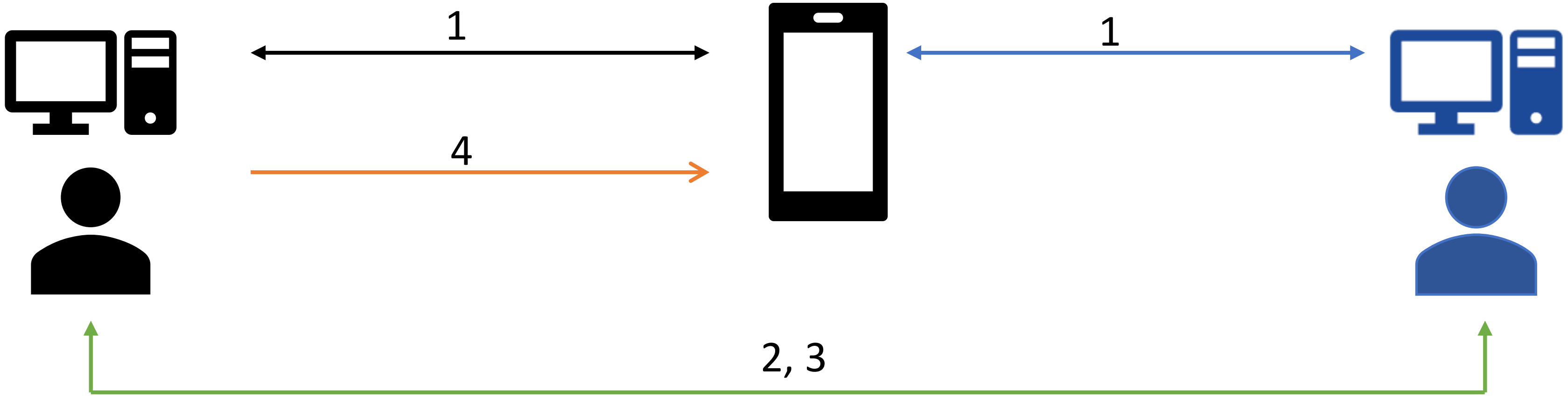
ASSUMPTIONS

CLIENT	SERVER	USER
Knows Server's Public key	Knows it's private key	Remembers user login and password
Does not store user password	Does not know client's public key until it sends it	
Does not store any keys i.e. all keys are cleared after a session termination	Knows username and password hashes of all registered users (like UNIX)	

KEY VALUES

- SHA512 for hashing
- AES with GCM mode
- Diffie-Hellman for Session key generation
- RSA for public/private key generation

ARCHITECTURE



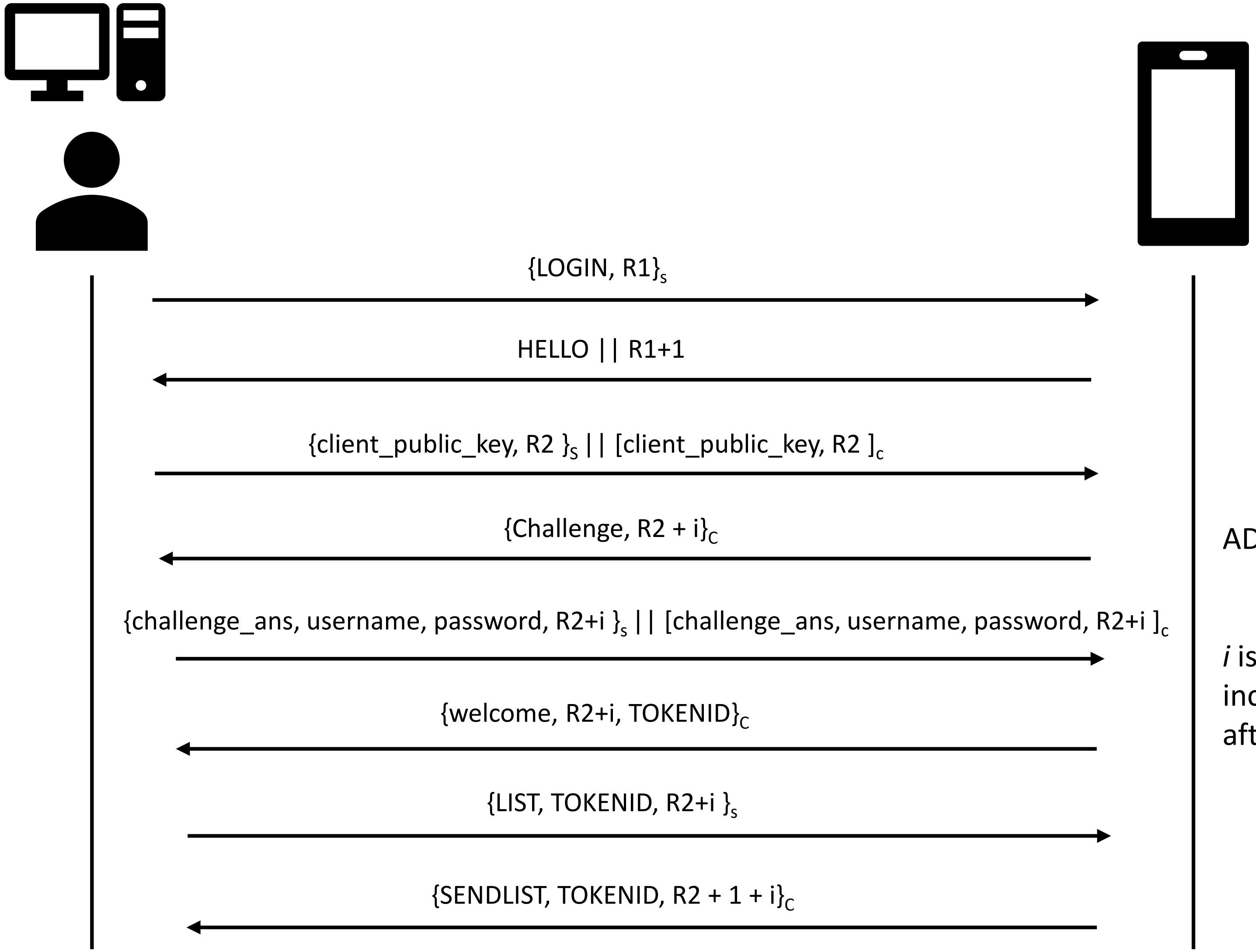
- <u>Basic set of messages exchanged for proper communication -</u>

1 – Authentication (Login with username and password)
2 – Establishment of keys
3 – Message exchange between clients in format *send USER MESSAGE*
4 – Logout

- <u>Other messages exchanged</u> –

list – Send/request logged-in client list
error – Handle erroneous messages

CLIENT <-> SERVER AUTHENTICATION



{LOGIN, R1}$_s$

HELLO || R1+1

{client_public_key, R2 }$_S$ || [client_public_key, R2]$_c$

{Challenge, R2 + i}$_C$

{challenge_ans, username, password, R2+i }$_s$ || [challenge_ans, username, password, R2+i]$_c$

{welcome, R2+i, TOKENID}$_C$

{LIST, TOKENID, R2+i }$_s$

{SENDLIST, TOKENID, R2 + 1 + i}$_C$

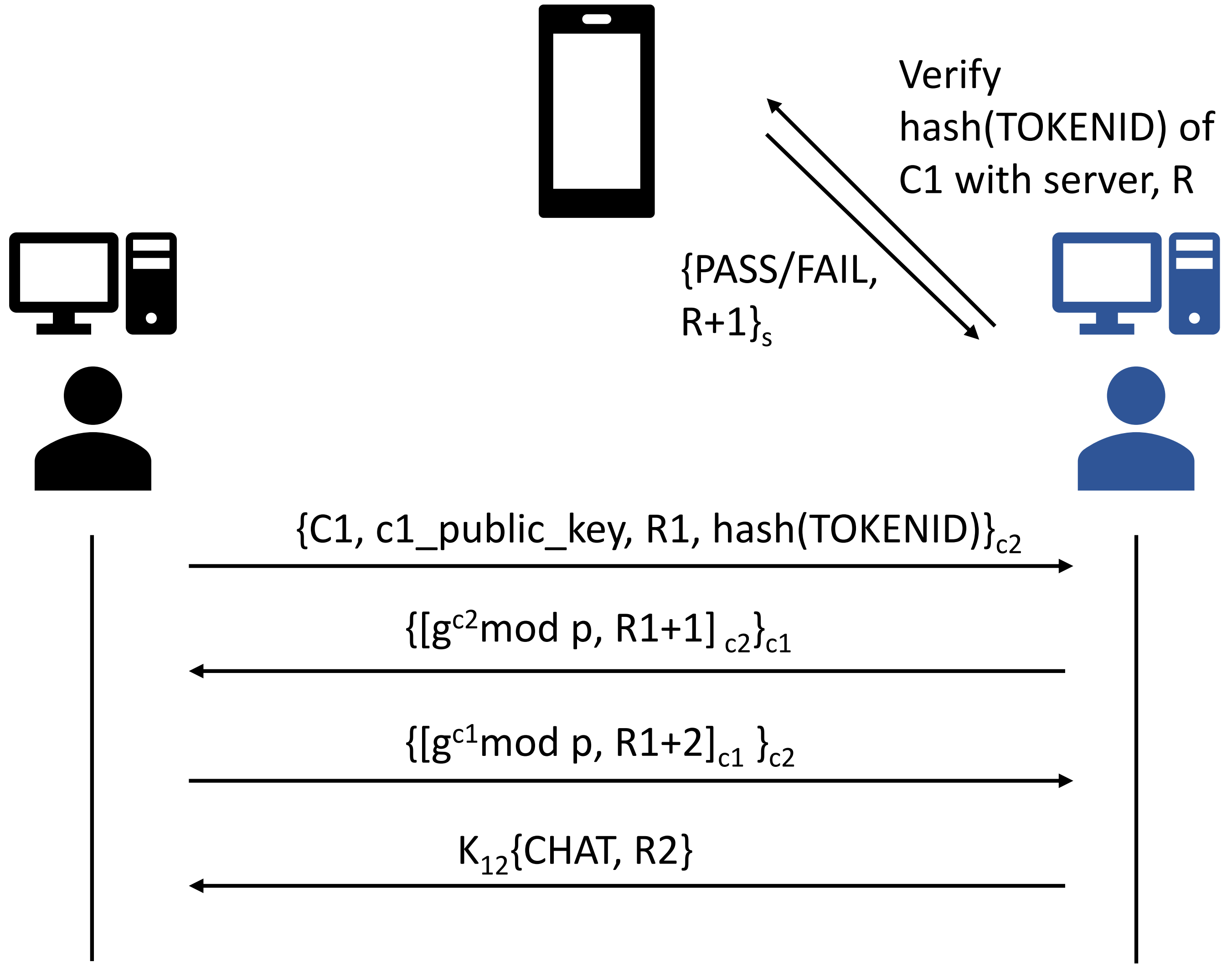
TOKENID =
ADDRESS + CHALLENGE_ANS

i is a number that will be initialized to 1 and incremented by 1 after each step. Reset to 1 after session ends.

EXPLANATION

- STEP 1 – Client sends a LOGIN request and a random number. Message is encrypted using server’s public key.
- STEP 2 – Server will send HELLO and the decrypted random number.
- STEP 3 – Client sends public key and new random number encrypted with server’s public key and signs it with it’s private key.
- STEP 4 – Server sends a challenge and incremented random number by *i* encrypted with client’s public key
- STEP 5 – Client will send USERNAME, PASSWORD, challenge answer and a number incremented by *i*. Message in encrypted with server’s public key and signed with client’s private key.
- STEP 6 – Server authenticates client and sends WELCOME and a TOKENID.
- STEP 7 – Client sends LIST command, TOKEN ID and a incremented random number by *i*.
- STEP 8 – Server sends LIST with TOKENID and the incremented value.

CLIENT <-> CLIENT AUTHENTICATION



Verify hash(TOKENID) of C1 with server, R

{PASS/FAIL, R+1}$_{s}$

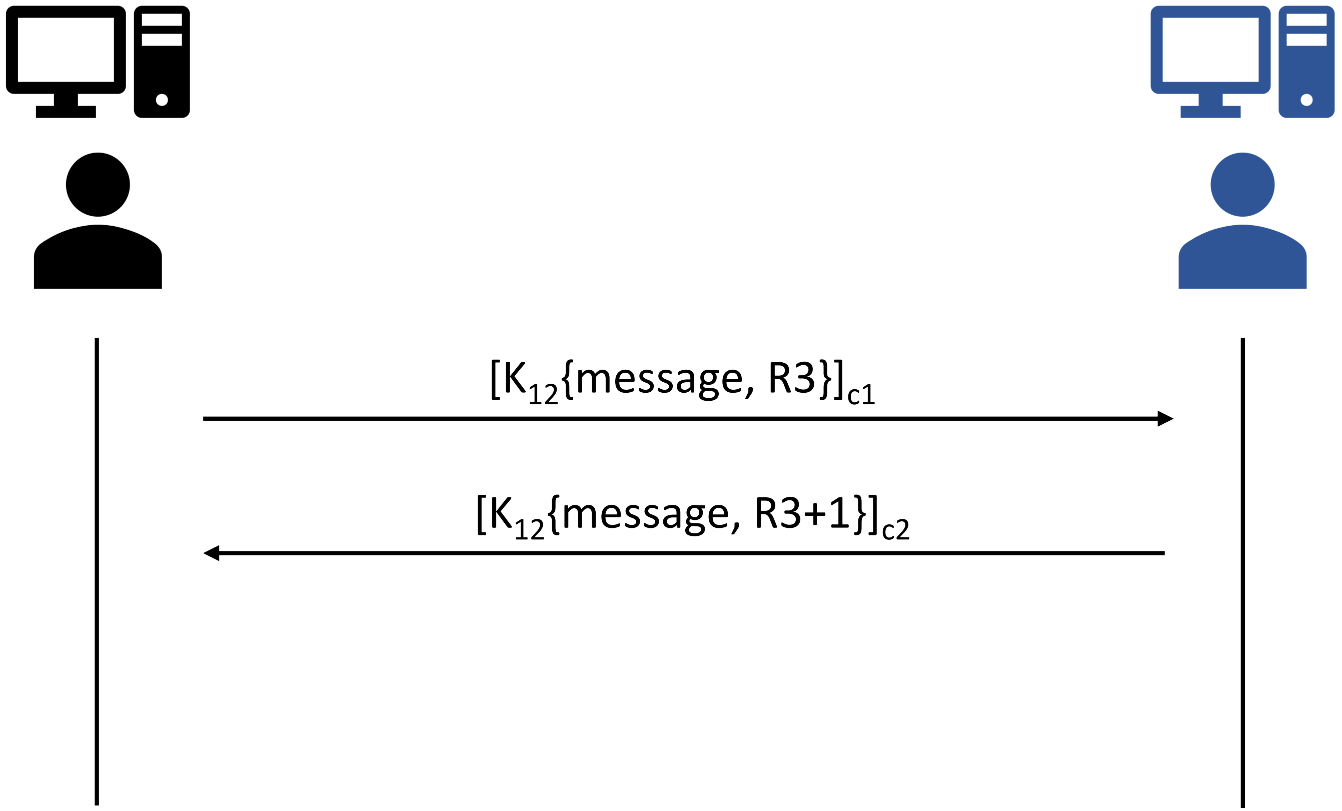
{C1, c1_public_key, R1, hash(TOKENID)}$_{c2}$

{[g^{c2}mod p, R1+1] $_{c2}$}$_{c1}$

{[g^{c1}mod p, R1+2]$_{c1}$ }$_{c2}$

K$_{12}${CHAT, R2}

CLIENT <-> CLIENT MESSAGE EXCHANGE



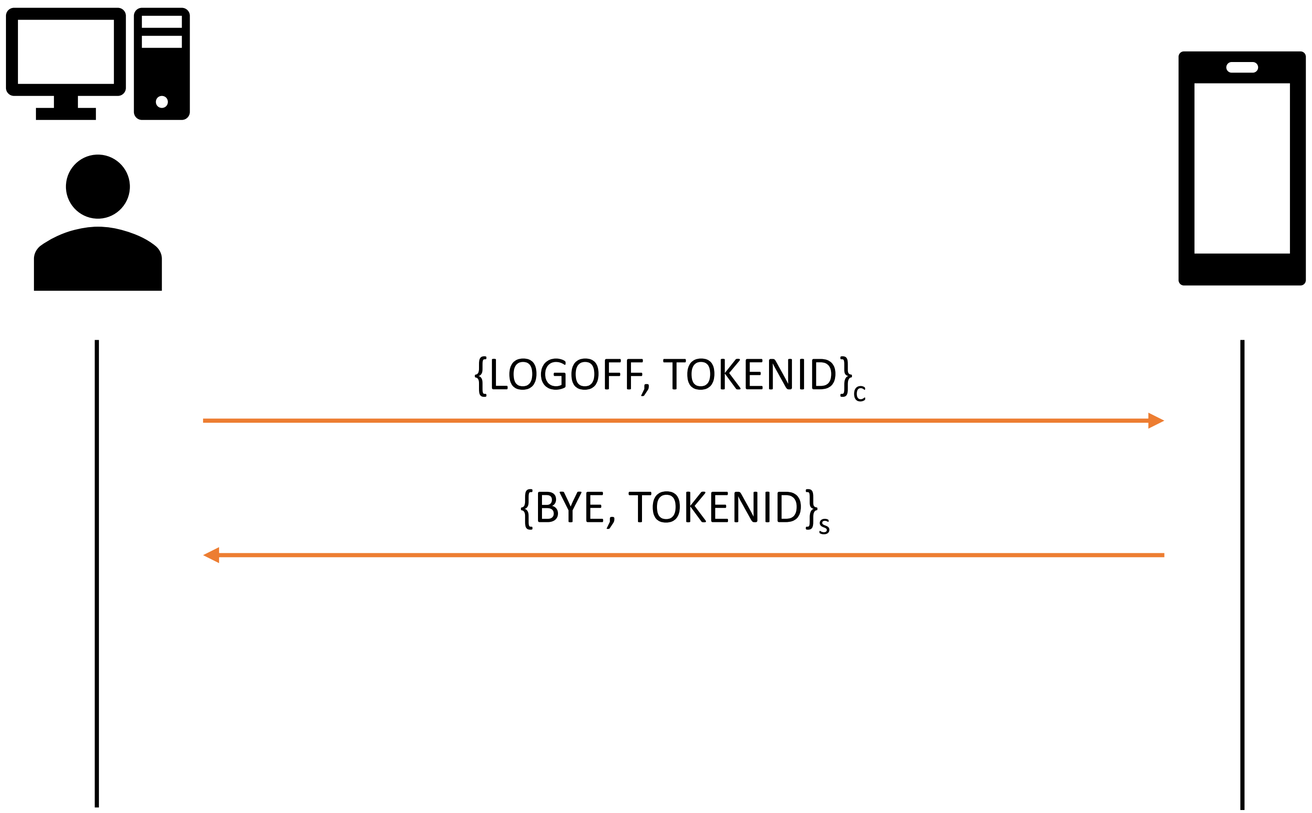
$[K_{12}\{\text{message}, R3\}]_{c1}$

$[K_{12}\{\text{message}, R3+1\}]_{c2}$

Explanation

- STEP 1 – C1 sends username, hash of TOKENID, random number and it’s public key to C2
- STEP 2 – C2 verifies identification of C1 with server by using hash of TOKENID
- STEP 3 – C2 sends username, first half of Diffie-Hellman key and incremented random number. Message is encrypted using C1’s public key and signed using C2’s private key.
- STEP 4 – C1 sends second half of Diffie-Hellman key and incremented random number. Message is encrypted using C2’s public key and signed using C1’s private key.
- STEP 5 – Ready to exchange messages by encrypting with shared key K_{12} and signing with it’s private key.

CLIENT <-> SERVER LOGGING OFF



$\{\text{LOGOFF, TOKENID}\}_c$

$\{\text{BYE, TOKENID}\}_s$

EXPLANATION

- STEP 1 – Client will send LOGOUT message with the TOKENID
- STEP 2 – Server will clear all data related to client i.e. Address, port, keys and token.
- STEP 3 – Server broadcasts an updated list to all clients that are still logged in.

PROTECTION AGAINST ATTACKS

- DoS
  - Server checks for all half-open connections time-to-time and if found, terminates it.
  - Client is denied log in after *n* attempts for fixed time *t*
- Perfect Forward Secrecy (PFS)
  - The exponents are generated randomly for creating Diffie-Hellman keys and are forgotten after each session terminates.
- End-point identity hiding
  - No user information is sent out in the open.