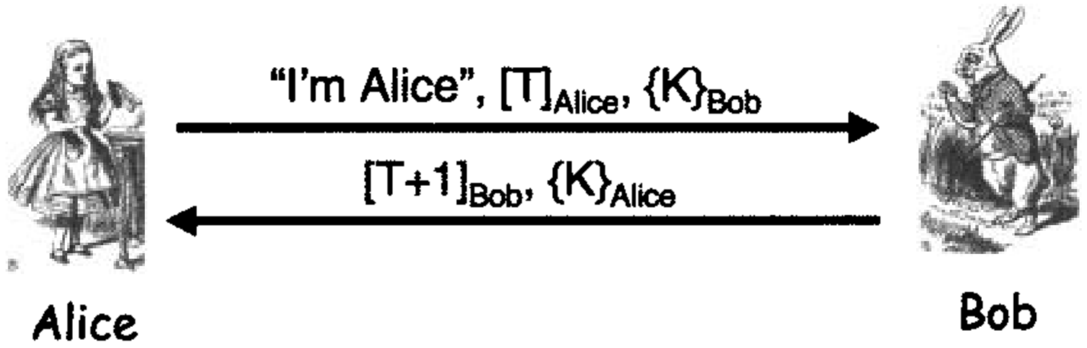
Homework 4 COMP 3632

This assignment counts 10% of your final grade.

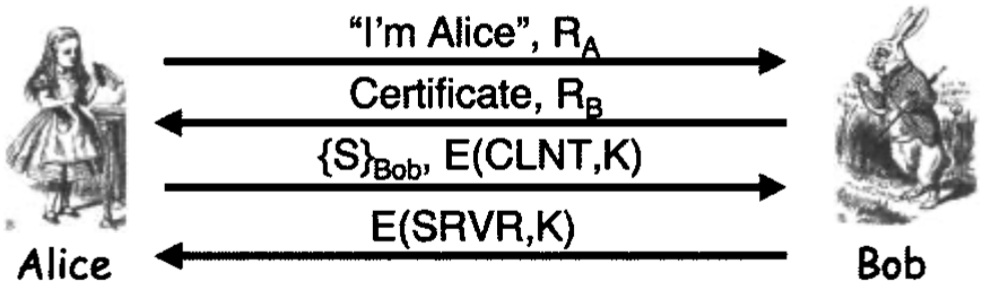
1. (4pt) As discussed in the lecture, timestamps can be used to replace nonces in designing secure protocols.

   (a) (2pt) What is the main advantage of using timestamps?

   (b) (2pt) The following protocol is designed for mutual authentication and to establish a session key K. Here, T is a timestamp. Please provide an attack to this protocol.


"I'm Alice", $[T]_{Alice}$, $\{K\}_{Bob}$
$[T+1]_{Bob}$, $\{K\}_{Alice}$
Alice
Bob

2. (6pt) Considering the following protocol. Suppose Certificate is **not** a problem here; the correct certificate is delivered from Bob to Alice. The session key is $K = h(S, R_A, R_B)$.



   (a) (2pt) We discuss constant string like CLNT and SRVR in the lecture. Why do we need them in typical authentication protocols?

   (b) (2pt) Does Alice authenticate Bob? Explain your answer.

   (c) (2pt) Does Bob authenticate Alice? Explain your answer.

3. (5pt) Smart contract security.

   (a) (2pt) Please identify the vulnerability presented in the following smart contract snippet.

```
pragma solidity ^0.4.0;
contract bug {
    uint public supply = 0;

    mapping(address => uint256) public balances;
    mapping(address => mapping (address => uint256)) allowed;

    function bug(uint256 _initSupply) public{
        balances[msg.sender] += _initSupply;
        supply = _initSupply;
        Transfer(0x0, msg.sender, _initSupply);
    }

    function transfer(address _to, uint _tokens) public returns (bool success){
        require(balances[msg.sender] - _tokens >= 0);
        balances[msg.sender] -= _tokens;
        balances[_to] = balances[_to] += _tokens;
        Transfer(msg.sender, _to, _tokens);
        return true;
    }

    function approve(address _spender, uint _tokens) public returns (bool success) {
        require(balances[msg.sender] >= _tokens);
        allowed[msg.sender][_spender] = _tokens;
        Approval(msg.sender, _spender, _tokens);
        return true;
    }
    ...
    function () public payable{
        uint purchasedTokens = msg.value;
        balances[msg.sender] += purchasedTokens;
        supply += purchasedTokens;
    }
    function withdrawEther(uint _amount) public returns (bool success){
        require(balances[msg.sender] >= _amount);
        balances[msg.sender] -= _amount;
        supply -= _amount;
        if(!msg.sender.send(_amount / 2)){
            revert();
        }
        return true;
    }
}
```

(b) (3pt) Please elaborate on how the identified vulnerability can be exploited by attackers. How to mitigate this vulnerability?

4. (8pt) We introduce "Proof-of-Work" scheme in the blockchain lecture.

   (a) (2pt) What is proof-of-work (PoW)? Please briefly explain your answer. In typical blockchain scenario, who is responsible to solve PoW?

   (b) (3pt) Ethereum blockchain is adopting a new scheme named "Proof-of-Stake" to substitute the original scheme. Explain what is "Proof-of-Stake" in *your own language*. What is the advantage of PoS comparing to PoW?

   (c) (3pt) Another scheme, namely "Proof of Elapsed Time (PoET)", has also generated some buzz. Explain what is PoET in *your own language*. What is the advantage of PoET comparing to PoW?

5. (6pt) Smart contract attack and defense.

   (a) (4pt) Consider the following contract, which acts as part of a typical Raffle game, where users transfer certain amount of Ether to function `process` and also reserve a number as the function input.

```
contract Raffle {
  mapping(uint256 => address) reserved;

  function process(uint256 value) public {
  // check whether corresponding entry has been initialized or not
  // If it equals to zero, then it is not initialized
      if (reserved[value] == 0) {
        // can only enter once when uninitialized (0)
        // msg.sender is the address of the user
        // value is the reserved number by the user
        reserved[value] = msg.sender;
      }
  }
}
```

   Explain the potential "Transaction-Ordering-Dependence" bug of the above smart contract.

   (b) (2pt) In the class, we have discussed the problem of using `block.timestamp` as the source of the randomness for gambling contracts. To mitigate the problems, here we proposed the following contract as the random number generator and deployed to public network. What is the advantage or vulnerability of this random number generator? For this question, you can safely assume the complexity for variable `seed` , `key`, implementation of the `keccak256` and `abi.encodePacked` wouldn't be the source of vulnerability and the `key` has already been sanitized. The source code of this smart contract is *not* open-sourced to public.

```
contract RandomGenerator is Ownable {
 bytes32 private seed = "I_am_a_long_string";
 function get_rand(bytes32 key) public onlyOwner returns (bytes32){
  seed ^= key;
  return keccak256(abi.encodePacked(key,seed,"y0u_c4nn0t_gu3ss"));
 }
}
```

6. (8pt) Machine Learning Security. Consider a neural network model G for image classification task is trained on the training dataset $D_{train} = \{x_i, y_i\}_{i=0}^{n-1}$, the predict result for x_i is $\hat{y}_i = G(x_i)$.

   (a) (3pt) Give the definition of adversarial examples and poisoning attacks and describe the corresponding attacker's objective.

   (b) (3pt) If we have access to the testing dataset D_{test} and the trained model G, but have no access to the training dataset D_{train}, can we generate adversarial examples? Can we detect poisoning attacks? Explain your answer.

   (c) (2pt) What would be the possible root cause of adversarial examples for machine learning models? How can we defeat against adversarial examples? For this question, you may need to search the Internet; please use your own language to answer, briefly.

Submission Instructions

All submissions should be done through the Canvas system. You should submit a pdf document with your answers. It is important to name your files correctly. Please check out the late submission policies on the course website (`https://course.cse.ust.hk/comp3632`) in case you didn't attend the first lecture.