A SIMPLE COMMUNICATION GAME

In order to grasp some of the basic ideas of applying cryptography, we begin this class with a simple protocol called coin flipping.

Assume that we have two parties, Alice and Bob. They want to use a coin flipping game to determine who has the final say on where they should go for fun tonight. If Alice and Bob are physically at the same location, the game could be played easily since none of them can cheat the other side. However, problems arise if Alice and Bob are not at the same place and have to play the game without seeing each other. For instance, what happens if they have to play the game over the phone? No matter who flips the coin, the other side has no way to verify it. In order to solve this kind of problem, we need a special method to prevent both sides from cheating each other. Below we will present a procedure between two parties, (a protocol), that achieves just that.

To get some motivation of how coin flipping can be achieved without physical presence, assume that Alice suggests flipping the coin in a very deep well so that after throwing the coin, she has no way to change the result. In this case, the well constitutes a commitment since it enforces Alice to be consistent. However, to verify the result, Bob still has to go the well to look at it (but this may be done at a later time).

Now let’s try to use some “magic” function $f: D \rightarrow R$ to solve the coin flipping problem over the telephone. The function $f$ in this case will serve the role of the well that allows some sort of commitment. Here are the steps:

1) Alice selects $x \in D$ at random. ($x$ is a bit string.)
2) Alice transmits to Bob the value $y = f(x)$. (The function $f$ here is like the well.)
3) Bob guesses the least significant bit (LSB) of $x$, and denotes it as $b$.
4) Bob sends to Alice his guess $b$.
5) If $b = \text{LSB}(x)$, Alice says “Heads”. Otherwise, Alice says “Tails”.
6) Alice sends $x$ to Bob. (This step is like taking Bob to the well.)
7) Bob verifies that $f(x) = y$, and if $b = \text{LSB}(x)$, Bob says “Heads”, otherwise Bob says “Tail”.

In order for the above protocol to qualify for a solution to coin-flipping, the function $f$ must satisfy a number of properties:

Desired Properties of $f$ (informally)

1) $f$ is easy to compute.
2) $f$ is hard to invert.
3) Collision resistant. It is hard to find $x, x'$ with $x \neq x' \Rightarrow f(x) = f(x')$

Property (1) is required for the protocol to be actually executed by the two parties. Observe now that property (3) protects Bob from a cheating Alice.
Property (2) protects Alice from Bob’s cheating because property (2) makes for Bob difficult to invert what $x$ is. However, property (2) may not good enough because Bob may still easily get the least significant bit of $x$. Therefore property (2) must be refined so that guessing the $LSB(x)$ given $f(x)$ is hard (i.e., this means that the least significant bit of $x$ is a so-called “hard bit” for the function $f$).

In practice (and with ad-hoc security) one can implement the above protocol by substituting the function $f$ with some hash function such as SHA-1 or MD5.

**Tasks related to our class**

1) Specifications of various security related problems.
2) Design of solutions of such problems.
3) Identifying required the properties/assumptions of the various components employed in a solution.
4) Giving formal arguments why the specifications are met based on the assumptions.
5) Investigate the possible implementations of the components.

**Areas of Interest**

1) Cryptography: “Foundations”

   Dealing with the basic properties of the underlying primitives. For example:

   How do we formalize “Is $f$ hard to invert”?
   Is hard to invert sufficient for the existence of a ‘hard bit’ in $f$?

2) Cryptography: “Practice”

   How to implement function $f$ in an efficient way?
   what are natural examples of such functions?
   under what assumptions the required properties can be achieved?

3) “Computer Security”:

   Making sure that an implementation of the solution actually conforms to its intended design.
   E.g. can we implement properly the step “select $x \in D$ at random”?
   What are the hardware/software requirements/assumptions?