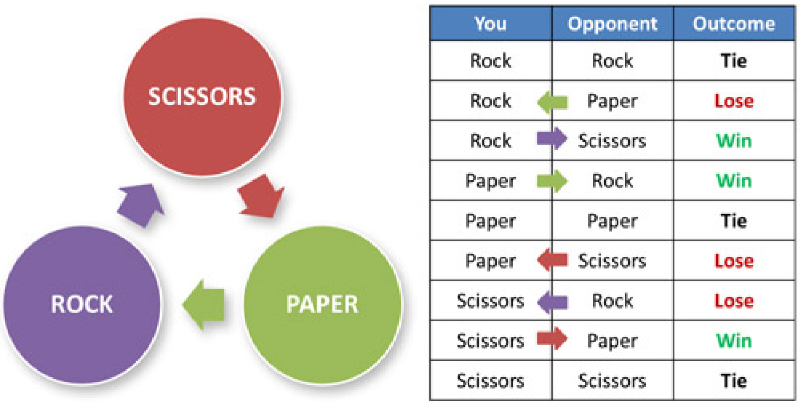
Rock-paper-scissors hand game (ro sham bo)

Tufa Alexandru Daniel 1628927

17 November 2017

1 Introduction

Rock-paper-scissors is a hand game usually played between two people. Each player simultaneously makes a choice between rock (a closed fist), paper (a flat hand) and scissors (a fist with the index an middle fingers extended) and shows it by outstretching the hand. In a single game the outcome can be a win, a loss or a draw depending on the choices the two players made. As we can see in the figure no single choice prevales over another so there is no winning or losing decision. For this reason it is highly used to settle a dispute or make an unbiased decision given its randomness. However, in a match that lasts more than one game, skilled players can recognize and exploit non random behavior in opponents.



You	Opponent	Outcome
Rock	Rock	Tie
Rock	← Paper	Lose
Rock	→ Scissors	Win
Paper	→ Rock	Win
Paper	Paper	Tie
Paper	← Scissors	Lose
Scissors	← Rock	Lose
Scissors	→ Paper	Win
Scissors	Scissors	Tie

Figure 1: Possible outcomes

2 Description

We want to describe a protocol that allows a match of N = 2p + 1 games to be played between two players by exchanging messages. We need to enforce data integrity, prevent players from obtaining unfair advantage from pre-computation and make sure that games are played in real time by not allowing time to the members to brute-force messages. We can assume that there are no third parties so don't have to worry about attacks and we shouldn't give the players the possibility to cheat; we assume that if they are given the choice of cheating without being discovered they will not hesitate in doing so. Without loss of generality we'll call our members Alice and Bob and assume Alice is always the first one to start the conversation.

3 Components

In order to create such protocol we require some ingredients that will be used:

- Cryptographically Secure Pseudo Random Number Generator must be used so that the encryption key is always different and secure. If we used the same key, after the first game Bob would always make a winning choice and to avoid this we need different keys and also secure so that Bob can't predict them.

- AES-256 is our chosen algorithm to encrypt the choice of the players so that we ensure a cheat-proof protocol

- SHA-256 is the Secure Hash Algorithm that we'll pass to our message authentication code.

- HMAC is the way in which we verify data integrity. This mechanism uses a cryptographic hash function, in our case SHA-256, and also a secret cryptographic key. It's defined as follows:

  $$HMAC_{\mathrm{K}}(m, H) = H((k \oplus opad || H(k \oplus ipad || m)))$$

  where opad is the outer padding (0x5c5c5c...5c5c, one-block-long) and ipad is the inner padding (0x363636...3636, one-block-long)

4 Initialization

To ensure data integrity, prevention against lost messages and to enforce an anti-cheating environment an initialization protocol has to be used. Firstly data integrity has to be assured, so both parties need to know the HMAC key. This can be done in the following way:

1. Alice chooses a random number R, it doesn't have to be cryptographically secure, and computes M = HMAC_K(R,H) and sends to BOB (R, K)

2. Bob receives (R, K) , computes HMAC_K(R) = $\text{M}^{'}$ and sends it back to Alice

3. If M is equal to $\text{M}^{'}$ then Bob has received the right key else restart the protocol from point 1

After this point we assume that both parties agree on the same HMAC key. Every message M will be sent in the form (M,HMAC_K(M,H)) and if the hash of the messages differs from the received TAG then every party will reply with a "WRONG" message and wait for a correct message. If a member receives a "WRONG" message then the last procedure is repeated.
What if a message is lost? To prevent this both players pick a random Δ and if a message doesn't arrive when this timeout expires then Δ is doubled and the previous procedure is repeated. Because of this delay more than one identical messages could arrive so every new message is stored in a variable so that equal arrived messages are discarded, while a new message will replace the old one in the variable.
To prevent cheating both parties will initialize two variables `wins` and `gameCounter` to 0. They must also agree on the number N of games that are to be played. This is done after the agreement of the key of HMAC:

1. Alice sends to Bob (N, TAG)

2. If Bob agrees with N he sends back (OK, TAG) and initializes his local variables. If this is not the case he sends to Alice (NOT OK, TAG)

3. If Alice receives (OK, TAG) from Bob she initializes her local variables otherwise the conversation is restarted from point 1 with a different N

5 Game Protocol

If the choice of a player is sent in plain text, obviously the other member will make a winning choice. This can be avoided if every choice is sent encrypted with a different key between each member and between each round. After both parties have received the choice of the other member, they receive the decrypting key. To avoid that a player stores the key to decrypt a successive message, a new random key is generated every round using our chosen CSPRNG.

1. Alice generates $\mathrm{C_A} = \mathrm{AES_{K1}}$(”`choice`”), computes $\mathrm{TAG} = \mathrm{HMAC_K}(\mathrm{C_A},\mathrm{H})$ and sends to Bob ($\mathrm{C_A}$,TAG)

2. Bob receives ($\mathrm{C_A}$,TAG), stores $\mathrm{C_A}$, computes $\mathrm{C_B} = \mathrm{AES_{K2}}$(”`choice`”)[1] and $\mathrm{TAG} = \mathrm{HMAC_K}(\mathrm{C_B},\mathrm{H})$ and sends to Alice ($\mathrm{C_B}$,TAG)

3. Alice stores $\mathrm{C_B}$ and sends ($\mathrm{K_1}$, $\mathrm{HMAC_K}(\mathrm{K1},\mathrm{H})$)

4. Bob receives K1, computes $\mathrm{A} = \mathrm{AES_{K1}}(\mathrm{C_A})$ and is the first member that acknowledges who has won. If he is the winner he increments his local variable `wins`, afterwards he sends to Alice (K2,WINNER[2],TAG[3])

5. Alice computes $\mathrm{B} = \mathrm{AES_{K2}}(\mathrm{C_B})$, checks who has won this game and verifies if this choice is coherent with the one sent by Bob. If she is the winner she increments her `wins` variable.

6. Alice increments `gameCounter` and checks if it's equal to N. If this is not the case,then all the protocol is repeated from step 1 after she has send to Bob (REPEAT, TAG) so that Bob can also update it's local variable `gameCounter`. If `gameCounter` is equal to N then she checks if her `wins` are more than $\mathrm{p} + 1$ so that she can set WINNER accordingly.

7. If the match has finished Alice sends to Bob (WINNER,TAG)

8. Bob receives WINNER and checks it's correctness by confronting it with his local variable `wins` and also checks if the total number of games played is truly N

[1] Note that the key used by Bob, k2, is different from the key used by Alice, k1

[2] WINNER is the placeholder for the real winner of this game that could be Alice or Bob

[3] Given a message M, for semplicity we will denote $\mathrm{TAG} = \mathrm{HMAC_K}(\mathrm{M},\mathrm{H})$

At the end of the protocol both players are aware and agree on the winner of the match. Because the messages are sent encrypted, Bob can't pick a choice that would be better than the one made by Alice. Also no party member can claim that they have won if it's not a correct statement, because the other player would verify it by confronting with his local variable `wins`. We also ensured that Alice can't play an infinite number of riunds until she wins because otherwise, at the end, thanks to his local variable `gameCounter` Bob would notice that it's higher than N.