Criptografía

Segundo Cuatrimestre de 2007

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico

Truco Mental

Integrante	LU	Correo electrónico
Albanesi, Matías	??/??	@
Carbajo, Pablo	717/04	pcarbajo@dc.uba.ar
Freijo, Diego	4/05	giga.freijo@gmail.com
Venturini, Maura	??/??	@

Abstracto

En el presente trabajo se diseño e implementó un protocolo para jugar al truco mediante el cual se asegura el cumplimiento de las reglas de juego.

Palabras Clave

Encriptación, algoritmos de clave pública/privada y simétricos, RSA, AES

Índice

1. El juego del truco **4**

1.1. Reglas de juego 4

1.2. Lógica de juego 4

2. El protocolo **8**

2.1. El reparto 8

2.1.1. Protocolo de reparto de cartas 8

2.1.2. Pre-inicio del juego 10

2.2. Transcurso del juego 11

3. Consideraciones **13**

El objetivo del trabajo es diseñar e implementar un protocolo para el reparto de cartas y juego de Truco Mental entre dos jugadores. El protocolo está diseñado para garantizar una justa repartición de cartas y que cada parte juege sólo cartas que le fueron repartidas.

El lenguaje elegido es Python. La implementación realizada permite que dos jugadores juegen una mano a través de una conexión por red. Al ejecutar el programa, éste pregunta si debe ser ejecutado en modo cliente o modo servidor. Si se ejecutaran varias instancias del servidor en la misma computadora, se podría dar un servicio de truco a varios jugadores simultánea e independientemente.

1. El juego del truco

1.1. Reglas de juego

Si bien reglas del juegos varían en distintos países, en

[http://usuarios.arnet.com.ar/leo890/truco.htm]

se puede encontrar una expliación de las reglas del juego tal cual se juega en Argentina. A lo largo del trabajo usaremos el término submano para referirnos a lo que el citado documento se refiere como proceso de bajar cartas. De esta manera cada mano se puede dividir en 3 submanos, denominadas primera, segunda y tercera. Nótese que una mano puede finalizar antes de ser jugadas las 3 submanos, es incluso una submano puede quedar inconclusa si la mano termina antes de que la submano termine.

1.2. Lógica de juego

1- Manejo de las Cartas ==========

Escala: 0 01E 1 01B 2 07E 3 07O 4 03O,03C,03B,03E 5 02O,02C,02B,02E 6 01O,01C 7 12O,12C,12B,12E 8 11O,11C,11B,11E 9 10O,10c,10B,10E 10 07B,07C 11 06O,06C,06B,06E 12 05O,05C,05B,05E 13 04O,04C,04B,04E

Se define un orden parcial entre las cartas, que indica el valor relativo entre ellas. Esto nos va a servir para despues comparar, en base a las cartas en la mesa, quién "mató", y a quién le corresponde jugar.

* Para ver quién gana la mano, buscamos las dos cartas que están sobre la mesa y vemos cuál tiene el nivel más alto (esa es la que gana la mano). En caso de estar en niveles iguales hay parda y le toca jugar al jugador que es mano. carta (la mano).

2- Manejo de Cantos ==========

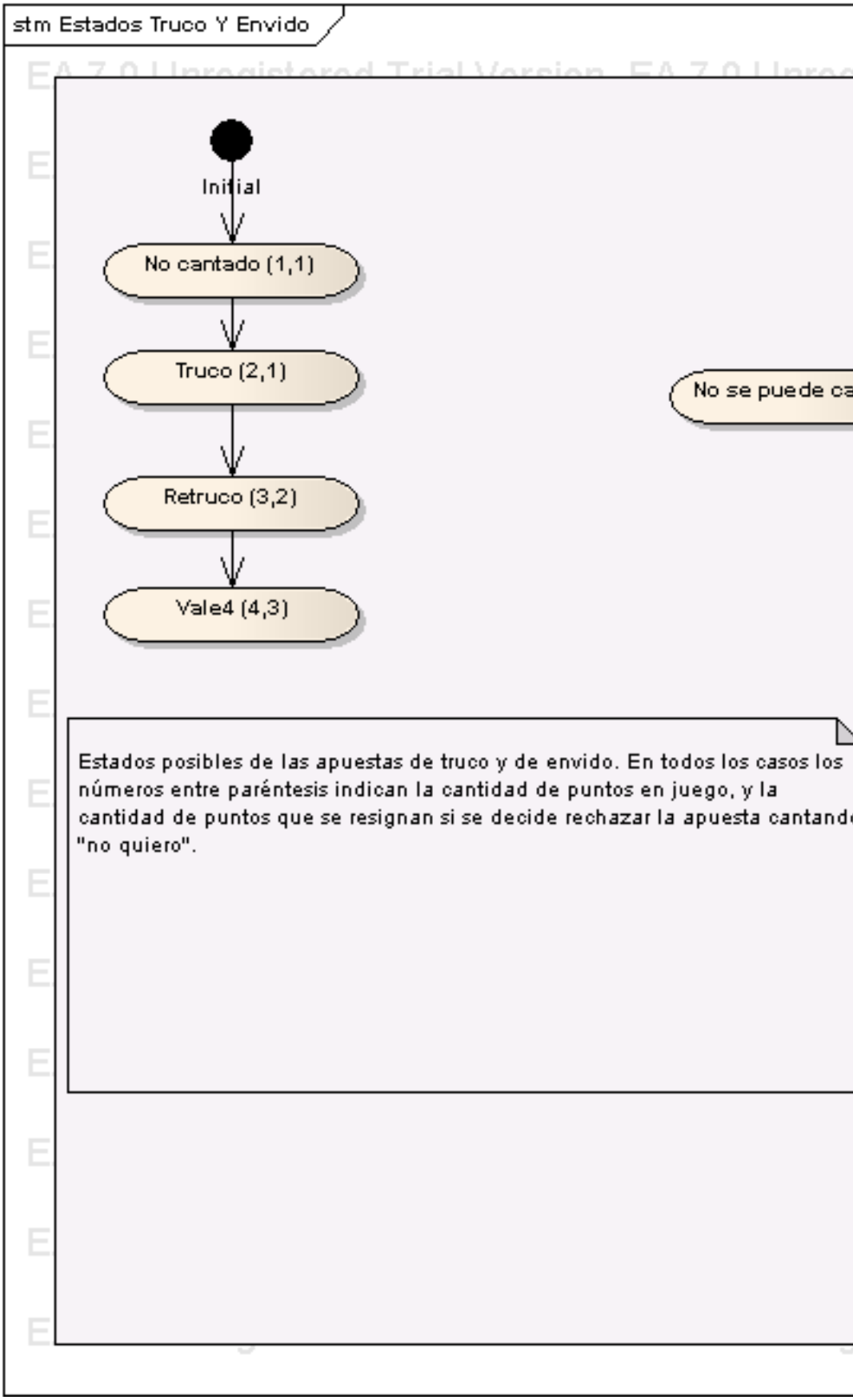
Envido —— * Se puede cantar Envido hasta antes de jugar la primera mano * Se puede seguir contestando hasta que se diga "Quiero"; acá se procede a ver qué cartas son del mismo palo y se suman los valores (en caso de tenes flor se suman los dos más altos), o "No Quiero"; no cuento nada y sigue el juego. * Si se cantó "Truco"no se puede cantar ."En"vido"* Si ya se dijo "Quiero."o "No Quiero", no se puede seguir cantando.

Truco —— * Se puede cantar en cualquier mano e inclusive antes de jugar la primera carta. * El ganador de 2 de 3 manos es el que gana la mano. * Una vez que los 2 jugaron las 3 cartas no se puede cantar "Truco".

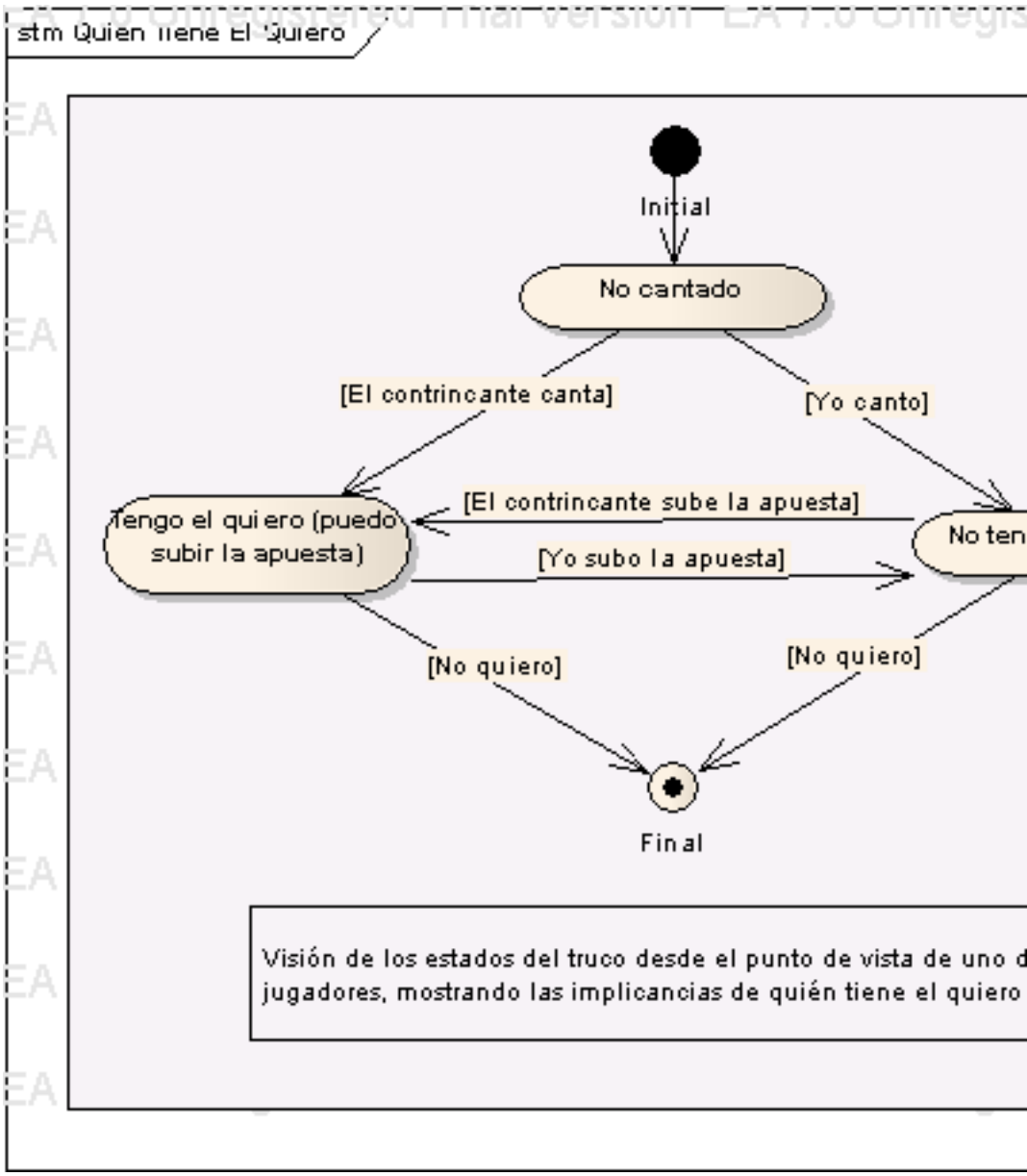
* Los cantos deben manejarse como interrupciones. Un jugador puede cantar "Truco"habiendo jugado su carta o no. No es asi el caso del ."En"vido"; aca hay que controlar que el jugador tenga el token (le toque jugar) y no haya tirado su carta.

3- Registros a Tener en Cuenta: ==========

* Cantidad de Manos Ganadas * Si se cantó ."En"vido"* Si se cantó "Truco"* Al finalizar la mano: - Si se cantó ."En"vido"me fijo si en las cartas sobre la mesa están



Truco Y Envido.png



Tiene El Quiero.png

los puntos del .Envido". Si no, muestro las que faltan aunque se haya ido al mazo. - Si se fue al mazo (o se canta "Trucoz no se quiere), entonces no se deben mostrar las cartas que no se hayan jugado a menos que ocurra el caso anterior.

4- Decisiones de Implementación: ==========

** Se van a necesitar las siguientes variables para saber en que estado estamos dentro de Envido durante la partida:

** Envido: —— Una variable EstadoEnvido que va a asumir los valores: 0, si no se canto nada 1, si se canto Envido 2, si se canto Envido Envido 3, si se canto Real Envido 4, si se canto Falta Envido 5, si ya no se puede seguir cantando (si el jugador respondio "Quiero.º "No Quiero") Una variable que almacene la cantidad de puntos en juego si se dice Quiero (PtosQuieroEnv) Una variable que almacene la cantidad de puntos en juego si se dice No Quiero (PtosNQuieroEnv) Una variable que me indique si puedo redoblar la apuesta del Envido (Es decir, si tengo el Quiero,

PuedoCantarEnv)

Nota: La cantidad de Puntos en juego si se dice "No Quiero.es la cantidad de cantos que se hayan efectuado y la cantidad de puntos en juego si se dice "Quiero.es sumar 2 si se canta Envido, 3 si se canta Real Envido. Se tomo esta decision ya quela cantidad de puntos en juego si se canta Envido, Envido, Real Envido no es la misma que si se canta Envido, Real Envido.

Supongamos que la variable PtosQuieroEnv no existe y que manejamos el Envido con las otras dos variables. En el primer caso, la variable PtosNQuieroEnv tendria un valor 3 y en el segundo caso 2. La variable EstadoEnvido en el primer caso, pasaria del estado 0 al 1, luego al 2 y finalmente el 3; y en el segundo, pasaria del estado 0 al 1 y finalmente al 3. Ahora, cuando computemos los puntos en juego, suponiendo que se dijo "Quiero.a partir de EstadoEnv, en los dos casos terminaria con el mismo valor!, lo cual es incorrecto. Es por esto que se opto por tener un registro de las estapas del Envido en la que se encuentra (EstadoEnv), la cantidad de puntos en juego si se sice "Quiero"(esto soluciona el problema antes mencionado), y la cantidad de puntos en juego si se dice "No Quiero"que sirve para contar la cantidad de cantos que se efectuaron.

** Truco: — Una variable EstadoTruco que va a asumir los siguientes valores: 0, si no se canto nada 1, si se canto Truco 2, si se canto Re Truco 3, si se canto Vale Cuatro 4, si ya no se puede seguir cantando (si el jugador respondio "Quiero.o "No Quiero") Una variable que cuente los puntos del truco en caso que la respuesta sea "Quiero"(PtosQuieroTru) Una variable que cuente los puntos del truco en caso que la respuesta sea "No Quiero"(PtosNQuieroTru) Una variable que me indique si puedo redoblar la apuesta del Truco (Es decir, si tengo el Quiero, PuedoCantarTru)

Nota: Tanto los puntos del truco querido como los del no querido se van incrementando de a uno. La diferencia esta en que PtosQuieroTru de 0 pasa al valor 2; en cambio, PtosNQuieroTru pasa de 0 a 1.

2. El protocolo

Se define un procotolo que permite el reparto de cartas previo a una mano de truco y la posterior utilización de las mismas durante la mano. El diseño del protocolo se hizo con los objetivos de garantizar la transparencia del juego, tanto en el reparto de las cartas como durante el desarrollo del juego. El protocolo brinda cierta seguridad de que las cartas han sido repartidas azarosamente, y permite verificar que cada jugador haya recibido las cartas que decide jugar. Durante la especificaión del protocolo, se usará A para referirse al servidor, y B para referirse al cliente.

2.1. El reparto

2.1.1. Protocolo de reparto de cartas

El siguiente es el protocolo que da como resultado tres cartas para cada jugador elegidas de forma azaroza.

1. B le pide conexión a A

```
------------
|  "INIT"  |
------------
4 bytes
```

2. A genera una clave simétrica k (por ejemplo, con AES), encripta las 40 cartas $M_1...M_40$ con k y las envía a B (k sirve para asegurar que el mazo enviado por A en este paso es válido).

```
-----------------------------------------------
|   k(M1)   |   k(M2)   |   ...   |   k(M40)   |
-----------------------------------------------
  160 bits    160 bits               160 bits
```

3. B genera un p primo grande y genera e_b^1, d_b^1 (utilizadas para asegurar un reparto justo) y e_b^2, d_b^2 (utilizadas para asegurar que se jueguen las cartas repartidas) tal que

$$e_b^1 * d_b^1 = 1[mod\ p-1] = e_b^2 * d_b^2$$

Envía a A el p y las cartas ya encriptadas con k encriptadas a su vez con e_b^1 (usando RSA)

$$e_b^1(k(M_i)) = k(M_i)^{e_b^1}$$

```
-------------------------------------------------------------------------
|   p   |   e1b(k(M1))   |   e1b(k(M2))   |   ...   |   e1b(k(M40))   |
-------------------------------------------------------------------------
  1024 b      1024 b           1024 b          1024 b        1024 b
```

4. A usa p para generarse sus propias claves

$$e_a^1 * d_a^1 = 1[mod\ p-1] = e_a^2 * d_a^2$$

Elige al azar 3 cartas de las enviadas por B (B_1, B_2, B_3) y las firma con e_a^2. Envía cada carta como una tupla

$$< e_b^1(k(Bi)), e_a^2(e_b^1(k(Bi))) >=< k(Bi)^{e_b^1}[mod\ p], k(Bi)^(e_b^1 * e_a^2)[mod\ p] >$$

A su vez, repite el paso anterior realizado por B enviándole a este el resto de las cartas (Ri) encriptadas con su clave:

$$e_a^1(e_b^1(k(Ri))) = k(Ri)^(e_b^1 * e_a^1)[mod\ p]$$

```
-------------------------------------------------------------------------
|   e1b(k(B1))   |   e1b(k(B2))   |   e1b(k(B3))   |   e2a(e1b(k(B1)))   |
-------------------------------------------------------------------------
     1024 b           1024 b           1024 b              1024 b

-----------------------------------------
   e2a(e1b(k(B2)))   |   e2a(e1b(k(B3)))   |
-----------------------------------------
       1024 b                1024 b

-------------------------------------------------------------------------------
|    e1a(e1b(k(R1)))   |   e1a(e1b(k(R2)))   |   ...   |   e1a(e1b(k(R37)))  |
-------------------------------------------------------------------------------
        1024 b               1024 b                        1024 b
```

5. B recibe sus cartas (las tuplas) y les aplica la desencripción de e_b^1 con d_b^1:

$$< d_b^1(k(B_i)^{e_b^1}[mod\ p]), d_b^1(k(B_i)^{e_b^1 * e_a^2}[mod\ p]) >=$$
$$< k(B_i)^{e_b^1 * d_b^1}[mod\ p], k(B_i)^{e_b^1 * e_a^2 * d_b^1}[mod\ p] >=$$
$$< k(B_i), k(B_i)^{e_a^2}[mod\ p] >$$

A su vez, elige 3 cartas al azar (A_1, A_2, A_3) del resto (las R_i) y les aplica también d_b^1:

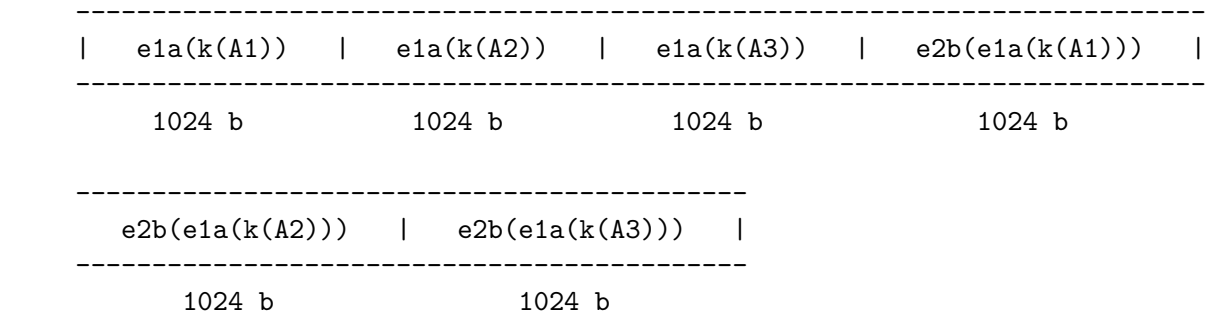
$$d_b^1(k(A_i)^{e_b^1 * e_a^1}[mod\ p]) = k(A_i)^{e_b^1 * e_a^1 * d_b^1}[mod\ p] = k(A_i)^{e_a^1}[mod\ p]$$

Para completar la mano de A, debe completar las tuplas con su firma:

$$e_b^2(k(A_i)^{e_a^1}[mod\ p]) = k(A_i)^{e_a^1 * e_b^2}[mod\ p]$$

y se las envía a A:

$$< k(A_i)^{e_a^1}[mod\ p], k(A_i)^{e_a^1 * e_b^2}[mod\ p] >$$



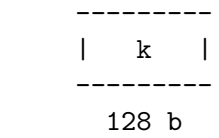
6. A recibe sus cartas y les aplica la desencripción de e_a^1 con d_a^1:

$$< d_a^1(k(A_i)^{e_a^1}[mod\ p]), d_a^1(k(A_i)^{e_a^1 * e_b^2}[mod\ p]) >=$$
$$< k(A_i)^{e_a^1 * d_a^1}[mod\ p], k(A_i)^{e_a^1 * e_b^2 * d_a^1}[mod\ p] >=$$
$$< k(A_i), k(A_i)^{e_b^2}[mod\ p] >$$

A utiliza k para ver las cartas que le tocaron, su mano queda

$$< A_i, k(A_i)^{e_b^2}[mod\ p] >$$

Por último, envia k a B



7. B recibe k, con lo que la utiliza para ver los M_i que le mando A en el paso 2 (poseia $k(M_i)$) y verificar que le mandó un mazo valido. También desencripta su mano para ver las cartas que le tocaron:

$$< B_i, k(B_i)^{e_a^2}[mod\ p] >$$

2.1.2. Pre-inicio del juego

Este es final del inicio del juego. Se setean datos para ser usados durante el resto de la mano.

1. Ahora que ambos tienen sus manos, B se genera un par de claves RSA comunes y corrientes

   $$(e_b^3, n_b), (d_b^3, n_b)$$

   y envía la pública (e_b^3, n_b) a A. Usará d_b^3 sólo para firmar sus acciones.

   A su vez, genera un paquete con el timestamp inicial decretando esta como la hora de inicio de juego. Lo envía firmado con d_b^3.

   ```
   ------------------------------------
   |   e3b   |   nb   |   d3b(ts_i)   |
   ------------------------------------
     2048 b    2048 b      2048 b
   ```

2. A lee con la clave pública de B lo firmado por él y verifica que el timestamp es válido. Se genera también sus pares de claves RSA

   $$(e_a^3, n_a), (d_a^3, n_a)$$

   y envía su aceptación (firmada con d_a^3) de la hora de inicio.

   ```
   --------------------
   |   e3a   |   na   |
   --------------------
     2048 b    2048 b
   ```

3. Al recibir B la verificación, puede empezar a jugar (notar que es mano).

2.2. Transcurso del juego

Durante el transcurso del juego se deberán cumplir las siguientes reglas:

- Es mano el que pidió conexion.
- Supongo que B quiere jugar B_2. Entonces debe enviar la firma que posee:

  $$k(B_2)^{e_a^2}[mod\ p]$$

  Cuando A lo recibe, le aplica su desencripción:

  $$d_a^2(k(B_2)^{e_a^2}[mod\ p]) = k(B_2)^{e_a^2 * d_a^2}[mod\ p] = k(B_2)$$

  Y con aplicar k, obtiene B_2.

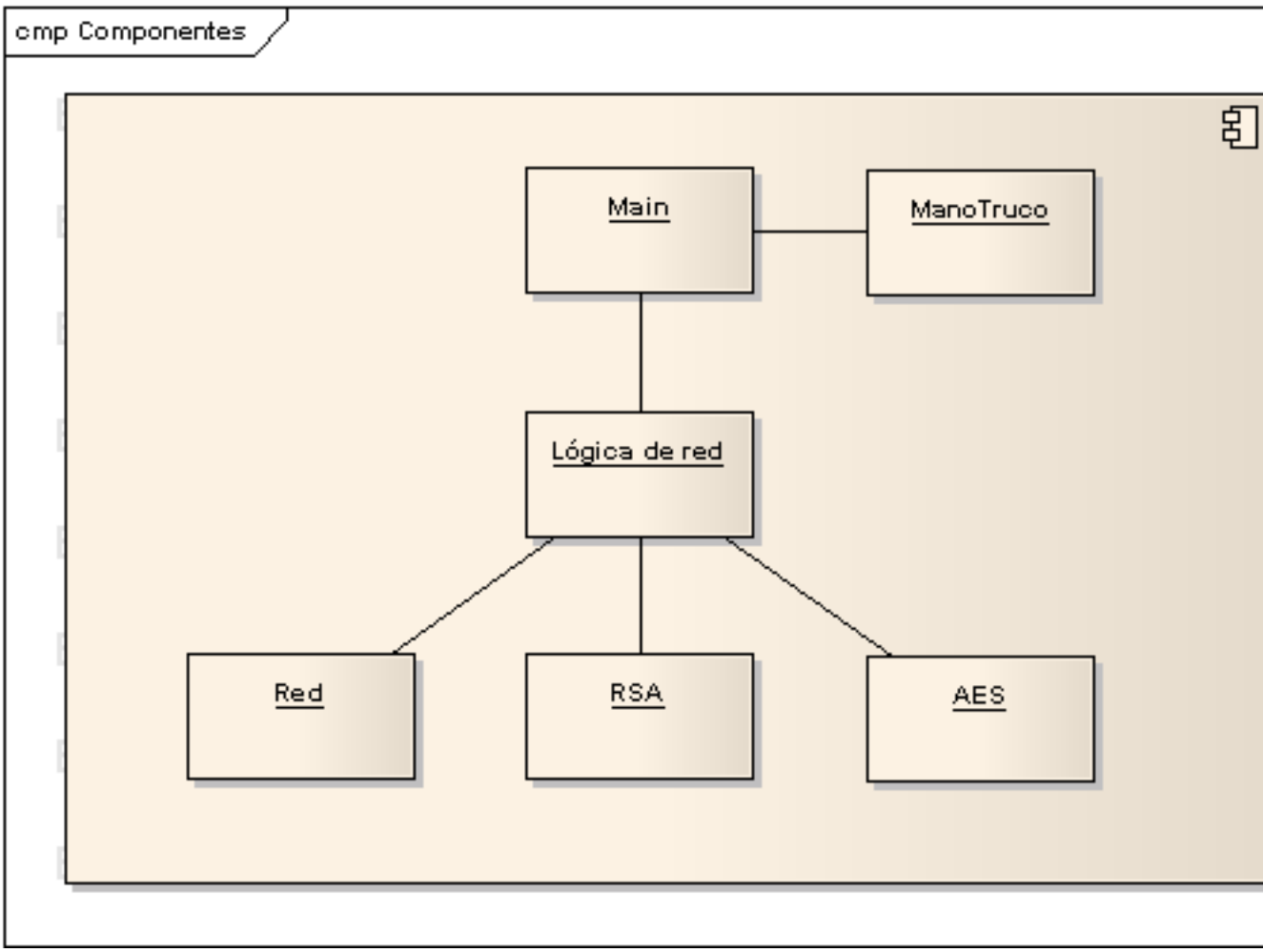
- Cada vez que se canta o se juega una carta, se debe adjuntar un timestamp del momento del canto/juego. Luego, el paquete debe ser firmado (con d_a^3 / d_b^3 según corresponda) para evitar el repudio del emisor más adelante ("...no no, yo no te cante, entendiste mal...") y la reutilización del canto como prueba falsa más adelante ("...sí sí, vos cantaste truco; hace 5 horas, pero cantaste, mirá...").

- El que finalize el juego (el que se vaya al mazo, el que mate la última carta del contrincante, el que gane en el falta envido) debe además enviar un timestamp firmado, marcándolo como el final oficial del juego. El oponente, aunque esté caliente, debe confirmarle si la hora es válida.

3. Consideraciones

Se decidió dejar los aspectos asincrónicos del juego del truco fuera del alcance del trabajo, ya que el foco está en la parte criptográfica del mismo. Por lo tanto, se implementó una simplificación del juego en la cual los turnos de juego están bien definidos, y cada jugador debe esperar su turno para poder jugar una carta o realizar algún canto. Esta simplificación no trae grandes inconvenientes a la hora de jugar, pero simplifica notablemente la implementación del juego, ya que permite prescindir del uso de threads, evitando cualquier posible condición de carrera (por ejemplo si un jugador canta ẗrucop̈ero el otro tambin cantra "truco.ªntes de recibir el canto del primero). De esta manera, el jugador que debe bajar una carta es el que tiene el turno. Si en lugar de jugar una carta, decide hacer un canto, cede su turno momentáneamente y sólo hasta que el contrincante conteste el canto. Una vez realizados los intercambios de turno requeridos por el canto realizado, el turno vuelve al jugador que debía bajar una carta.

Se decidió dividir el sistema en distintos módulos, cada uno con su responsabilidad bien definida. Toda la lógica necesaria para poder jugar al Truco y manejar los turnos se encuentra en el módulo ManoTruco. El módulo red se encarga de mantener una conexión TCP entre los jugadores, mientras que el módulo Lógica de red es el que implementa el protocolo diseñado, utilizando los módulos RSA y AES según corresponda, y brindando una interfaz que permita al resto del sistema enviar y recibir jugadas de cartas y cantos. El módulo main es el que implementa la interfaz con el usuario, y usando una instancia de ManoTruco y una de Logica de red se encarga coordinar y llevar adelante la mano de truco. En el siguiente diagrama se muestran los módulos mencionados y cómo se relacionan entre sí.


cmp Componentes
Main
ManoTruco
Lógica de red
Red
RSA
AES

Adi Shamir, Ronald L. Rivest, Leonard M. Adleman. Mental Poker. David A. Klarner Ed. Th

http://usuarios.arnet.com.ar/leo890/truco.htm