Escuela Superior Técnica
Gral. Div. Manuel N. Savio

Posgrado de Especialización en Criptografía y Seguridad Teleinformática

PROTOCOLOS CRIPTOGRÁFICOS ESPECIALES

Alumno: Lic. Ariel Maiorano

Profesor tutor: Dr. Pedro Hecht

Abril de 2012

Índice

1. Prefacio

El presente trabajo, de carácter descriptivo, tiene como objetivo el describir sucintamente y ejemplificar una serie de protocolos criptográficos. El texto tratará exclusivamente acerca de algunos de estos "protocolos criptográficos especiales", dando por conocidos a los principales, más importantes o más comúnmente implementados. De esto se desprende que aquí no serán abordados protocolos para el envío de mensajes cifrados, o de intercambio de llaves; para tratar otros que, por ejemplo, permiten certificar el fechado de un mensaje de manera segura, o jugar un "cara o seca" a distancia. Tal es la motivación principal del trabajo.

Concretamente, se intentará explicar y ejemplificar un conjunto de protocolos ideados para aplicarse con un fin último distinto, o al menos no exclusivamente, al de la obtención de confidencialidad, control de integridad y/o autenticación, aunque en muchos casos estos otros protocolos a tratar se valgan de, estén basados en, o resulten ser una variación de algunos de los protocolos y/o primitivas criptográficas principales.

Lo referente a la descripción o explicación de los protocolos corresponderá, por cada apartado, a un breve resumen describiendo sus generalidades y a una tabla que, con la ayuda de los típicos personajes Alice, Bob, y otros, enumerará paso a paso los procedimientos o algoritmos de cada parte. En cuanto a la ejemplificación práctica, se desarrollarán dos programas a tal fin, únicamente para la realización de pruebas, elementales y de carácter demostrativo, pero que permitirán una exposición concreta y práctica de algunos de los protocolos tratados en el trabajo.

2. Antecedentes

Como justificación principal puede citarse a uno de los textos guía o rectores de este trabajo, el libro *Applied Cryptography* [2], de Bruce Schneier, cuando en uno de sus capítulos introductorios hace referencia a la diversidad de problemas a los la criptografía podría dar solución:

"[...] cryptographic protocols: what people can do with cryptography. The protocols range from the simple (sending encrypted messages from one person to another) to the complex (flipping a coin over the telephone) to the esoteric (secure and anonymous digital money exchange). Some of these protocols are obvious; others are almost amazing. Cryptography can solve a lot of problems that most people never realized it could."

En el mismo sentido, al comienzo del capítulo acerca de protocolos, Shafi Goldwasser y Mihir Bellare, en sus *Lecture Notes on Cryptography* [4], del curso dictado en el MIT, explican:

"One of the major contributions of modern cryptography has been the development of advanced protocols. These protocols enable users to electronically solve many real world problems, play games, and accomplish all kinds of intriguing and very general distributed tasks. Amongst these are zero-knowledge proofs, secure distributed computing, and voting protocols [...]"

Queda de esta manera entonces explicitado que existe una variedad protocolos criptográficos aplicables en diferentes áreas, pero que, en palabras del primer autor citado –discúlpese la traducción- se trata de problemas que "la mayoría de las personas desconoce que podrían resolverse utilizando criptografía".

Cabe aclarar por último que el concepto de "especial" se entiende en este trabajo, al menos en el título, en un sentido amplio o abarcativo. Por ejemplo, se describirá la técnica criptográfica "visual", ideada por Moni Naor y Adi Shamir [10], donde es esta técnica o esquema ("*cryptographic scheme*" en el *paper* original) lo distinto o diferente a las maneras más comunes, aunque se aplique en última instancia para el cifrado de información, y el protocolo propiamente dicho (procedimientos de las partes –cifrar mediante llave o *pad* secreto, enviar texto cifrado vía canal inseguro, etc.-) no resulte muy diferente al de un OTP (*One Time Pad*).

Para cerrar estos comentarios en respecto a las justificaciones o antecedentes, refiriendo a la vez a otra fuente bibliográfica muy utilizada en el desarrollo de este trabajo, podría citarse también la explicación dada en el artículo principal de criptografía en Wikipedia [11], donde, en relación a la criptografía en general, se explica que se la entendió, antes de su "era moderna", como sinónimo de cifrado, esto es, la conversión de la información a partir de un estado legible sin sentido aparente. El remitente de un mensaje cifrado acordaba la técnica de decodificación necesaria para recuperar la información original sólo con determinados destinatarios, lo que impedía a otras personas la decodificación. Sin embargo, como se explicita en el artículo, es desde la Primera Guerra Mundial y el advenimiento de las computadoras que los "métodos" criptográficos se han vuelto cada vez más complejos, y su aplicación más generalizada.

3. Desarrollo

De acuerdo a los lineamientos establecidos en el Reglamento detallado en el Anexo 2 del Acta Nro. 30 del comité académico para la especialización, y tratándose éste de un Trabajo Final Integrador de carácter descriptivo, el desarrollo, o parte principal del mismo, se concentra en la descripción de los protocolos criptográficos enumerados en el índice.

Se explicarán haciendo foco en los procedimientos que involucran a las partes, luego de una breve descripción general del protocolo en cada caso, que intentará exponer también ejemplos de las situaciones de aplicación posibles, y/o de problemas típicos a los que daría solución.

Complementando lo anterior, y como lo que podría considerarse la segunda parte principal de este trabajo, como se adelantó en el prefacio, se desarrollarán dos programas de pruebas, o de demostración. Corresponde aclarar explícitamente que no se trata de programas finales o de implementación y uso directo para otros fines más allá de la ejemplificación; fueron ideados y desarrollados únicamente como soporte práctico al presente texto en la descripción de los protocolos.

El primero de los programas es una aplicación de escritorio Java, de ventanas (Swing), que reúne la posibilidad de “hacer de” o “actuar como” Alice, Bob, etc., para un subconjunto de los protocolos descriptos en el trabajo. A la vez, siendo el protocolo de *fair coin flip over the phone*, o cara o seca justo vía telefónica, uno de los temas motivadores del trabajo, se implementó una versión para ser utilizada justamente mediante el teléfono, aunque este caso se trata de una aplicación para teléfonos con sistema operativo android, y la comunicación se realiza vía mensajes de texto.

En los casos en que un protocolo determinado haya sido implementado en alguno de los programas para ejemplificar prácticamente los pasos que corresponden a cada parte, se hará referencia en este texto a las capturas de pantallas correspondientes, a ser presentadas en el apéndice, luego de anteponer allí mismo también las instrucciones generales para el uso y la compilación de los programas, que serán adjuntados –binario y código fuente- en un disco CD-ROM.

3.1. Secret Splitting / Sharing (Secreto dividido / compartido)

Como explican Menezes, Oorschot y Vanstone [1], la motivación original de este mecanismo de secreto compartido fue la de proteger copias de resguardo o *backups* de llaves criptográficas. Por supuesto también se menciona que proveen la manera de distribuir "confianza", o proveer un control compartido de actividades críticas, como podrían ser: la firma de cheques corporativos, apertura de bóvedas bancarias, etc.

Básicamente, existe un secreto, que se divide en partes, y cada una de estas partes es otorgada a diferentes usuarios, de manera tal que un subconjunto determinado de estos usuarios (utilizando sus partes-clave por supuesto) pueden, juntos, reconstruir el secreto. Cabe destacar que la cantidad de usuarios necesarios para reconstruir el secreto original puede ser menor a la cantidad total de partes distribuidas, aunque Schneier [2], por su parte hace una diferenciación entre *Secret Splitting* y *Secret Sharing*, considerando al primero para los casos donde todas las partes son necesarias para recuperar el secreto, y esto es imposible sin alguna de ellas o sin el proveedor (de confianza) del reparto del secreto. Para la segunda categorización de Schneier, se da el ejemplo de los tres de cinco coroneles necesarios para el disparo de misiles.

Al margen de lo antedicho, ambos autores citados llaman *Threshold scheme* a la alternativa más flexible, por identificarla de algún modo, y refieren el mecanismo de ecuaciones polinómicas en un campo finito de Adi Shamir. Concretamente, ya que se requieren "t" puntos para definir un polinomio de grado t-1, el conocedor (en quien se confía) del secreto "S", para distribuirlo entre "n" participantes de manera que sólo t de ellos puedan recuperarlo o reconstruirlo, generará un polinomio con coeficientes aleatorios (en Zp, p primo mayor al máximo entre S y n), salvo para el término independiente, que será S, y enviar a cada uno un punto (x, f(x)), con lo que podrán recuperar el secreto utilizando interpolación.

Éste será el protocolo descripto a continuación, aunque puede consultarse más información en otros textos de la bibliografía como [3] y [4], y en Wikipedia: [13] y [14]; donde se explica por ejemplo que la técnica fue "inventada" independientemente también por George Blakley en el año 1979, se

encuentra una implementación reducida del algoritmo de Adi Shamir en lenguaje de programación JavaScript [14], y además de lo ejemplos de aplicación como ser el registro o almacenamiento seguro de llaves de cifrado, códigos de misiles y cuentas bancarias numeradas, se explica la raíz de la problemática comparativamente para con los métodos más comunes: los métodos tradicionales para el cifrado serían podrían no ser los más adecuados para lograr simultáneamente un alto nivel de confidencialidad y fiabilidad. Esto es justificado considerando que a la hora del almacenamiento o registro de la llave o clave de cifrado, uno debe elegir entre mantener una única copia de esta llave, en una única ubicación, para el máximo secreto, o mantener varias copias de la llave en diferentes lugares para una mayor fiabilidad. Por un lado, el aumento de la fiabilidad de la clave mediante el almacenamiento de varias copias disminuye la confidencialidad porque implicaría la creación de vectores de ataque adicionales, hay más oportunidades en ese caso para que una copia de la llave se haga conocida por un potencial intruso o atacante. Aquí es entonces cuando se hace evidente, como solución a lo anterior, este mecanismo, el de secreto compartido, que permitiría alcanzar niveles confidencialidad y fiabilidad arbitrariamente altos.

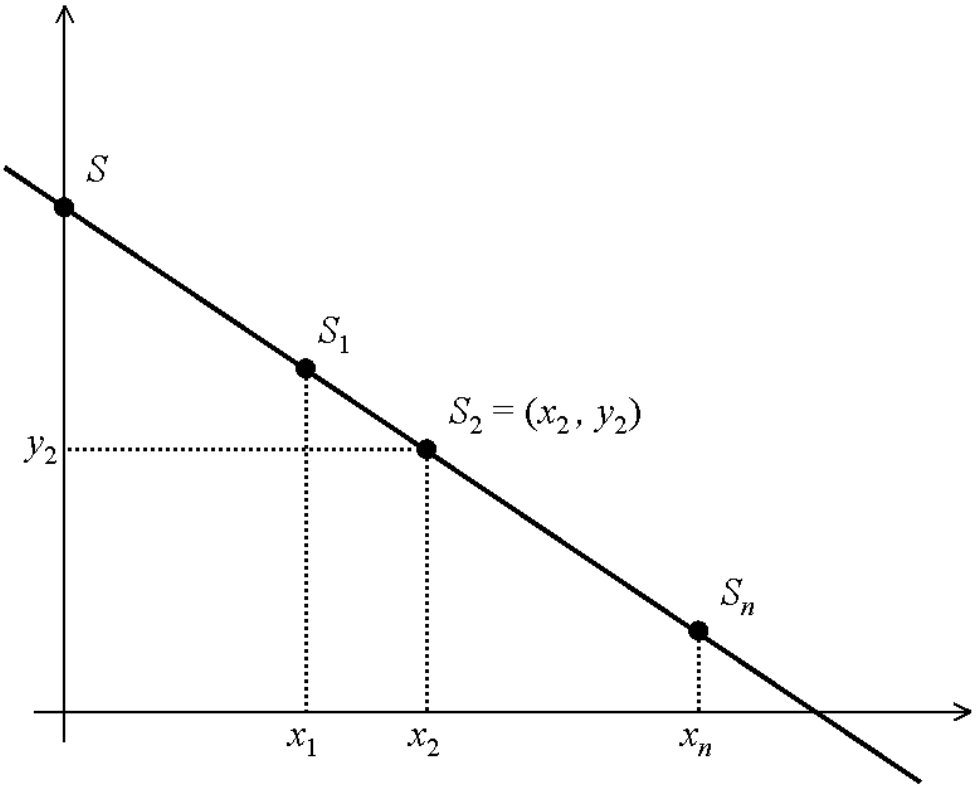
Como se ha adelantado que se haría para todos los protocolos a tratar en este trabajo, se describe en la tabla que sigue el protocolo de acuerdo a los procesos o procedimientos correspondientes a cada parte; téngase en cuenta que “Trent” se considera usuario (proveedor) de confianza, todos confían en Trent; y que Carol es el nombre de otro personaje, sin características especiales, se trataría de “otro usuario más”, como Alice y Bob.

Trent	Alice	Bob	Carol
1. Conoce el secreto S > 0 (ejemplo = 12). 2. Repartirá el secreto entre n usuarios (ejemplo = 3). 3. Establece que el secreto podrá reconstruirse entre t	9. Recibe el par 1,3. 12. Decide reconstruir con Bob. 14. * Dado que este ejemplo	10. Recibe el par 2,7. 13. Decide reconstruir con Alice.	11. Recibe el par 3,11.

usuarios (ejemplo = 2). 4. Elige un primo p > max(S, n) (ejemplo = 13). 5. Establece el coeficiente de la ecuación a0 = S (ejemplo = 12). 6. Elige los siguientes (t - 1) coeficientes independientes aX aleatoriamente en Zp (ejemplo: a1=4). 7. Calcula Sx = f(x) mod p para cada parte, o cada n (ejemplo: f(x)=4x+12, entonces, para Alice, o n=1: f(1)=4+12mod13= 3, para Bob: f(2)=8+12mod13=7, y para Carol: f(3)=12+12mod13=11. 8. Envía a cada parte el par (x, f(x)); x es público.	corresponde al caso especial de t = 2 (ver nota al pie de tabla), teniendo dos puntos en la recta, calculamos la pendiente m = [(7-3)/(2-1)] = 4, si 3 = 4(1) + S, entonces S = -1, mod p = 12 = S.		

* Nótese: como se describe en la sección que explica este esquema en el sitio de los *RSA Laboratories* (http://www.rsa.com/rsalabs/node.asp?id=2259), en el caso especial cuando "t" = 2 (es decir, sólo el acuerdo entre dos partes, o dos participantes con sus llaves "parciales", son necesarios para reconstruir el secreto), el polinomio es una línea y el secreto es el punto en el cual la línea corta al eje y. Concretamente, este punto sería el punto (0, f(0)) = (0, a0). Cada parte del secreto compartido correspondería a un punto en la línea. Son necesarios dos puntos para reconstruir el secreto porque con

un único punto, la línea podría ser cualquiera que pase sobre ese punto, por lo tanto el secreto podría ser cualquier punto del eje y.



3.2. Timestamping Services (Servicios de fechado)

Se trata en este caso de asegurar el fechado de un documento o mensaje. Existen muchos casos en los cuales sería necesario certificar que un documento existió en una fecha determinada. Estos protocolos de fechado permitirían asegurarlo. El ejemplo típico corresponde al control de derechos de copia y patentes, como es descripto por Schneier [2]. Existen diversas alternativas descriptas en el texto citado (*Arbitrated Solution, Improved Arbitrated Solution, Linking Protocol* y *Distributed Protocol*), la descripción que sigue está basada en la segunda alternativa, pero el texto de Menezes, Oorschot y Vanstone [1], bajo *Trusted timestamping service*, se describe también a algunas de las citadas y finalmente, paso a paso, el procedimiento equiparable a la tercera alternativa de Schneier.

Resumidamente, sin considerar el encadenamiento o el manejo de "árboles" de fechado, el protocolo se explicaría de la siguiente manera: el servicio de fechado de confianza (parte o usuario Trent en estos ejemplos) generará un recibo fechado para Alice cuando ella le presente un documento a fechar. Esto es entonces lo que le permitirá a Alice certificar, ante Bob por ejemplo, en tanto él también confíe en Trent, que el documento existió en la fecha del recibo. Trent, básicamente, agregó un *timestamp* o fechado al documento de Alice, firmó el resultado (documento + fechado), y devolvió esto, que consideraríamos el recibo, a Alice. Cabe aclarar que (siendo el uso más común –y el que será descripto en la tabla a continuación-), Alice podría haber enviado a Trent sólo el resultado de una función de *hash* aplicada al documento, lo que le hubiera permitido mantener en secreto el documento, al menos hasta el momento de necesitar certificar su fecha ante Bob.

Puede consultarse más información, en las fuentes citadas, y, específicamente en lo relativo a la estandarización de diferentes protocolos, en Wikipedia [15]. También se encontrará allí algo de información histórica; por ejemplo, se explica que la idea de la seguridad del fechado de información tiene siglos de antigüedad. Cuando Robert Hooke descubrió la ley de Hooke en 1660, él no quería publicarla inmediatamente, pero a la vez quería ser capaz de reclamar la prioridad de su descubrimiento. Así es que publicó el anagrama ceiiinosssttuv y más tarde la "traducción" *ut tensio sic vis* (que en latín significaría "como es la extensión, es la fuerza"). Del mismo modo, por ejemplo, Galileo publicó por primera vez su descubrimiento de las fases de Venus en forma de anagrama. En la actualidad se ejemplificó, en una versión más simple respecto a las técnicas descriptas en este

apartado, el caso de una organización de investigación industrial que más tarde puede necesitar probar, para fines de patentamiento, que realizaron un descubrimiento particular, en una fecha particular; ya que los medios magnéticos pueden alterarse fácilmente, esto puede no ser un problema trivial. Una posible solución es que un investigador calcule y registre un *hash* criptográfico del archivo de datos correspondiente. En el futuro, cuando necesitara comprobar la versión de este archivo de datos recuperado, sea por ejemplo desde una cinta de backup, el *hash* podría ser nuevamente calculado y se compararía contra el valor registrado, y seguramente también publicado, anteriormente.

Alice	Trent	Bob
1. Considerando al documento d, y la función de hash f(x), calcula y envía a Trent f(d). 3. Recibe t y r de parte de Trent. 4. Un tiempo más adelante, desea certificar con Bob la fecha t de del documento d, por lo que le envía ambas cosas, junto a su recibo r.	2. Recibe f(d), a lo que concatena la fecha actual t, realiza otro hash f(f(d)+t) para firmar el resultado y devolverle t y su firma como recibo r a Alice. .	5. Bob recibe d, t y r, computa f(d), f(f(d)+t) y verifica la firma de Trent.

3.3. Bit Commitment (Compromiso de bit)

Mediante este protocolo se lograría asegurar o comprometer (*to commit*) el valor (mensaje, documento, etc.) sin revelarlo sino hasta cuando sea disponga realizar la verificación.

El ejemplo más comúnmente citado es el de una oferta económica, como lo explican Goldwasser y Bellare [4] al tratar el tema: Alice quiere comprometer cierto valor, de manera que no pueda luego echarse atrás, pero no desea que Bob conozca, al menos en esta instancia, cuál es este valor. Alice arma una "caja fuerte electrónica", la usa para guardar el valor de la oferta, ella retiene su llave llave, pero envía la caja a Bob. Este paso representaría el compromiso. Más adelante, imaginemos, cuando el plazo para la presentación de ofertas venciese, Alice le enviará la llave, y Bob podrá enterarse del valor ofertado por ella, cuando envió la caja.

Schneier [2] propone otro ejemplo al respecto, el de un corredor de bolsa, Alice, y de un inversor, Bob. Bob le pregunta a Alice qué acciones compraría, ella responde que si le dijese que acciones compraría, él no necesitaría de Alice, entonces ofrece decirle cuáles acciones compró anteriormente, lo que Bob declina, porque no puede estar cierto de que Alice sea honesta y no elija acciones que resultaron favorecidas.

Según ambos autores, el problema podría solucionarse de diversas maneras (mediante algoritmos de cifrado simétrico o generadores seguros de números pseudo-aleatorios, entre otros), pero la forma que se describirá a continuación corresponde a la que se vale de funciones *hashing*.

Alice genera dos cadenas de bits aleatorias, R1 y R2, y agrega el bit (o conjunto de bits) que está dispuesta a comprometer, realiza un *hash* sobre esto y se lo envía a Bob, junto con uno de los valores aleatorios; al llegar la hora de revelar el bit (o conjunto de bits) comprometidos, Alice se lo enviará directamente a Bob, junto con los dos números aleatorios, para que él compruebe que uno de los valores aleatorios se condice con lo recibido anteriormente, y que el *hash* también se corresponde.

Se trata en este caso también de una generalización o base común de otros protocolos, por ejemplo, del que se describirá en el apartado siguiente, *Fair Coin Flips* o "cara o seca justo", y de las pruebas de conocimiento cero, a describirse también más adelante, o de una variante específica de secreto compartido, ya tratado. Puede consultarse Wikipedia para más información [16]. En ese artículo se demuestra la idea de este mecanismo y sus posibles aplicaciones con un ejemplo simple: supóngase

que dos personas quieren jugar “piedra-papel-o-tijera” por correo electrónico. Es evidente que el problema en esta situación es que un jugador puede simplemente esperar hasta recibir el correo electrónico de otro jugador que diga, por ejemplo, “piedra”, y recién después responder rápidamente diciendo “papel”, ganando injustamente. Este es entonces un problema que puede ser resuelto mediante el uso de mecanismos de *bit commitment*. Al comienzo del juego, cada jugador se compromete enviando el resultado del mecanismo aplicado a su elección de “piedra”, “papel” o “tijera”. Después de esto, cada uno revela la elección a la que se comprometieron anteriormente, de manera que no es posible hacer trampa.

Alice	Bob
1. Genera R1 y R2 aleatoriamente. 2. Mediante una función de *hash* f(x), calcula f(R1 + R2 + v), siendo v el valor, bit o cadena de bits a comprometer. 3. Envía el resultado de este cálculo y el valor de R1 a Bob. 5. Llegado el momento de revelar el compromiso, envía R1, R2 y v a Bob para que realice las verificaciones.	4. Recibe f(R1 + R2 + v) y R1. 6. Recibe R1, R2 y v, para comprobar que R1 es igual al R1 anterior, y que, al calcular f(R1 + R2 + v), también resulte igual a lo recibido anteriormente.

3.4. Fair Coin Flip (Cara o seca justo)

Este protocolo fue propuesto por Manuel Blum en 1982, originalmente como "*coin flipping over the telephone*", o "cara o seca por teléfono", según Goldwasser y Bellare [4], y resuelto mediante la utilización de un protocolo de *bit-commitment*. Existen varias maneras entonces de resolver prácticamente este problema, pero el mecanismo que se describirá en este trabajo será el que se basa en la utilización de funciones de una vía (*hash*), donde la seguridad del protocolo depende de la seguridad de la función de una vía seleccionada, de acuerdo a cómo es descripto por Schneier [2].

En pocas palabras, Alice y Bob deben acordar qué función utilizarán, Alice generará un número aleatorio y enviará a Bob el resultado de la función aplicada a ese número, para que Bob adivine –y se lo comunique a Alice- si el número sería par o impar (esto es lo que equivaldría a un "cara o seca"), con lo que Alice podría devolverle el resultado, junto con el número original, y con este número Bob podrá también aplicar la función y confirmar si acertó o no.

Alice	Bob
1. Genera número aleatorio x. 2. Envía f(x) a Bob y le pide que adivine. 5. Recibe cara o seca de parte de Bob. 6. Comunica a Bob si adivinó, y le envía x para que calcule f(x) y compare el resultado con el valor recibido inicialmente.	3. Recibe f(x) de parte de Alice 4. Adivina si x es par o impar (cara o seca) y se lo comunica a Alice. 7. Recibe x para calcular f(x) y comprobar si adivinó correctamente.

3.5. Undeniable Digital Signatures (Firmas digitales innegables)

El primer protocolo de firma digital "especial" a tratar en el trabajo corresponde a un esquema que se diferencia del típico en que el protocolo impone que para la verificación de la firma se necesitará la cooperación (en palabras Menezes, Oorschot y Vanstone [1]) o consentimiento (Schneier, [2]) del firmante. Luego de comentar que el nombre de estas firmas debió haber sido *something like "nontransferable signatures"*, el segundo autor citado aclara que el nombre se debe al hecho de que si el firmante fuese obligado a reconocer o rechazar una firma, él no podría falsear el rechazo. Continúa explicando que la idea básica es simple: Primero Alice envía a Bob una firma; luego Bob genera un número aleatorio y se lo envía a Alice; Alice entonces realiza un cálculo utilizando el número aleatorio recibido y su llave privada, para enviar luego a Bob el resultado; Bob confirma esto último.

La tabla que se presentará a continuación describe el algoritmo Chaum-van Antwerpen de acuerdo a cómo está definido en [1], presuponiendo como "a" a la llave privada de Alice, y el conjunto ("p", "alfa", "y") como la llave pública, donde p es un primo aleatorio igual a 2q + 1 (q primo), alfa es un elemento aleatorio en Zp elevado a (p -1)/q, módulo p, e y corresponde a alfa elevado a la "a" (aleatorio también). "m" corresponde al mensaje, y Alice es quien firma, o usuario firmante, y Bob quien ha de verificarlo.

Puede consultarse más información en las fuentes de la bibliografía citada y en el artículo correspondiente en Wikipedia [17].

Alice	Bob
1. Calcula $s = m^a \bmod p$. 2. Envía firma original s y mensaje m a Bob. 7. Alice recibe z de parte de Bob y calcula $w = (z^a)^{-1} \bmod p$ y envía esto a Bob.	3. Recibe s y m de parte de Alice. 4. Obtiene la llave pública (p, alfa, y) de Alice. 5. Obtiene dos números aleatorios x1 y x2 menores a q - 1. 6. Calcula $z = s^{x1}y^{x2} \bmod p$ y se lo envía a Alice.

	8. Recibe w de parte de Alice y calcula $w'=m^{x1}alfa^{x2}$ mod p y comprueba que w = w', es decir, que la firma es válida.

3.6. Fail-Stop Digital Signatures (Firmas digitales fail-stop)

De acuerdo a Menezes, Oorschot y Vanstone [1], estas son firmas digitales que le permitirían a Alice probar que una supuesta firma suya es falsa (cuando efectivamente ella no haya realizado tal firma por supuesto). Esto se logra mostrando que el mecanismo en que se basa este esquema ha sido comprometido. La posibilidad de probar una falsificación de firma puede fallar con un probabilidad muy baja, pero es independiente del poder de procesamiento del eventual falsificador, por eso es que, por ejemplo, Schneier [2], se vale de la explicación en que "Eve" (otro personaje, además de Alice, Bob, y otros; pero usado para representar el papel de un *eavesdropper* -traducciones posibles serían: espía, fisgón o intruso-) tiene a su disposición un gran poder de almacenamiento, mucho mayor que el de Alice, que, en un esquema tradicional de firma digital, Eve utilizaría para romper la llave privada de Alice y así poder realizar falsificaciones de firmas. Esto es lo que las firmas *fail-stop* (una traducción podría ser "de detención ante falla") permitirían evitar.

La idea básica en este mecanismo, introducido por Birgit Pfitzmann y Michael Waidner, se basa en que muchas llaves privadas diferentes podrían "funcionar" con cada llave pública posible, cada llave privada generaría una firma diferente también, y Alice sólo conoce (y utiliza) una de estas llaves privadas posibles. De esta forma entonces, Eve, al intentar recuperar por fuerza bruta la llave privada de Alice, podrá eventualmente recuperar una llave privada válida, pero al haber tantas, será mucho más probable que sea una distinta a la de Alice. Cuando Eve intente falsificar una firma con la clave privada que recuperó, generará una firma diferente a la que fuera generada si Alice firmara; y esta sería la forma entonces en que Alice podría probar que una supuesta firma suya es falsa.

En Menezes, Oorschot y Vanstone [1] encontramos descripto el algoritmo van Heijst-Pedersen, basado en el problema del logaritmo discreto en Zp. Se transcribe ahora a una tabla a la manera elegida para el TFI.

Alice	Bob
1. Siendo m el mensaje, comprendido entre [0, q - 1], calcula $s_{1,m} = x_1 + my_1 \bmod q$ y $s_{2,m} = x_2 + my_2 \bmod q$. La firma de m a enviar a Bob será ($s_{1,m}$, $s_{2,m}$).	2. Recibe la firma de parte de Alice, obtiene sus llaves públicas (B_1, B_1, p, q, a, B). Calcula $v_1 = B_1B_2^m \bmod p$ y $v_2 = a^{s1,m}B^{s2,m} \bmod p$. Aceptará la firma sólo si $v_1 = v_2$.

3.7. Blind Signatures (Firmas ciegas)

Goldwasser y Bellare [4] abordan este tema planteándolo como una solución a un problema de anonimato. Lo explican mediante la analogía en la cual un cliente presenta a su banco un cheque a firmar, pero bajo un papel carbónico, y dentro de un sobre cerrado. Así entonces el banco firmará el sobre, y esa firma quedará estampada también en el cheque. El cliente entonces podrá abrir el sobre y entregar el cheque a una tercera persona, y ésta, al depositar el cheque, permitirá al banco comprobar la firma, pero no saber quién fue el cliente que se lo hizo firmar. Por supuesto, es evidente, el cliente podría hacer un cheque con un importe arbitrario sin que el banco se entere, o sufrir también lo que sería una suerte de *replay-attack*; los autores citados tratan el problema más adelante, pero por lo pronto supongamos, de momento, que se trata de notas crédito numeradas de manera segura por ejemplo, de valor prefijado, en lugar de cheques. De esta forma, el cliente podría pagar a través del banco sin revelarle que él ha sido quien entrego el cheque a una tercera persona en particular (se supone por supuesto que el banco ha firmado otras notas de crédito a otros clientes, que han sido entregadas a otras terceras personas, y así es que el banco no podría relacionarlos).

Schneier [2] ejemplifica el caso primero con un escribano que certifica un documento sin conocerlo en absoluto, pero luego sigue la descripción de casos en donde el firmante conoce partes, o diferentes documentos del total a firmar, lo que llamó "*the cut-and-choose technique*", junto a interesantes ejemplos de probabilidad relacionados al control de contrabando en un aeropuerto y comunicación entre agentes de una fuerza de seguridad. Lo que sigue sería, muy resumidamente, el mecanismo basado en RSA que se explica cómo proponen el tema Menezes, Oorschot y Vanstone [1]: se considera que Alice envía a Bob una porción de información que Bob firma y devuelve a Alice. A partir de esta firma, Alice puede calcular la firma de Bob en un mensaje previo "m" a su elección. Al finalizar el protocolo B desconoce tanto m como la firma asociada. El propósito es prevenir que Bob que observe m, para que luego no pueda asociar a Alice con el mensaje firmado. La tabla a continuación describe la metodología o protocolo de David Chaum (criptógrafo del que se hablará más adelante en el trabajo al tratar otros protocolos, principalmente relacionados a problemas de anonimato), según es expuesto también por los autores citados previamente. Aquí, Alice recibe una firma de Bob en un mensaje "ciego". Alice computará la firma de Bob en un mensaje "m"

seleccionado a priori por ella, de tal forma que $0 <= m <= n - 1$. Bob desconoce tanto m como la firma asociada a m.

Alice	Bob
1. Selecciona un número aleatorio, secreto, k, tal que $0 <= k <= n - 1$ y mcd(n,k) = 1. Calcula $m^* = mk^e \bmod n$ y se lo envía a Bob (e es parte de la llave pública de Bob). 3. Calcula $s = k^{-1}s^* \bmod n$, que resulta la firma de Bob de m.	2. Calcula $s^* = (m^*)^d \bmod n$ y se lo envía a Alice.

Más información puede consultarse en Wikipedia [18], donde puede encontrar por ejemplo también una referencia (link) al *paper* original de David Chaum.

3.8. Oblivious Transfer & Signatures (Transferencias y firmas "desconocidas")

Quien ha introducido este concepto, Joe Kilian, ha utilizado un ejemplo en el cuál Alice venderá a Bob un número (de 100 bits de longitud) "parcialmente", es decir, entregará el 50% de los bits, asegurando a la vez a Bob que el número, completo, es efectivamente uno de los factores que él necesita, pero por el que puede pagar sólo la mitad del dinero que Alice pretende por el número completo.

Schneier [2] continua describiendo este tipo de transferencia considerando un conjunto de mensajes enviados por una parte a otra, recibiendo esta otra parte sólo un subconjunto aleatorio del conjunto original, y de esta manera se lograría entonces.

Concretamente, Alice envía a Bob este conjunto o grupo de mensajes, Bob recibe un subconjunto de estos mensajes pero Alice desconoce cuáles ha recibido. Alice luego usa una prueba de conocimiento cero para asegurar a Bob que el conjunto que recibió forma parte de lo comprometido (50% vendido del número que Bob necesitaba).

El autor citado describe un protocolo que implementa lo anterior de acuerdo a los siguientes pasos:

Alice	Bob
1. Genera dos pares de llaves pública/privada y envía las dos públicas a Bob. 3. Recibe de parte de Bob la llave simétrica cifrada con una de sus dos llaves públicas, descifra una vez con cada una, uno de los resultados será la llave pero no puede determinar cuál. 4. Parte su mensaje (50% del número en el ejemplo) y envía a Bob los resultados de cifrar simétricamente, con los resultados (sólo uno es el correcto) del paso anterior	2. Selecciona una llave para un algoritmo de cifrado simétrico, recibe dos llaves públicas de parte de Alice, selecciona una de éstas para cifrar la llave seleccionada, y devuelve el resultado a Alice, sin informarle cuál de las dos llaves públicas utilizó para cifrar la llave. 5. Bob recibe (y puede interpretar) uno de los dos mensajes cifrado con la llave correcta.

como llaves, ambas partes. 8. Terminado le protocolo, Alice envía a Bob sus llaves privadas, y de esta forma prueba que no ha hecho trampa.	

Si bien se cuestiona la aplicación práctica de esto protocolo, también existen las llamadas *Oblivious Signatures* o firmas "desconocidas". Una forma o manera de implementar este mecanismo o protocolo sería, por ejemplo: una parte en este caso cuenta con diferentes mensajes; la otra parte selecciona uno de estos mensajes para firmarlo, de tal manera que la primera no sepa cuál mensaje firmó la segunda parte.

Puede consultarse más información en los apartados al respecto en , Oorschot y Vanstone [1], Goldwasser y Bellare [4] y en Wikipedia [19]; en este último artículo se explica el *Rabin's oblivious transfer protocol*, o protocolo para transferencias "deconocidas' de Rabin.

3.9. Simultaneous Contract Signing (Firmas de contrato simultáneas)

El caso al que puede aplicarse este mecanismo o esquema está explicado por Goldwasser y Bellare [4] de la siguiente manera: Alice y Bob quieren firmar un contrato; cada uno de ellos quiere hacerlo sólo si el otro también lo hace. Esto es, ninguno quiere quedar en la posición de ser el único firmante. Entonces, si Alice firma primero, estará preocupada porque no sabe si Bob firmará o no, y viceversa. Puede imaginarse la situación considerando dos contratos, uno donde Alice promete algo a Bob y otro en el cual Bob promete algo a Alice, y la transacción de estos contratos debe hacerse entre partes que no confían la una en la otra. Uno de los enfoques para solucionar este problema consiste en que Alice firme con la primera letra de su nombre y le envíe el contrato a Bob. El hará lo mismo, y lo enviará de vuelta. Así sucesivamente. Se asume que ambos nombres tienen la misma longitud. De esta manera se habría avanzado, pero por supuesto el problema ahora sería que Bob puede no firmar con su última letra. Sin embargo, lo que puede hacerse es que esta diferencia (una letra en el ejemplo simplificado) sea despreciable. Siguiendo en la línea del ejemplo, puede no considerarse una letra (por cada parte, cada vez), sino un unos pocos milímetros de una letra a la vez. Ninguna parte está nunca mucho "más allá" que la otra. Si en algún momento alguna parte se detiene, ambas estarán aproximadamente en el mismo punto. Electrónicamente ahora, dejando de lado el ejemplo aunque la idea por supuesto es la misma, lo que se transacciona son cadenas de bytes, firmas digitales del contrato. Alice y Bob lo firmaron parcialmente, en su mayor parte. Ahora continúan intercambiando estas cadenas pero de a un bit por vez. Como plantean los autores citados, existe un problema con este esquema. ¿Qué sucedería si una persona decide no enviar la firma sino una cadena de datos cualquiera (basura)? La otra parte no se enteraría sino hasta el final del procedimiento. Aquí es entonces donde de las transferencias *oblivious* ayudan a solucionar el problema.

Ahora en palabras de Schneier [2], el protocolo descripto en su texto bajo el apartado *"Simultaneous Contract Signing without an Arbitrator (Using Cryptography)",* o Firma de Contratos Simultanea sin un Arbitro (utilizando criptografía), se resumiría de la manera detallada en la siguiente Tabla. Téngase en cuenta que se utiliza el algoritmo de cifrado simétrico DES, pero podría utilizarse en cambio cualquier otro del mismo tipo.

Alice	Bob
1. Selecciona aleatoriamente 2n llaves DES agrupadas en pares. 2. Genera n pares de mensajes, Li y Ri: "Esta es la parte derecha (*Left*) /izquierda (**Right**) de mi firma en el índice i". Cada mensaje podría incluir una firma digital del contrato u un *timestamp*. El contrato se considerará firmado si la otra parte puede producir ambas mitades, Li y Ri, de un único par de firmas. 3. Cifra sus pares de mensajes con cada una de sus llaves DES; el mensaje de la derecha con la llave derecha del par y el mensaje de la izquierda con la llave izquierda del par. 5. Envía a Bob (y recibe de su parte) el conjunto de los 2n mensajes cifrados, indicando a cuales mitades de cuales pares corresponden. 7. Aquí comienza con Bob la transferencia *oblivious*, enviándose entre ellos cada uno de los pares de llaves. En este caso Alice enviará la llave utilizada para cifrar la parte derecha del mensaje o la llave con que cifró la parte izquierda, independientemente, por cada uno de los	4. Realiza las mismas operaciones que Alice en los pasos # 1, 2 y 3. 6. Envía a Alice el conjunto de los mensajes cifrados (ver paso # 5 de Alice). 8. Realiza la transferencia *oblivious* de misma manera en que Alice lo hace, y por supuesto, simultáneamente. 13. Realiza los mismos pasos que Alice (ver pasos # 9, 10, 11 y 12) y de la misma forma puede determinar que la otra parte no ha hecho trampa.

n pares. 9. Descifra las mitades de mensajes recibidos que puede descifrar, utilizando las llaves recibidas, y asegurándose de que los mensajes descifrados son válidos. 10. Envía (y recibe de parte de Bob) los primeros (o primer bloque) bits de todas (2n) las llaves DES. 11. Repite el paso anterior con el segundo bloque de bits, luego con el tercero, y así sucesivamente, hasta que todos los bits de todas las llaves DES hayan sido transferidos. 12. Intercambia con Bob las llaves privadas utilizadas en el paso # 7 y así cada parte verifica que la otra no ha hecho trampa.	

3.10. Mental Poker (Poker mental)

Se tratará aquí del mecanismo que permite a Alice y a Bob jugar póker entre ellos vía correo electrónico. Schneier [2] lo describe como una variante del "cara o seca" implementado con criptografía de llave pública. En este caso, en lugar de que Alice componga y cifre dos mensajes, uno para cuando decide "seca" y otro para "cara", ella compondrá 52 mensajes, M1, M2, M3, … M52; uno por cada carta. Bob seleccionará 5 mensajes al azar, los cifrará con su llave pública, y se los enviará de nuevo a Alice. Ella descifrará los mensajes y se los enviará de vuelta a Bob, quien descifrando, se enterará de qué cartas le han tocado. Lo que sigue es que Bob elija otros cinco mensajes al azar y se los envíe a Alice; ésas serán (luego de descifrar por supuesto) las cartas de ella. Durante el juego, pueden entregarse o distribuirse cartas adicionales repitiendo el mismo procedimiento. Al finalizar el juego, ambas partes revelan sus cartas y los pares de llaves utilizados y de esa manera cada uno podría cerciorarse de que ninguno ha hecho trampa.

En Goldwasser y Bellare [4] puede consultarse una definición más formal del esquema, paso a paso, detallando las fórmulas de que involucran los números primos y cálculos modulares correspondientes, para dos jugadores, además de un problema que podría permitir la extracción parcial de información desde los mensajes, pero lo que sigue será cómo el primer autor citado expone cómo sería el mecanismo en el caso de que sean tres los jugadores. De esta forma, básicamente, se extiende el protocolo anterior, y a tres o a más jugadores. También en este caso, el algoritmo criptográfico debe ser conmutativo. Se describe el procedimiento o protocolo en la siguiente tablita a la manera elegida para describir protocolos en este trabajo:

Alice	Bob	Carol
1. Genera un par de llaves (pública y privada) para una única "mano". No revela ninguna todavía. 4. Genera 52 mensajes,	2. Genera un par de llaves (pública y privada) para una única "mano". No revela ninguna todavía. 6. Recibe los 52 mensajes	3. Genera un par de llaves (pública y privada) para una única "mano". No revela ninguna todavía. 7. Recibe los 47 mensajes

uno por cada carta (cada mensaje podría ser o contener un número aleatorio para la verificación a posteriori). 5. Envía a Bob los resultados de cifrar con su llave pública los 52 mensajes. 5. Recibe de parte de Bob y de Carol los mensajes, los descifra con su llave privada y les devuelve el resultado a cada uno según corresponda. 12. Descifra lo enviado por Carol y de esta forma conoce sus cartas. 13. Al final de la "mano" se revelan las cartas y las llaves para que todas las partes puedan confirmar que nadie hizo trampa.	de parte de Alice. Elige 5 mensajes al azar, los cifra con su llave pública y envía esto a Alice. Envía los 47 mensajes restantes a Carol. 6. Recibe lo enviado por Alice, lo descifra, y así conoce sus cartas. .	de parte de Bob, selecciona 5 al alzar, los cifra con su llave pública y se los envía a Alice. 7. Recibe lo enviado por Alice, lo descifra, y así conoce sus cartas. 8. Selecciona aleatoriamente 5 mensajes entre los 42 restantes y se los envía a Alice.

3.11. One-Way Accumulators (Acumuladores de una vía)

En las palabras de Schneier [2], los acumuladores de una vía serían un mecanismo novedoso para lo solución a los problemas inherentes a la implementación de una lista de membresía. Se trataría de una suerte de función de una vía, pero con la particularidad de ser conmutativa. De esta forma, es posible obtener un *hash* de la lista o base de datos de miembros en cualquier orden, y obtener el mismo resultado. Sería posible además agregar nuevos miembros, recalcular o regenerar el *hash*, sin que se necesario considerar el orden.

El procedimiento de cada parte correspondería, detallado de manera general en este caso, a lo desripto en la siguiente tabla:

Alice	Bob
1. Calcula el acumulado de todos los miembros de la lista, salvo ella y guarda este valor junto con su nombre. 3. Intercambia con Bob (por ejemplo) los resultados obtenidos en los primeros pasos, y confirma que el nombre Bob adjunto a su acumulado (el de Bob) es igual a su nombre (Alice) adjunto a su propio acumulado, y así sabe que Bob es miembro.	2. Bob (como podría hacerlo cualquier otro miembro de la lista) realiza la misma operación que Alice en el paso #1. 4. Realiza la misma verificación que Alice en el paso 3, para confirmar también que Alice es miembro de la lista.

3.12. All-or-Nothing Disclosure of Secrets (Revelación de secretos "todo o nada")

En este caso Alice tendría una variedad de secretos a la venta. Bob quiere comprarle algún o algunos de estos secretos, pero no quiere que Alice se entere de cual. Ella por supuesto no venderá dos o más secretos al precio de uno.

Así es como explica Schneier [2] el problema para el cual la solución sería este protocolo. Se trata de la revelación a "todo a nada", también referenciado como ANDOS, por sus siglas en inglés *All-or-Nothing Disclosure of Secrets*, o revelación de secreto/s a todo o nada, como se intentó traducir en el título. Aquí tan pronto como Bob haya obtenido acceso a uno –o parte- de los secretos de Alice, habrá perdido la posibilidad de conocer algo acerca de los otros secretos.

Existen diferentes maneras, o diferentes protocolos, de lograr una implementación de este tipo. La que se describe a la manera del TFI en la tabla a continuación, asumiendo partes honestas, es un ejemplo de la fuente citada:

Alice	Bob
1. Cifra todos sus secretos con RSA y envía a Bob $Ci = S_1^e \bmod n$. 3. Envía a Bob $P' = C'^d \bmod n$.	2. Selecciona el secreto C_b, genera un número aleatorio r, y envía a Alice $C' = C_b r^e \bmod n$. 4. Calcula $S_b = P' r^{-1} \bmod n$.

3.13. Key Escrow (Custodia de claves)

Para Menezes, Oorschot y Vanstone [1], el objetivo de esta técnica es el de proveer el servicio de cifrado de "tráfico" de usuario (por ejemplo, voz o datos), en el cual las llaves de sesión utilizadas estarían disponibles o accesibles a terceras partes, debidamente autorizadas, en situaciones especiales ("accesos de emergencia"). De esta forma se permitiría a estas terceras partes, que monitorearon el "tráfico", la capacidad de descifrarlo. Según los autores, el interés público en estas técnicas creció de manera importante cuando agencias y fuerzas de seguridad promovieron su utilización para, legalmente, facilitar las intervenciones telefónicas para combatir actividades criminales. Destacan así mismo otros usos, como por ejemplo la recuperación de información cifrada una vez extraviadas las llaves originales, por una parte autorizada o legítima por supuesto.

Schneier [2] por su parte cita a Silvio Micali, cuando comentó que el monitoreo de líneas telefónicas es un método efectivo para llevar delincuentes frente a la justicia, además de prevenir la utilización de estos medios para fines ilegales. Explica que existe una preocupación lógica por la creciente utilización de criptografía de llave pública por parte de criminales y terroristas; por lo que plantea dos alternativas posibles: o se utilizan cripto-sistemas débiles, o se "resignan", a priori, las llaves privadas a una autoridad competente. Al margen de otras consideraciones políticas, y de que Micali recibió un millón de dólares de parte del gobierno de los EE.UU. por el uso de sus patentes, Schneier describe su "*fair cryptosystem*". Aquí la llave privada es dividida en partes y estas partes son distribuidas a diferentes autoridades. Como en "secreto compartido", las autoridades pueden reunirse y reconstruir la llave privada, sin embargo, las partes tienen la propiedad adicional de poder ser verificadas individualmente.

En la siguiente tabla se describe de forma general cómo trabajaría el protocolo (aquí Bob y Carol son las autoridades –dos en este ejemplo pero podrían ser más-, y Trent asumiría el rol de un KDC, o *Key Distribution Center*).

Alice	Bob	Carol	Trent
1. Genera su llave pública	2. Recibe las	3. Recibe las	4. Recibe de

y privada. Divide su llave privada en partes (unas públicas y otras privadas). Envía –cifradas- a cada una de las autoridades (Bob y Carol) las "partes" públicas de llave privada. Envía su llave pública a Trent.	partes que envió Alice y realiza la validación. Almacena de manera segura la parte privada y envía a Trent la pública.	partes que envió Alice y realiza la validación. Almacena de manera segura la parte privada y envía a Trent la pública.	parte de Alice su llave pública. Recibe de parte de las autoridades y en base a otro cálculo verificador, firma la llave pública de Alice y se la envía o la almacena.

Dada esta situación entonces, si un juez ordena un monitoreo, cada una de las autoridades (Bob y Carol) "resignaría" sus partes al KDC (Trent), y éste podría, recién entonces reconstruir la llave privada.

Más información, incluyendo un link a un memo enviado por la CIA a Al Gore en el año 1996, puede consultarse en Wikipedia[20].

3.14. Zero-Knowledge Proofs (Pruebas de "conocimiento cero")

Schneier [2] comienza el capitulo que describe este mecanismo con un cuento:

Alice: "Conozco la contraseña para el sistema de la Reserva Federal, los ingredientes de salsa secreta de McDonald's, y el contenido del cuarto volumen de Knuth"...

Bob: "No, no conocés todo eso."

Alice: "Sí, lo conozco."

Bob: "¡No!"

Alice: "¡Sí!"

Bob: "¡Probalo!"

Alice: "Está bien. Voy a contarte."Alice susurra en el oído de Bob.

Bob: "Qué interesante. Ahora que también lo sé, voy a contárselo al Washington Post."

Alice: "Oops".

De esta forma explica que, desafortunadamente, la forma usual que tendría Alice de probar algo a Bob sería contárselo, aunque lógicamente, de esta forma, Bob no sólo sabe que Alice conoce el secreto, sino también el secreto.

Una desventaja de los sistemas basados en contraseñas simples es la que permite que, cuando la contraseña se entrega (para realizar la prueba o la autorización) a otra parte (quien realizará la verificación), esta podrá "impersonar" o realizar una impostura, haciéndose pasar por la primera. Así lo explican Menezes, Oorschot y Vanstone [1], comentando después que los mecanismos de *challenge-response* (repregunta o desafío y respuesta) podrían mejorar esta situación, pero tendrían a sus vez sus propios inconvenientes. Por eso continúan explicando que los protocolos de conocimiento cero, *Zero-Knowledge*, o ZP, fueron diseñados para solucionar estos problemas. De esta manera se permite realizar la prueba (de parte de quien conocería la contraseña en los ejemplos) sin revelar información que le permita al verificador realizar una "demostración de conocimiento" equivalente a otras partes. En un sentido más general, estas pruebas permiten probar, valga la redundancia, la veracidad de una aseveración sin entregar información acerca de la aseveración en sí.

Más adelante, Menezes, Oorschot y Vanstone presentan un ejemplo de este tipo de pruebas, el Fiat-Shamir *identification protocol*, donde el objetivo sería que Alice pueda identificarse frente a Bob

probando su conocimiento del secreto S, sin revelarle información desconocida de S, y donde la seguridad del protocolo descansa en la dificultad de obtener raíces cuadradas modulo enteros compuestos grandes de factores desconocidos, que sería equivalente a la dificultad de factorizar n (siendo n = p * q).

Trent	Alice	Bob
1. Selecciona y publica un número n (a la manera RSA, n = pq, p y q primos) pero mantiene secretos p y q.	2. Selecciona un secreto S, coprimo con n, tal que 1<= S <= n-1. Calcula $v = S^2 \bmod n$, y envía esto a Trent como su llave pública. Iterativamente, selecciona un número aleatorio r tal que 0 <= r <= n-1 y envía a Bob $x = r^2 \bmod n$. 4. Calcula y envía a Bob: y = r si e = 0, ó y = rS mod n si e = 1.	3. Envía un *challenge* aleatorio, e = 0 ó 1. 5. Rechaza la prueba si y = 0, si no la acepta después de verificar $y^2 = xv^e \bmod n$.

Puede consultarse Wikipedia [21] para encontrar, además de la historia del mecanismo, un ejemplo abstracto, con diagramas, para ejemplificarlo de una manera muy simple y didáctica.

3.15. Simultaneous Exchange of Secrets (Intercambio simultáneo de secretos)

Dentro de la bibliografía utilizada para el trabajo, en lo referente protocolo, se citará lo que Schneier [2] describe con este nombre para los problemas de este tipo, junto a la solución criptográfica correspondiente, del intercambio simultaneo de secretos. Quizás porque se trata de una variante o variación, más específica, de la certificación de mensajes de correo electrónico –a su vez variación del protocolo descripto en el apartado 3.8 de este trabajo-, es que el autor lo describe justamente detallando sus diferencias.

Concretamente, la situación o problema resulta cuando Alice conoce un secreto, Bob conoce otro, y ambos tienen intención de revelárselo a la otra parte, pero sólo si ésta también lo hace.

Recapitulando entonces, y para describir completamente una versión de este protocolo, se describen los pasos a seguir en la siguiente tabla:

Alice	Bob
1. Cifra simétricamente su secreto con una llave aleatoria y envía el resultado a Bob. 2. Genera n pares de llaves: de cada par, la primera es aleatoria y la segunda es el resultado de un XOR entre la primera y de la llave del paso #1. 3. Cifra un mensaje de prueba con cada una de las llaves generadas en el paso #2, y envía todos los resultados a Bob. 5. Espera los bloques cifrados de parte de Bob y luego le envía una llave de cada uno de los pares generados. 6. Descifra las “mitades” posibles de acuerdo a las llaves recibidas y comienza	4. Realiza los mismos primeros tres pasos que Alice, pero con su secreto por supuesto. 8. Recibe de parte de Alice sus llaves y envía las suyas (sus “mitades”). 9. Realiza lo propio (ver paso #7 de Alice).

un proceso iterativo enviando parcial y simultáneamente, los pares de llaves de generadas. 7. Finalmente ambos pueden descifrar las mitades remanentes de los pares de mensajes y obtener la llave original realizando un XOR de cualquier par de llaves.	

3.16. Secure Elections (Elecciones seguras)

En el paper de P. Y. A. Ryan, The *computer ate my vote* [5], o la computadora se comió mi voto, se propone un esquema que el autor llamo "Prêt à Voter" (ver Wikipedia [29] para un breve resumen). Su enfoque se caracteriza por otorgar la mínima confianza posible a todas las partes (incluyendo el software, hardware, etc.), y podría también destacarse que se trata de un esquema en el cual el votante puede verificar que su voto ha sido incluido en el recuento general. Una descripción completa de este mecanismo está fuera del alcance de este trabajo, pero para citar un párrafo del paper original que va en línea con la idea principal de este TFI se transcribe: "*A rather novel requirement, not feasible in conventional systems, is that of voter-verifiability. Voters are able to confirm that their vote is accurately included in the count and, if not, to prove this to a judge. At the same time, the voter is not able to prove to a third party which way they actually voted. At first glance this seems impossible, but the schemes that we describe below do realise this requirement. Modern cryptography regularly makes the seemingly impossible quite routine!*".

En un sentido general, las elecciones electrónicas, como lo explican Goldwasser y Bellare [4], pueden ser consideradas como un ejemplo típico de "computación multiparte segura" (ver apartado siguiente). La versión o forma general del problema se da cuando existen m personas, cada una de ellas con su información privada x_i, y se quiere calcular el resultado una función que tomará como entrada estos valores, pero sin revelarlos. Los autores brindan en el texto una descripción de lo que sería el esquema ideado por Michael Merritt. Se intentará describir el mismo, de manera sucinta, en la tabla que sigue, considerando a Alice como al votante, y a Trent (n) como los centros de recuento de votos.

Alice	Trent (n)
1. Obtiene las llaves públicas de todos los centros (Trent (n)), y cifra de la siguiente manera, por cada uno de ellos, considerando su subíndice j (Alice), su voto v_j y un número aleatorio s_j:	2. Desde Trent(1) hasta Trent(n), cada cual cifraría $y_{i,j'} = E(y_{i+1j}, j)$. El nuevo índice j' es calculado tomando permutaciones aleatorias de los enteros [1..n], para finalmente obtener $y_{1,j}$ =

$E_1(E_2(\ldots E_n(v_j, s_j))) = y_{n+1j}$. Publica estos resultados.	$E_1(E_2(\ldots E_n(y_{n+1,\ j}, r_{n,\ j})\ldots r_{2,\ j})\ r_{1,j})$, por lo que a partir de este punto, el proceso de verificación consistiría en dos rondas de descifrado en el orden Trent(1) -> Trent(2) -> … -> Trent(n). Los valores descifrados y publicados, y el recuento realizado tomando la suma de los votos subíndice j.

En el texto de los mismos autores puede consultarse lo que más adelante exponen como *A fault-tolerant Election Protocol*, o protoclo de elecciones resistente a fallas, logrando esto al definir un umbral "t" y asumir que si hubiera menos de "t" centros (de recuento) "malos", basado en ideas de Josh Benaloh. En Wikipedia [22], hay además una buena enumeración de las distintas implementaciones y pruebas en diferentes lugares del mundo.

3.17. Secure Multiparty Computation (Computación multi-parte segura)

Si bien Goldwasser y Bellare [4] consideran como ejemplo típico de de computación mutlti-parte segura al problema de las elecciones seguras –lo que se describió en el apartado anterior-, es Schneier [2] quien describe varias opciones de implementación. De manera general, se trata en este caso de un protocolo en el cual diferentes partes pueden concentrarse para el cálculo o cómputo de una función determinada, a partir de un juego o conjunto determinado de variables. Cada participante o parte proporcionaría una o más de estas variables, el resultado de la función será luego por todas las partes conocido, pero, salvo por lo que se evidencie en el resultado de la función, ninguna de las partes conocerá o podrá lograr conocer información alguna respecto a las variables aportadas por las otras partes. El primer protocolo descripto por Schneier en este capítulo se trata de un conjunto de personas que quiere conocer el promedio de sus salarios sin que se conozcan o identifiquen los montos individualmente. El protocolo, paso a paso y considerando cuatro partes, se resumiría de la siguiente manera:

Alice	Bob	Carol	Dave
1. Genera un número aleatorio a adjuntar a su salario para cifrarlo con la llave pública de Bob, y se lo envía. 5. Recibe y descifra lo enviado por Dave, al restar el número aleatorio obtiene el total y conociendo la cantidad de partes le	2. Descifra lo recibido de parte de Alice, adjunta su salario al resultado, cifra esto con la llave pública de Carol y lo envía.	3. Recibe lo enviado por Bob en el paso #2, procede de la misma forma, y hace el envío a Dave.	4. Recibe lo enviado por Carol, de nuevo procede igual que ella y que Bob, pero envía el resultado a Alice.

es posible calcular y anunciar el promedio.			

3.18. Anonymous Message Broadcast ("Difusión" de mensajes anónimos)

Es a David Chaum a quien comúnmente se cita, como se verá a continuación, al tratar temas de necesidad de anonimato. Desde el texto de Schneier [2] por ejemplo, se describe el problema de los *Dining Cryptographers*, o Criptógrafos Cenando, que desde la versión original se traduciría:

"Tres criptógrafos se reúnen en su restorán de tres estrellas favorito; al sentarse a la mesa, el mozo les informa que alguien, anónimo, ya se ha hecho cargo de la cuenta por adelantado. Puede que sea uno de los criptógrafos quien se ha hecho cargo, o puede que haya sido la NSA. Los tres comensales respetan la posibilidad o derecho de hacerse cargo de la cuenta de manera anónima de cada uno, pero a la vez se preguntan si es la NSA quien estaría pagando…"

(Lo de que el restorán es de de tres estrellas es de Chaum –del paper original *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*- del año 1998, no de Schneier).

El problema planteado entonces es que los tres criptógrafos puedan determinar si alguno de ellos es quien pagará la cuenta (sin revelar quien por supuesto), o si es la NSA quien pagará. La solución de Chaum correspondería entonces a lo siguiente: cada criptógrafo arroja una moneda, cubriéndose con la carta del menú de tal manera que sólo él y el comensal que tiene a su derecha vean el resultado. Cada criptógrafo entonces dice en voz alta si los resultados que puede ver (el de la moneda que él mismo arrojó y el de la moneda que arrojó el criptógrafo a su izquierda en la mesa) son coincidentes. Si alguno de los criptógrafos es quien estaría haciéndose cargo de la cuenta, debería decir lo opuesto o contrario a lo que ve. De esta forma, un número impar de diferencias indicará que uno de los criptógrafos paga, mientras que un número par evidenciaría que es la NSA quien lo está haciendo (asumiendo que la cena se paga una única vez). Aunque alguno de lo criptógrafos sentados a la mesa sea quien paga, ninguno de los otros dos obtiene información suficiente como para determinar de quién se trata.

Es importante destacar que este protocolo puede extenderse a una cantidad mayor de "comensales", y por supuesto a otra variedad de aplicaciones más allá de la planteada para la situación del restorán. Es así que una aplicación posible de este protocolo sería, por ejemplo, posibilitar a un grupo de usuarios en red el envío de mensajes anónimos.

El plasmado de esto en una tabla que describa paso a paso el protocolo, como se hizo en el resto del trabajo para los otros protocolos tratados, aunque en este caso no se trate de una implementación técnicamente específica, correspondería a lo siguiente:

Alice	Bob	Carol
1. Arroja una moneda al aire, permite ver el resultado sólo a Bob. 4. Si no es quien paga, grita "coincidencia" si la moneda que arrojó resultó igual a cómo resultó la arrojada por Carol; caso contrario grita "diferencia". Si en cambio es quien paga, gritaría lo opuesto. 7. Si se gritó "diferencia" un número impar de veces, entonces alguno de los criptógrafos está pagando la cena.	2. Arroja una moneda al aire, permite ver el resultado sólo a Carol. 5. Si no es quien paga, grita "coincidencia" si la moneda que arrojó resultó igual a cómo resultó la arrojada por Alice; caso contrario grita "diferencia". Si en cambio es quien paga, gritaría lo opuesto.	3. Arroja una moneda al aire, permite ver el resultado sólo a Alice. 6. Si no es quien paga, grita "coincidencia" si la moneda que arrojó resultó igual a cómo resultó la arrojada por Bob; caso contrario grita "diferencia". Si en cambio es quien paga, gritaría lo opuesto.

En Goldwasser y Bellare [4] por otro lado, se describe el tema de manera general y brevemente bajo el título *Anonymous Transactions* o Transacciones anónimas. Ellos también mencionan a Chaum, comentando que defendió o abogó por el uso de transacciones anónimas como una manera de proteger a los individuos del mantenimiento, por parte del "*Big brother*" o "Gran hermano", de bases de datos

con todas sus transacciones, y propone, o propuso mejor dicho, el uso de pseudónimos digitales. De esta manera, sería posible realizar transacciones electrónicas asegurando que no podrían ser luego rastreadas y/o identificadas. Sin embargo, como primer problema planteado, se destaca el hecho de que siendo el individuo que realiza la transacción anónimo, la otra parte de esta transacción puede que quiera asegurarse de que la transacción esté debidamente autorizada.

Más información puede consultarse en Wikipedia, respecto a *Anonymous remailers* [24], o servidores de mails anónimos que reciben tanto el mensaje como las instrucciones de cómo o a quién reenviarlo –se encuentran referencias a implementaciones de software libre-, como también acerca del tema a ser tratado en el apartado siguiente, *Anonymous internet banking*, o banca anónima por Internet [25], donde por ejemplo se encontrará una referencia al paper *Achieving Electronic Privacy* de Chaum.

Continuando con las referencias, primero, nuevamente de parte de Chaum, puede consultarse online su artículo *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms* [7], donde explica una técnica basada en criptografía de llave pública que permite ocultar con quién se comunica un participante, sin requerir una tercera parte de confianza, permitiendo también respuestas vía direcciones de "retorno" o respuesta no rastreables.

En segundo lugar, un *paper* Frank Stajano y Ross Anderson, de la Universidad de Cambridge, Inglaterra: *The Cocaine Auction Protocol: On The Power Of Anonymous Broadcast* [8], que resulta muy interesante por plantear la situación de una subasta anónima entre partes que no confían entre sí, ni en un arbitro o parte de confianza. Se describen allí los problemas o ataques posibles en cada caso y se propone un protocolo.

Por último, de parte de Roger Dingledine, Nick Mathewson y Paul Syverson (desde el The Free Haven Project y el Naval Research Lab. de los EE.UU.), *Tor: The Second-Generation Onion Router* [9], describe el diseño del sistema de comunicación anónima quizá más conocido y popularmente utilizado hoy día, no basado en el protocolo de Chaum de los criptógrafos cenando (que fue presentado en el año 1988), sino en otro, también de su autoría o invención (pero en el año 1981), llamado *mix-networks*, o *digital mixes*, en donde se utiliza criptografía de llave pública para encapsular mensajes por capas a la manera de las muñecas rusas para su transmisión (cada capa) a través de diferentes *proxies* o "retrasmisores'(concepto de *onion routing*, o ruteo/enrrutamiento en capas de cebolla).

3.19. Digital Cash (Dinero digital)

Como lo explicaron Goldwasser y Bellare [4] en el año 2008, los medios principales para realizar transacciones monetarias hoy día vía Internet son: 1. el envío de la información de tarjetas de crédito, y 2. el establecimiento de una cuenta (corriente) con la contraparte a priori, o por adelantado.

Explican que la diferencia del uso de tarjetas respecto al efectivo, de manera general, es su falta de anonimato y susceptibilidad de monitoreo. La expresión "dinero digital" entonces es usada para describir el conjunto de técnicas y protocolos criptográficos para recrear el concepto de compra en efectivo en Internet.

También de manera general Schneier [2] por su parte explica que el dinero en efectivo es un problema; es molesto llevarlo, acarrea gérmenes, y puede ser robado. Los cheques y las tarjetas de crédito redujeron el flujo de dinero en efectivo, pero su completa eliminación es virtualmente imposible. Comenta el autor que los políticos y vendedores de drogas nunca lo permitirían. Continúa explicando que, por un lado, los cheques y las tarjetas de crédito permiten el monitoreo y auditoría sin poder ocultar a quien se le paga, y por otro lado, estos medios de pago permiten también que se invada la privacidad (de quien paga) de una manera desconocida hasta el momento (donde uno come, donde uno carga combustible, etc.). Felizmente, y aunque "complicados", en palabras del autor, existen protocolos que permiten el envío de mensajes autenticados pero no *traceables* (rastreables o localizables). Es importante aclarar que a estos mecanismos también se los denomina *anonymous digital cash* (dinero digital anónimo). Existen variedad de protocolos, se explicará a continuación, de manera general y en el formato de tabla de partes como en el resto del trabajo, la primera de las alternativas descriptas por Schneier (en este caso Trent hará el papel del banco, y Bob el de un comercio o quien, en general, recibe un pago):

Alice	Trent	Bob
1. Prepara 100 órdenes de pago anónimas por $1000 cada una, las coloca éstas con papel carbónico en 100 sobres diferentes, y se los envía a Trent. 3. Recibe el sobre firmado de parte de Trent y utiliza ese importe para pagarle a Bob.	2. Recibe los 100 sobres de parte de Alice, abre 99 y confirma que son órdenes por $1000; firma la que quedó cerrada, devuelve este sobre cerrado a Alice y resta $1000 de su cuenta corriente. 7. Verifica su propia firma antes de pagar a Bob. .	4. Bob recibe la orden de pago y verifica la firma de Trent para confirmar que la orden es válida; para luego entregársela a Trent para cobrar.

Más información puede consultarse en Wikipedia respecto al *Anonymous internet banking*, o banca anónima por Internet [25] (o cómo allí bien se aclara, pseudo-anónimo), *Ecash* [26] (donde se comenta la modalidad propuesta por Chaum mediante firmas ciegas) y Bitcoin [27] (una implementación, concreta, descentralizada).

3.20. Visual Criptography (Criptografía visual)

Como explican Moni Naor y Adi Shamir en su paper presentado en la conferencia Eurocrypt del año 1994 [10], se trata aquí de un tipo de esquema criptográfico que decodifica imágenes. Exponen que se trata de un esquema perfectamente seguro, fácil de implementar, y lo extienden a un variante "visual" del protocolo de secreto compartido.

Se consideró en ese trabajo el problema del cifrado de material impreso (texto impreso, manuscrito, fotografías, etc.), para que de manera segura pueda ser descifrado o decodificado a simple vista.

Para entender el modelo básico propuesto, imagínese una página impresa (texto-cifrado) a transmitir o enviar vía un canal inseguro, y, por otra parte, una transparencia. Esta transparencia cumpliría el rol de la llave secreta. Tanto la página impresa como la transparencia aparentarán no tener sentido, o mejor dicho, más específica y concretamente, en palabras de los autores citados, serán indistinguibles respecto a ruido aleatorio. Es cuando se superponga la transparencia sobre la página impresa que el texto-en-claro será revelado.

Como se destaca en el *paper* original, el sistema sería similar a un OTP (*One Time Pad*), en tanto que cada página impresa (texto-cifrado) es descifrada mediante una transparencia (llave) diferente.

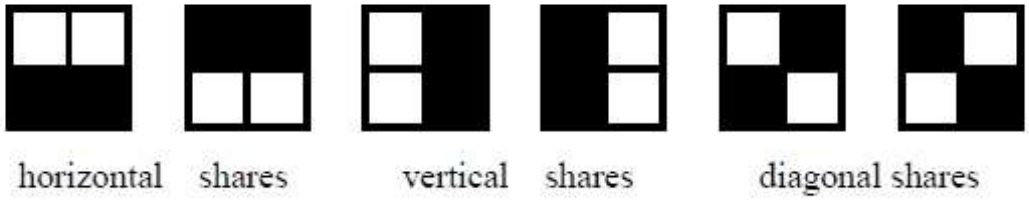
Cabe remarcar además que, dada la simpleza del esquema, el sistema podría ser utilizado por cualquier usuario, quien no debería a priori poseer conocimientos criptográficos, ni tendrá necesidad de realizar cálculos de ningún tipo.

Como se adelanto más arriba, los autores propusieron este esquema extendiéndolo como una variante visual del problema del secreto compartido, donde se genera una cantidad determinada de llaves o partes, y sólo a partir de otra cantidad determinada de llaves es posible reconstruir el secreto (ver apartado 3.1). En este sentido y en este esquema se consideraría –en su versión más simple-, que el mensaje o texto-en-claro consiste en una colección de pixels blancos y negros. Estos se manejarán individualmente, y cada uno de estos aparecerá "codificado" en cada una de las versiones o partes del secreto. Esta codificación resulta en un conjunto de pixels de salida por cada pixel de entrada. Es decir, por cada pixel en la imagen original, tendremos más pixels en cada transparencia, o conjuntos de subpixels. A cada grupo de subpixels, se los denominó *shares* (partes). Cada parte es entonces una colección de pixels blancos y negros. Estos se dispondrán de manera tal que faciliten la interpretación

visual, a simple vista. Esto significa, resumidamente, el agrupamiento de los subpixels próximos entre sí, para que una persona pueda percibir directamente las contribuciones de cada uno de los colores. El resultado de la codificación podría describirse con matrices booleanas de n x m, n correspondiendo al número de transparencias y m al número de subpixels por cada parte (o pixel de la imagen original). El valor 1 corresponderá a un elemento cuando en esa transparencia (índice de la matriz) y el color de ese subpixel (índice de la matriz) sea color negro. De esta manera es que la operación binaria "or" entre los elementos de las matrices se equipararía al proceso de superponer una transparencia sobre otra, y se define un valor umbral para determinar cuando los grises se "verán" blancos y cuando negros, ya que un pixel original blanco quedará, por decirlo de alguna manera, sucio (los subpixels generados no serán todos blancos).

Esto queda más claro de acuerdo al ejemplo sugerido por los autores para caso (2,2), es decir, el secreto compartido en dos partes, ambas necesarias para reconstruirlo. En este caso, cada pixel de la imagen original se convertirá en un grupo de cuatro subpixels. Estos cuatro subpixels se interpretarán visualmente como grises, y el resultado de superposición de transparencias por supuesto no evidenciará la imagen original en blanco y negro, sino en gris y negro. En este caso se propone la utilización de seis grupos de cuatro subpixels para la codificación, cada uno de estos grupos "parece" gris, ya que todos tienen, en diferente distribución o diagramación por supuesto, dos subpixels blancos y dos negros. Un pixel blanco de la imagen original es "compartido" utilizando dos grupos iguales (grises) y uno negro utilizando dos grupos complementarios (al superponerse, se interpretará el color negro). Cada grupo es una selección aleatoria; en el caso de estar compatiendo un pixel color blanco, se elige entre 3 distribuciones o diagramaciones, y el mismo grupo se comparte entre las dos partes; en caso de compartir un pixel negro, se elige entre 3 pares complementarios, y se comparte imprimiendo un grupo para una parte, y el complementario para la otra.

Como ejemplo, se copia a continuación la figura # 1 del *paper* original que ejemplifica esto de las distribuciones o diagramaciones, es decir, las alternativas para los *shares* o partes según se describió más arriba.



Aunque la implementación de esta técnica o el esquema, en el uso concreto, diferente a los protocolos descriptos hasta el momento, para cumplir en este apartado presentando una la tabla que detalle los pasos llevados a cabo para la implementación del protocolo como en el resto del trabajo, se detallan de manera general a continuación las tareas que les corresponderían a cada parte en una modalidad en particular (Trent conociendo el secreto, y generado las transparencias de acuerdo a *shares* o partes de 4 -2 x 2- pixels por cada pixel de "entrada", como en la imagen anterior, y Alice y Bob siendo quienes recibirán estas transparencias generadas por Trent a partir del secreto –o imagen secreta- para poder reconstruirla):

Trent	Alice	Bob
1. A partir de una imagen de n x m pixels, genera dos imágenes de (n x 2) x (m x 2) pixels. La primera imagen estará compuesta por *shares* o partes aleatorias (alguna de las 6 combinaciones posibles –ver imagen-), la segunda imagen se construirá de acuerdo a	2. Recibe la primera imagen construida por Trent. Si decide "reconstruir" con Bob, deberá obtener su transparencia y superponerla a la suya para revelar la imagen secreta.	3. Recibe la segunda imagen construida por Trent. Si decide "reconstruir" con Alice, deberá obtener su transparencia y superponerla a la suya para revelar la imagen secreta.

la imagen secreta; por cada pixel negro, pintado, o "con información" como se refiere en el *paper* original, se utilizar el *share* complementario, para el resto de los pixels (blancos o sin información), se usará el mismo *share* aleatorio que se usó en la primera transparencia. Se envía la primera transparencia a Alice y la segunda a Bob.		

Puede consultarse más información en Wikipedia [34], donde se presenta una animación que superpone dos transparencias, que demuestra el uso de esta técnica con el logo de la enciclopedia online.

4. Conclusiones

El trabajo intentó pasar revista sobre un conjunto de protocolos criptográficos que podrían utilizarse para la resolución de diferentes problemas. Como se ha visto, varios de estos protocolos descansan o se apoyan sobre otras primitivas criptográficas, y otras veces incluso, hasta se trata de un recorte o aplicación parcial de otros protocolos, aquellos considerados principales o más comúnmente utilizados. Sin embargo, se consideró de algún interés, en el presente trabajo, el haberlos enumerado –aunque no se trate de una enumeración completa-, descripto y ejemplificado –hasta donde fue posible, de acuerdo al limitado alcance del trabajo-, entendiendo que esto pudiese servir al menos como una reseña de ejemplos de problemáticas y/o situaciones dadas en el cada vez más amplio espectro de las comunicaciones, que podrían resolverse con la ayuda de herramientas criptográficas, más allá de las más habituales, o de las que se nos vienen a la mente en primera instancia al pensar en criptografía. El desarrollo de dos programas de ejemplo o demostración además, complementando las descripciones del presente texto, aportarían la posibilidad de realizar pruebas concretas, viendo o monitoreando el "ida y vuelta" de los datos específicos en cada caso, para cada uno de aquellos protocolos que fueron implementados. El código fuente provisto, en un lenguaje de programación y para plataformas popularmente utilizadas actualmente, haciendo uso de una librería criptográfica también reconocida y muchas veces referencia en lo que es código abierto en la materia, de alguna manera (entiéndase: sin mayores pretensiones y con cierta indulgencia) podría servir de ejemplo y/o base para futuros desarrollos criptográficos simples, de prueba o de demostración, que también necesiten trabajar con números enteros grandes, primos, números aleatorios seguros, aritmética modular, generación de llaves públicas y privadas (utilizadas luego tanto para el cifrado y descifrado como para la firma y verificación de firmas digitales), codificaciones varias (base64, pkcs8, etc.), y demás problemáticas a resolver en el desarrollo de este tipo de programas, resueltas aquí de forma definitivamente no óptima ni elegante, pero sí, en cuanto fue posible y de acuerdo al objetivo del trabajo, simple y sencillamente.

5. Apéndices

Como apéndice principal, se adjuntarán a continuación las capturas de pantalla de los programas de pruebas desarrollados con el fin de ejemplificar prácticamente los procedimientos involucrados en algunos de los protocolos descriptos a lo largo del desarrollo del trabajo; principalmente, se adjuntan aquellas capturas que muestran cómo describen los programas los pasos llevados a cabo en la implementación de los protocolos.

Como apéndice externo, en un disco CD-ROM, serán adjuntados los códigos fuente de ambos programas, el binario ejecutable de la aplicación java principal –de escritorio-, y el instalador o archivo ".apk" de la aplicación android. La primera aplicación fue desarrollada en lenguaje de programación Java por supuesto, utilizando el entorno de desarrollo eclipse, junto con su plugin "*WindowBuilder*", y las librerías criptográficas del proyecto "*Bouncy Castle*". La aplicación android por su parte, fue desarrollado con el mismo entorno de desarrollo (eclipse) y con el SDK (*Software Development Kit*, o Kit de Desarrollo de Software) provisto por Google para el desarrollo de aplicaciones en esta plataforma.

Para la ejecución del programa de escritorio deberá contarse con un entorno Java versión 1.6 instalado en el equipo, y para arrancarlo bastará realizar un *doble-click* sobre el archivo ".jar". Las partes pueden comunicarse vía red TCP/IP, pero por supuesto puede utilizarse "localhost" (parámetro *default*) para la realización de las pruebas en un único equipo. El menú principal contiene del una serie de botones por cada una de las partes, agrupados de acuerdo al protocolo implementado. Cada uno de estos botones abrirá una nueva ventana, en la cual podrán completarse los parámetros necesarios para el establecimiento de la conexión y la realización de los pasos correspondientes. Respecto a la aplicación android, la versión mínima del sistema operativo soportada sería la 2.1.

En cada uno de los apartados de esta sección de apéndices se mostrarán específicamente las pantallas con la información clave o más importante de cada alternativa. Se presenta a continuación una captura de pantalla del menú principal, o primer pantalla del programa de escritorio:

Como se adelantó en la introducción del trabajo, el programa permite "hacer de" o "actuar como", Alice, Bob, Carol, Dave, o Trent, para cada uno de los protocolos particularmente. Cada uno de los botones con los nombres de los personajes, agrupados por protocolo, abrirán nuevas ventanas para realizar las operaciones correspondientes según corresponda.

Las capturas de pantalla que se mostrarán a continuación en los apartados siguientes presentan la forma general en que se requieren los parámetros necesarios en la aplicación.

Las conexiones entre las partes se realizan vía TCP/IP, por eso siempre es necesario especificar tanto las direcciones (números IP, nombre de dominio o "localhost" -como en los ejemplos, para conectarse al mismo equipo-) y los puertos (en los ejemplos se utilizaron puertos cualesquiera, mayores a 1024, pero, en tanto coincidan por supuesto, la elección sería libre, con límite máximo de 65535).

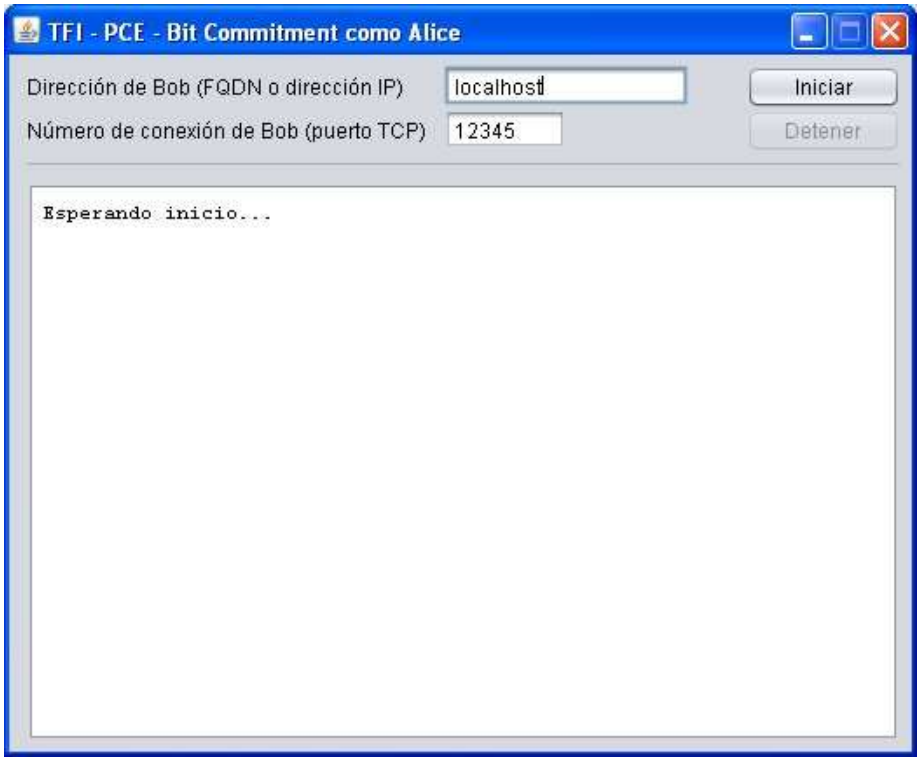
Cuando el protocolo precise información extra más allá de la forma de contactar o conectar con las otras partes involucradas, estos requerimientos se verán en la misma pantalla o ventana original de la parte en cuestión, o en otras ventanas o cuadros de dialogo sucesivos, que aparecerán durante la ejecución del protocolo, en el momento necesario.

Respecto a la aplicación android, el "transporte" sería en este caso vía mensajes de texto o SMS; se intentó que la aplicación sea muy intuitiva y no necesitaría mayores explicaciones más allá de la aclaración acerca de la manera de enviar estos mensajes: el programa codificará el resultado del *hash* a enviar y delegará al sistema operativo el envío, por lo que se le presentará al usuario la interfaz de envío que comúnmente utiliza, con el cuerpo del mensaje prellenado, y la posibilidad de seleccionar el destinatario de manera estándar. La otra parte, por supuesto, debe haber ejecutado la aplicación y haber elegido el modo "espera" para poder llevar a cabo la operatoria.

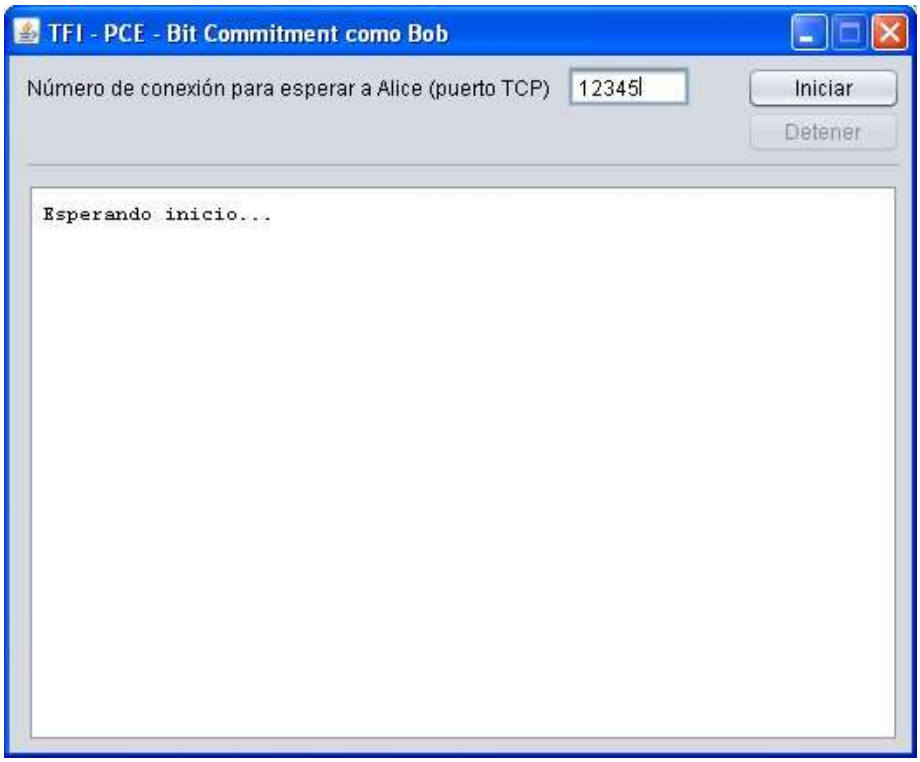
Téngase en cuenta por último que es el primer apartado entre los que siguen (el referido a la implementación de un esquema de *bit-commitment* de una oferta económica) donde se describen algunas generalidades respecto al uso o interfaz de usuario de la aplicación en general, es decir, que aplican de la misma forma también a los otros de los protocolos implementados en la aplicación, por lo que se sugiere revisar esas descripciones y capturas en primer lugar.

5.1. Capturas de Bit Commitment (Compromiso de bit)

Esta es la ventana que se presenta al seguir, dentro del grupo de *bit-commitment*, el botón “como Alice” desde el menú principal de la aplicación. Como se ve, se debe especificar la dirección y puerto de Bob para contactarlo. La información a “comprometer” (aquí, en esta implementación de ejemplo, una oferta económica) será requerida más adelante.



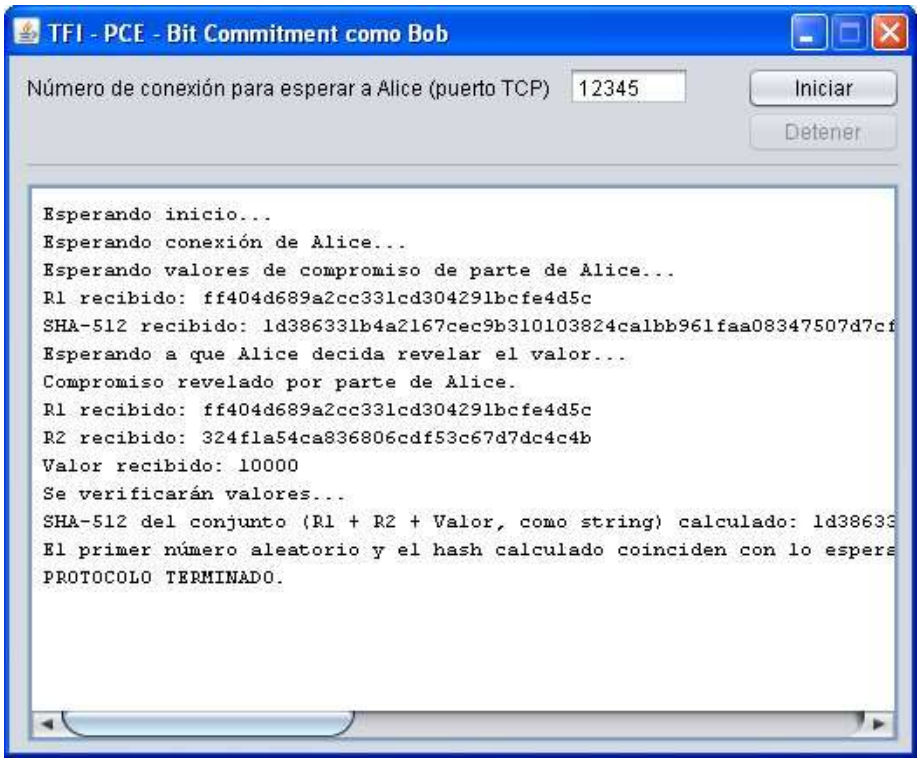
La captura siguiente corresponde a lo que visualizará al presionar el botón “como Bob”, en donde sólo es necesario especificar en que puerto se “escuchará” o se “esperará” la conexión a ser iniciada por Alice. Este mecanismo se repite, según corresponda, en el resto de las implementaciones de los protocolos en la aplicación; debe tenerse presente que antes de iniciar la ejecución de los pasos del protocolo por parte de Alice (en su ventana), debe iniciarse (presionar el botón “Iniciar”) en esta ventana para recibir la información que Alice generará y enviará en su primer paso, caso contrario, al intentar conectar con Bob, se mostrará un error de conexión.



En la captura siguiente, Alice, después de generar los números aleatorios R1 y R2, debe especificar el valor a comprometer. Cabe aclarar, también en relación a un aspecto común a toda la aplicación, que los mensajes que se visualizan en el área de texto principal, en el centro de la ventana, corresponden a los mensajes descriptivos mostrados a manera de bitácora o *logs* de los procedimientos realizados.

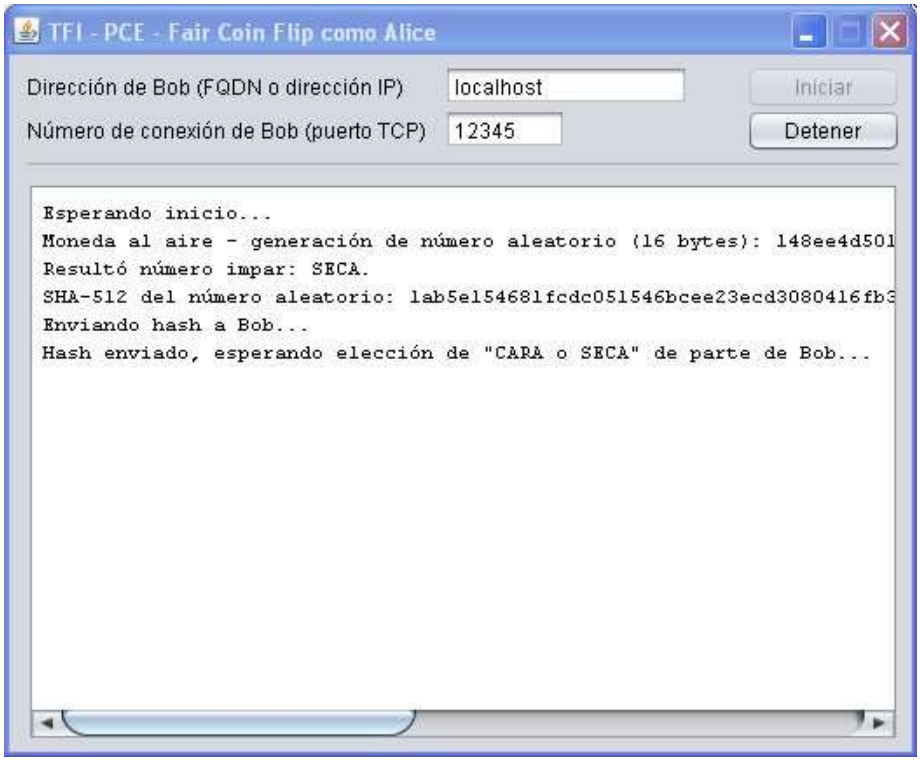
Este cuadro de dialogo cumple la función de espera o *delay* para la revelación del compromiso tomado por Alice.

Finalmente, aquí se muestra la ventana de Bob, una vez que Alice reveló sus valores, donde se detalla por ejemplo los valores recibidos y el resultado de la verificación correspondiente.

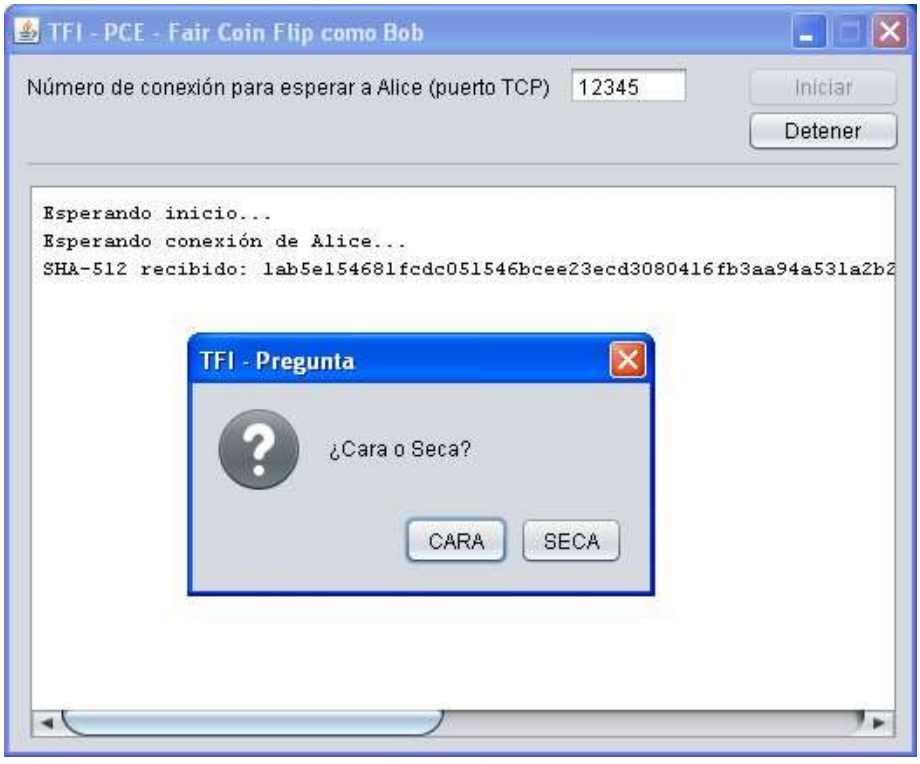


5.2. Capturas de Fair Coin Flip (Cara o seca justo)

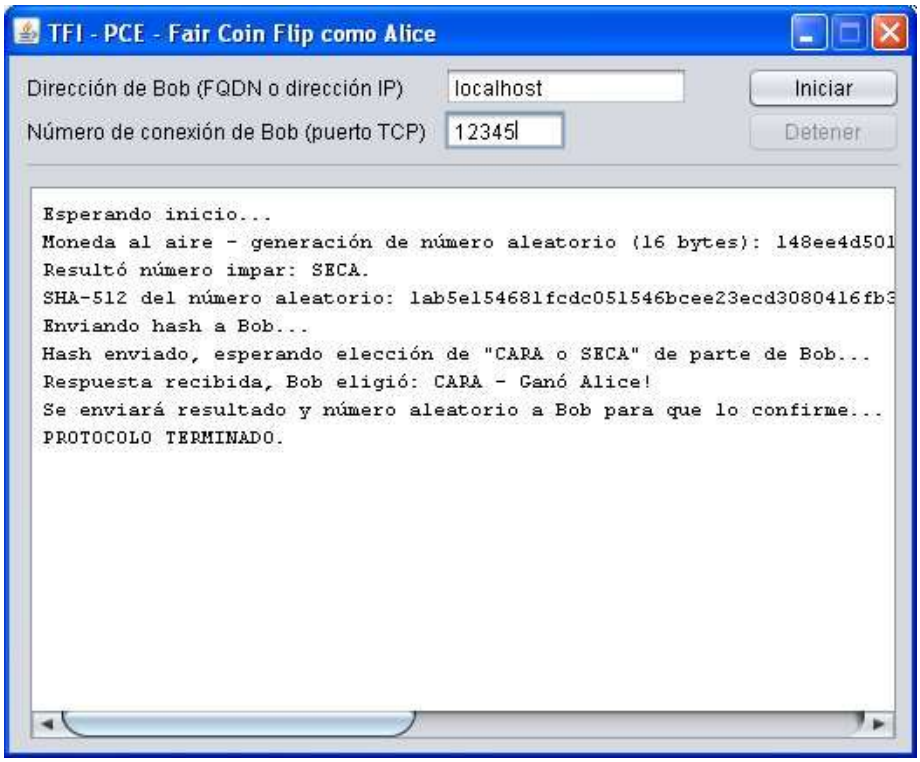
Se presenta a continuación la pantalla o ventana que ve Alice luego de iníciar, generando el número aleatorio y preguntando a Bob "¿cara o seca?":



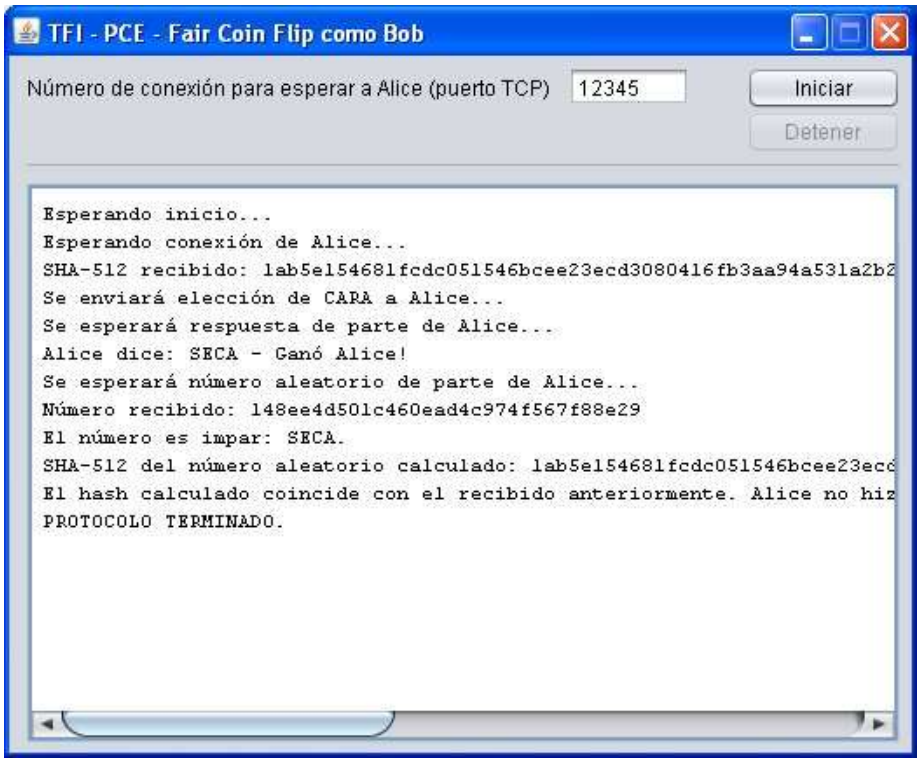
De esta manera visualiza Bob, en su ventana, la pregunta de parte de Alice:



De nuevo Alice, una vez recibida la selección hecha por Bob (en el caso del ejemplo, Bob eligió "cara").



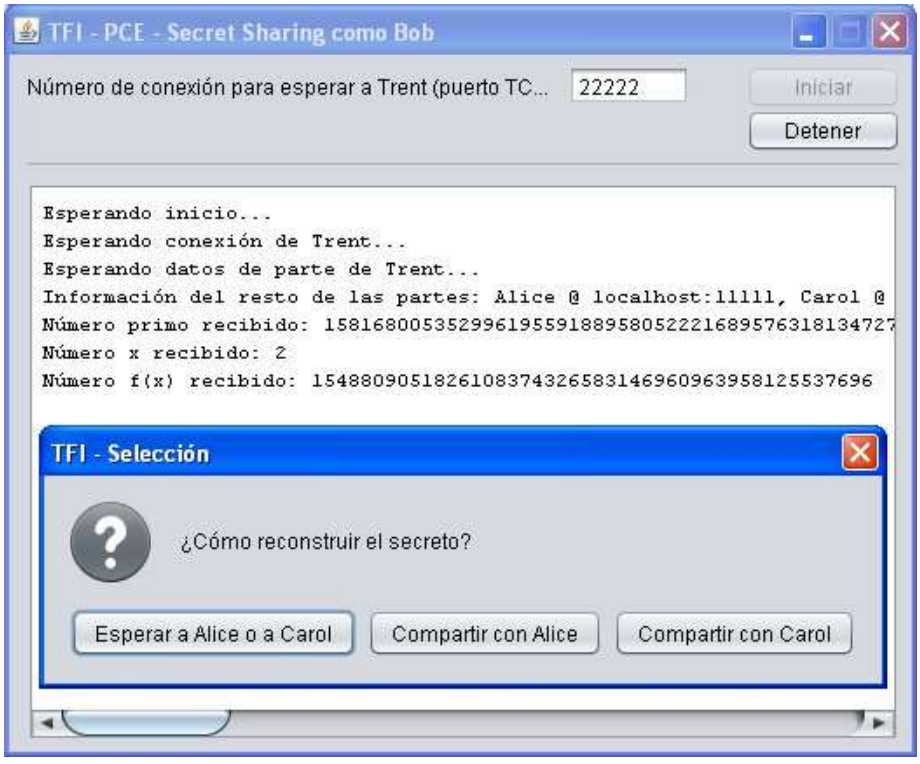
Esta última pantalla le informa a Bob acerca de los pasos llevados a cabo para confirmar que Alice no ha hecho trampa.



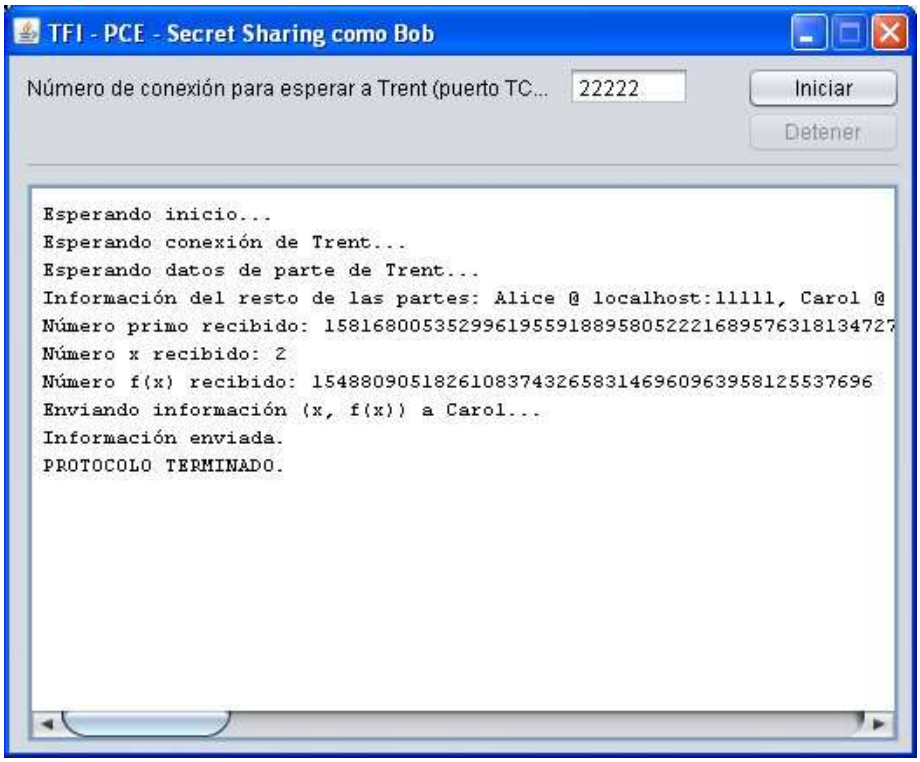
5.3. Capturas de Secret Sharing (Secreto compartido)

Aquí es Trent quien conoce el secreto, y quien debe generar y enviar las partes, por lo que, respecto a los ejemplos anteriores, más datos son requeridos para iniciar.

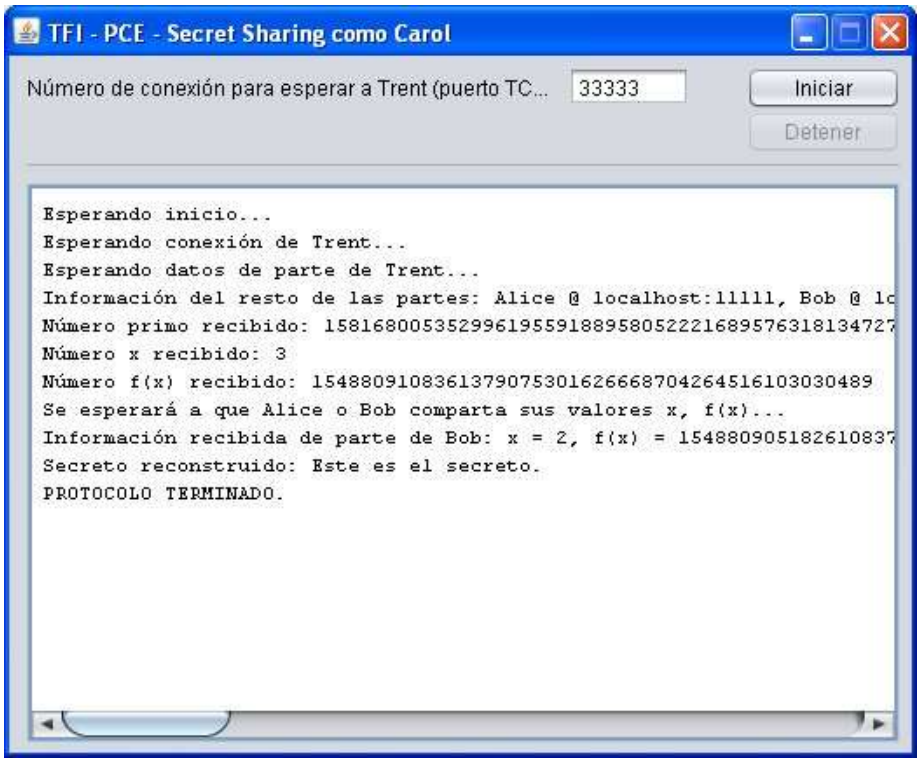
Se muestra aquí cómo se pregunta a Bob si compartirá con alguna de las otras partes:



Después de seleccionar “Compartir con Carol” Bob ve...

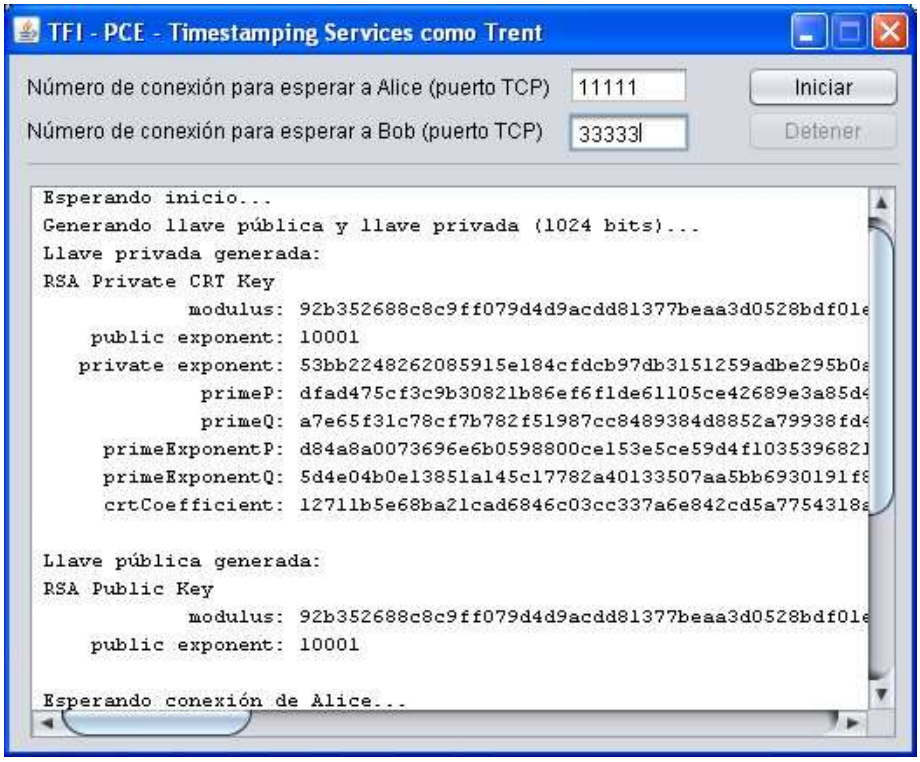


Carol por su parte (después de haber seleccionado “Esperar a Alice o Bob”) recibe la información de parte de Bob, y junto a la información recibida anteriormente de parte de Trent, reconstruye el secreto:

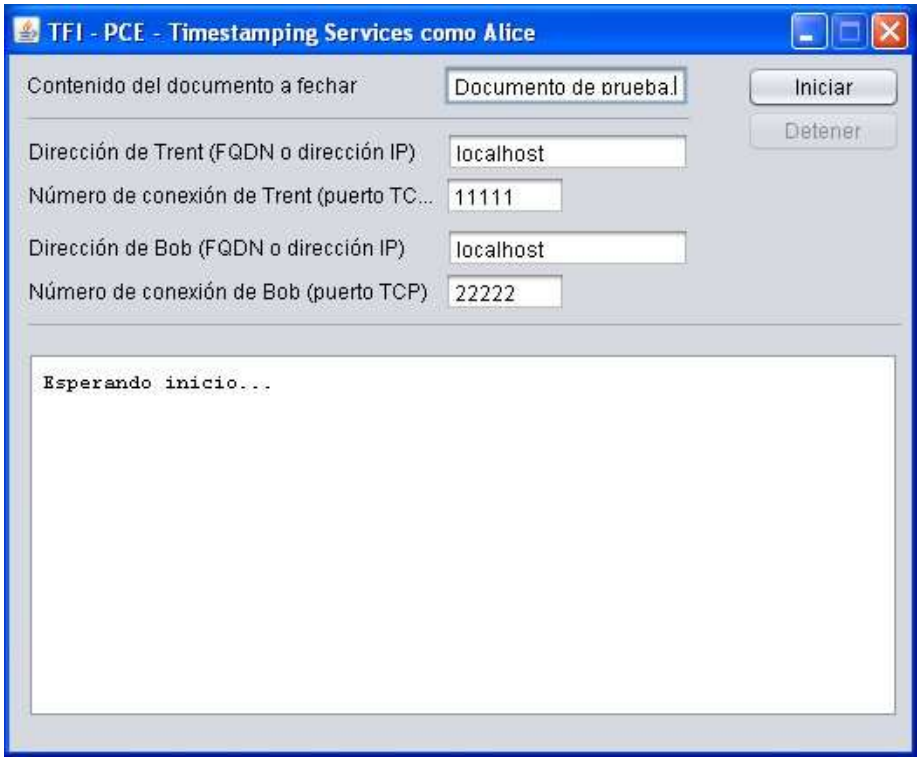


5.4. Capturas de Timestamping Services (Servicios de fechado)

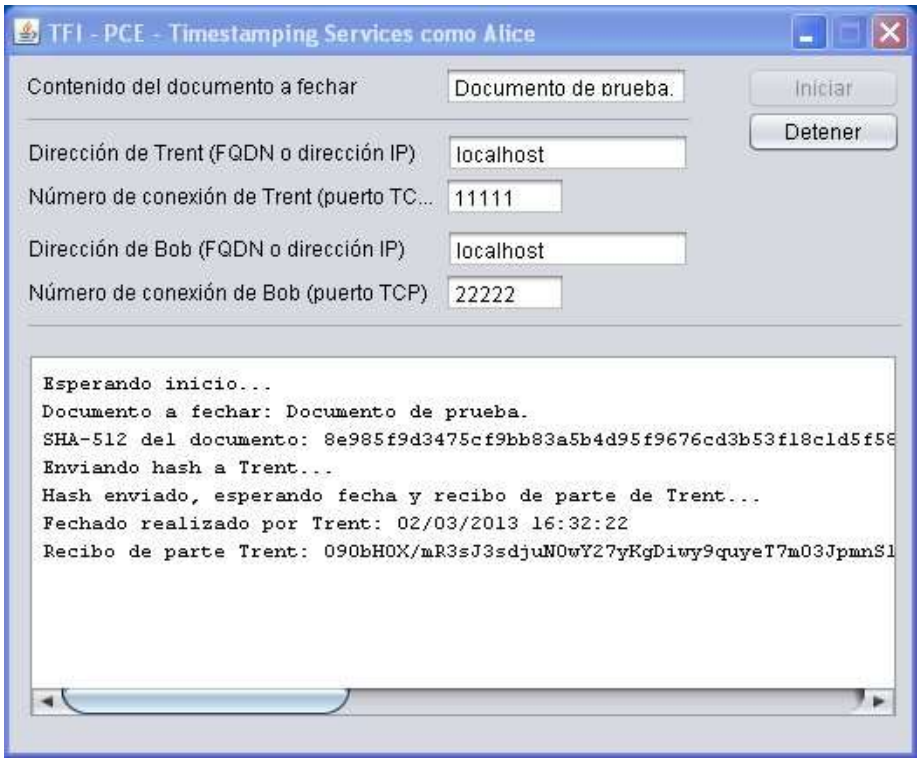
Aquí se muestra el resultado de la generación del par de llaves (pública y privada) de Trent para quedar a la espera de la conexión de Alice:



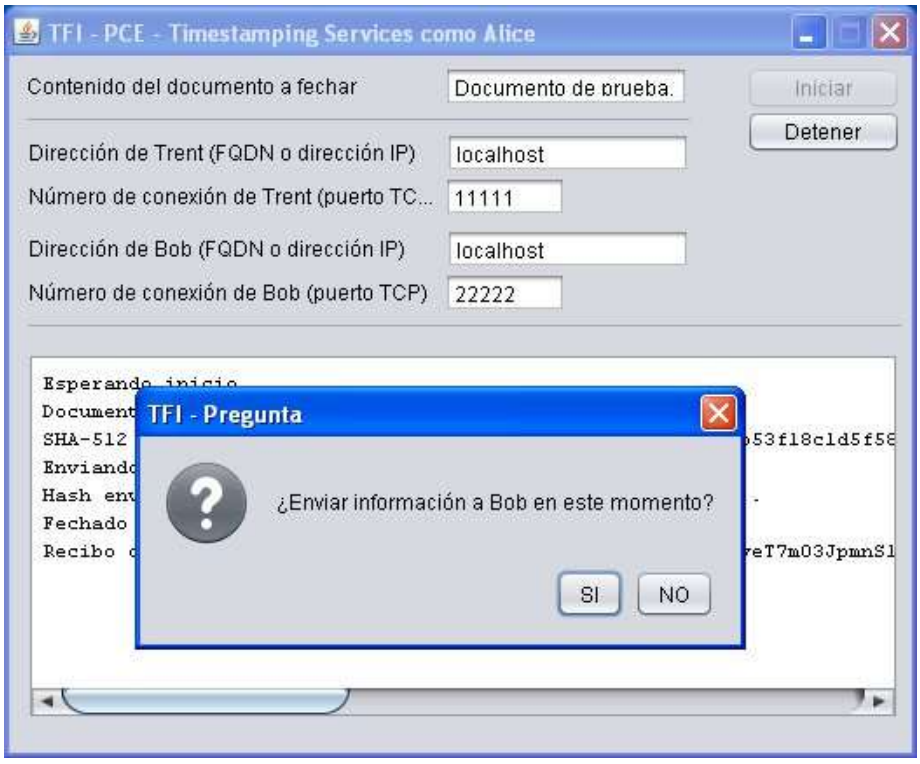
Alice debe especificar lo que se entiende aquí como el contenido del mensaje a fechar:



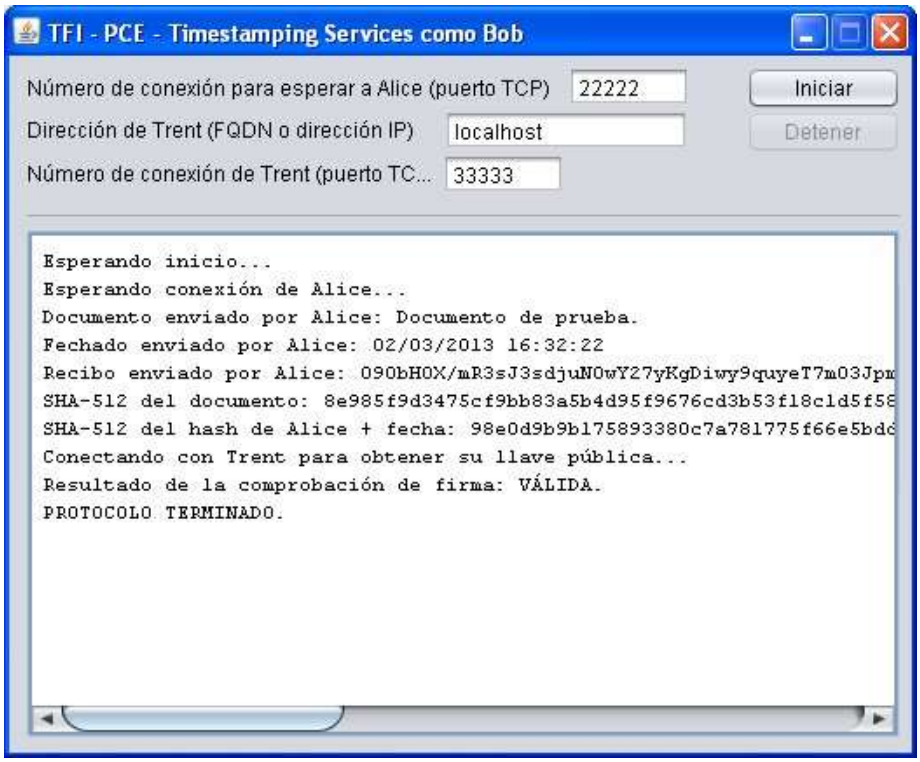
Al iniciar, contactará a Trent, enviará la información necesaria y recibirá su recibo:



Hecho lo anterior, de nuevo como una suerte de *delay*, se espera a que Alice decida enviar la información correspondiente a Bob.

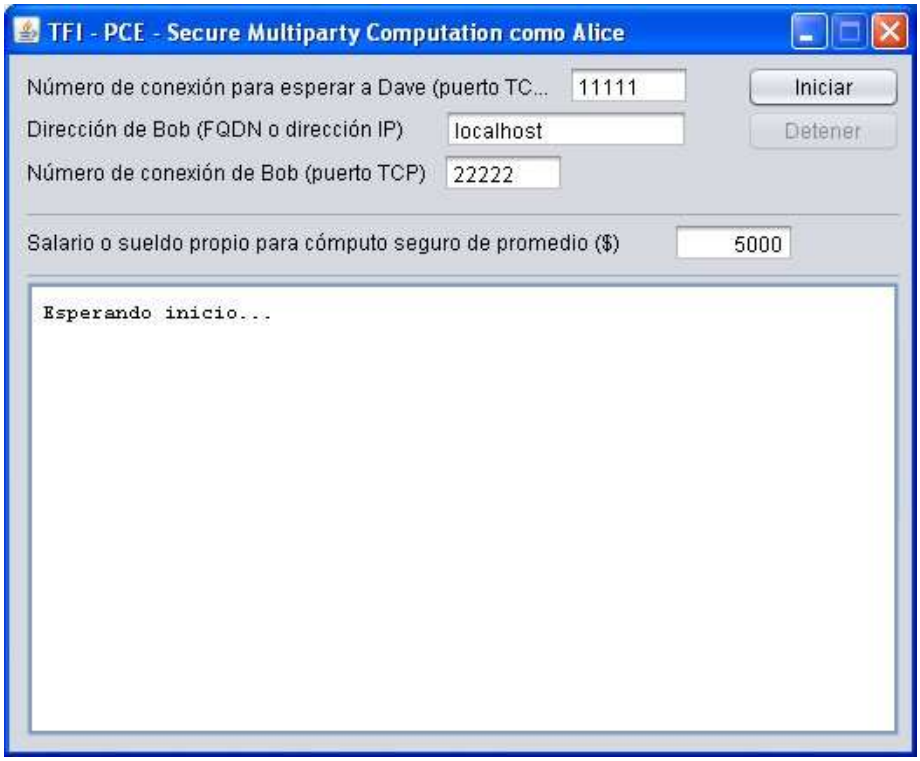


Por último, Bob recibe la información enviada por Alice, contacta a Trent para obtener su llave pública, y realiza las verificaciones correspondientes:

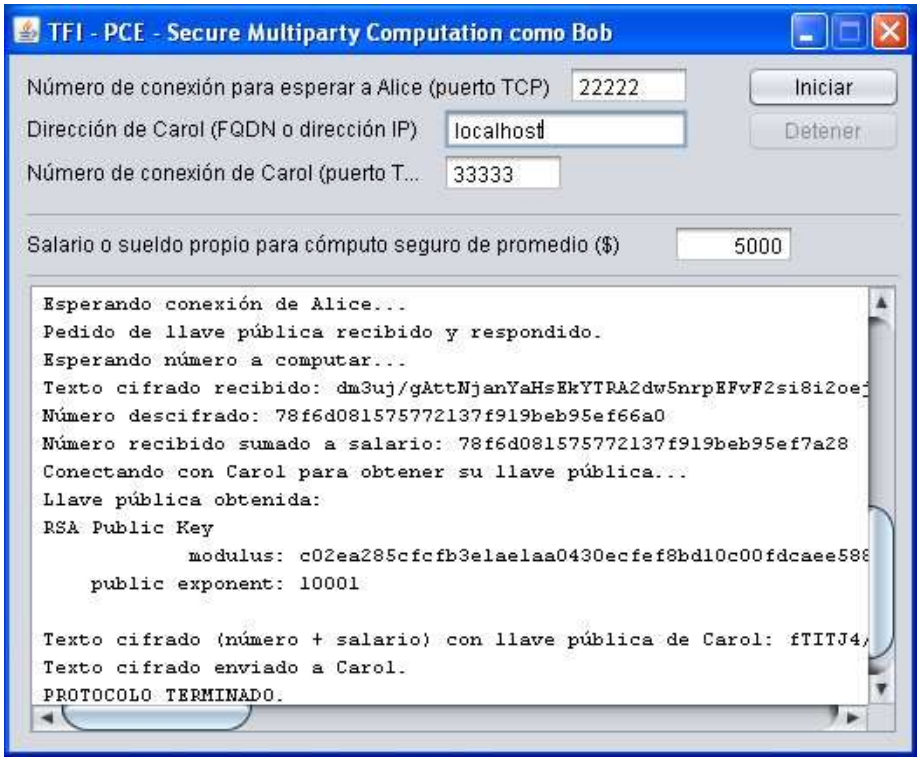


5.5. Capturas de Secure Multiparty Computation (Computación multi-parte segura)

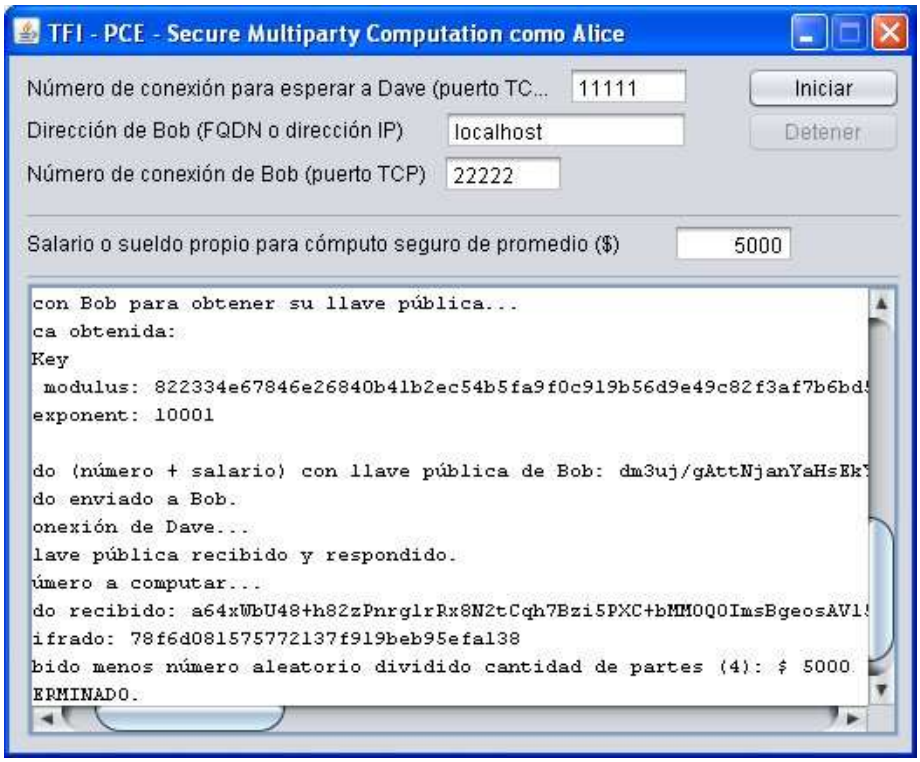
Aquí se ve cómo se implementó un "cálculo de promedio salarial" seguro, primero Alice:



Aquí se muestra como Bob recibió el número (cifrado con su llave pública), suma su salario, y envía esto a Carol para que realice el mismo procedimiento (aunque enviará su resultado a Alice).

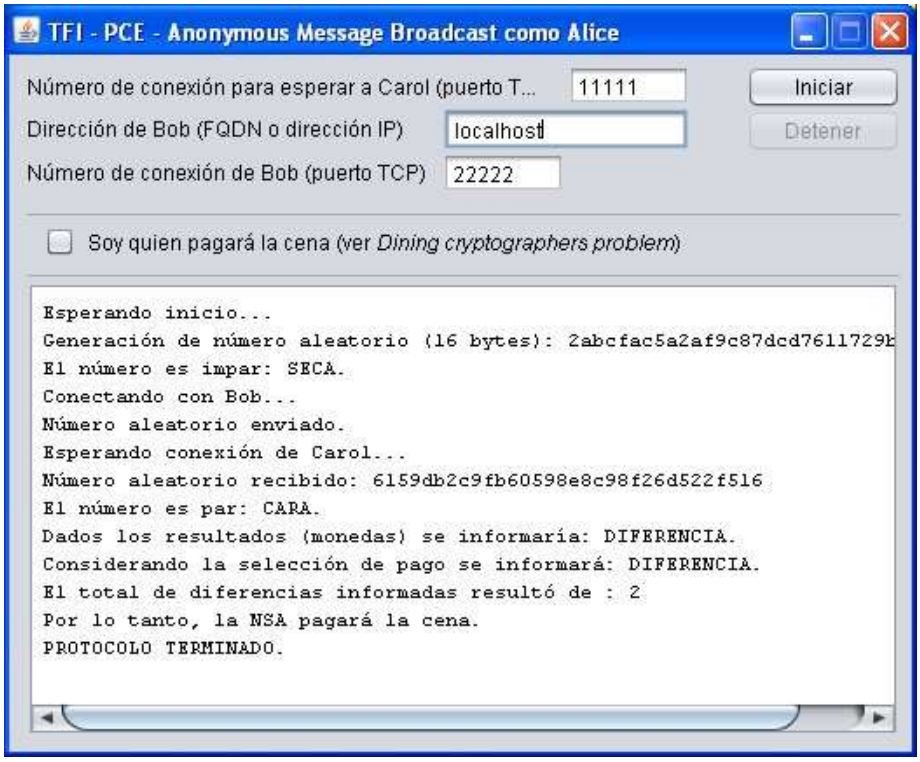


Ahora con Alice nuevamente, quien ya cuenta la información necesaria para calcular el promedio de salario de todos los participantes..

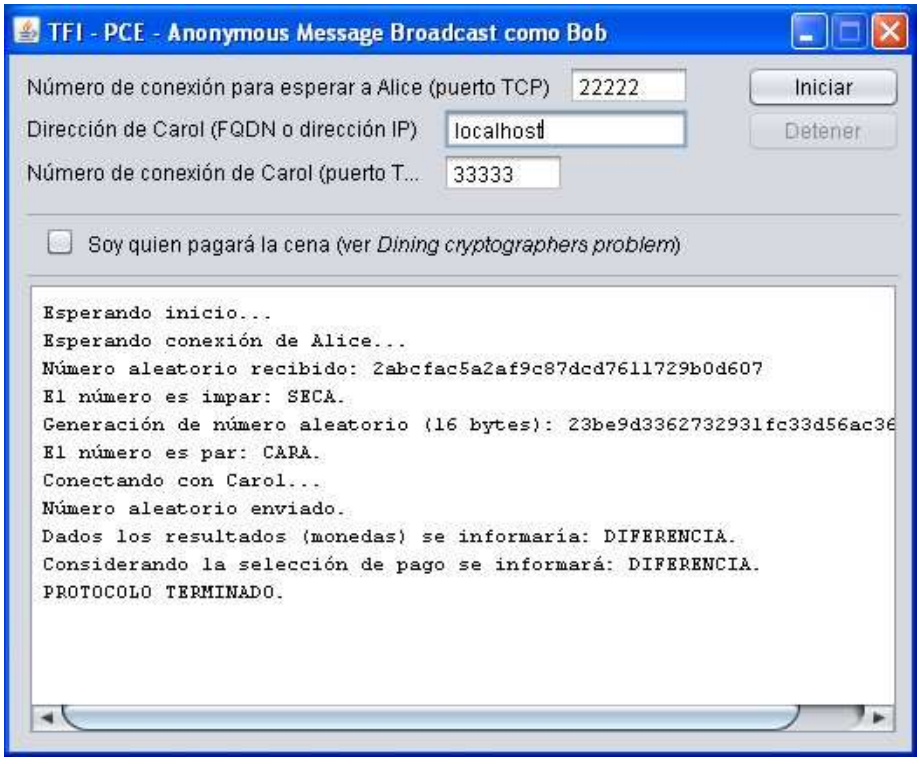


5.6. Capturas de Anonymous Message Broadcast ("Difusión" de mensajes anónimos)

Esta corresponde a la ventana que vería Alice en la implementación de los *Dining cryptographers problem*, o problema de los criptógrafos cenando, luego de haber terminado el procedimiento.



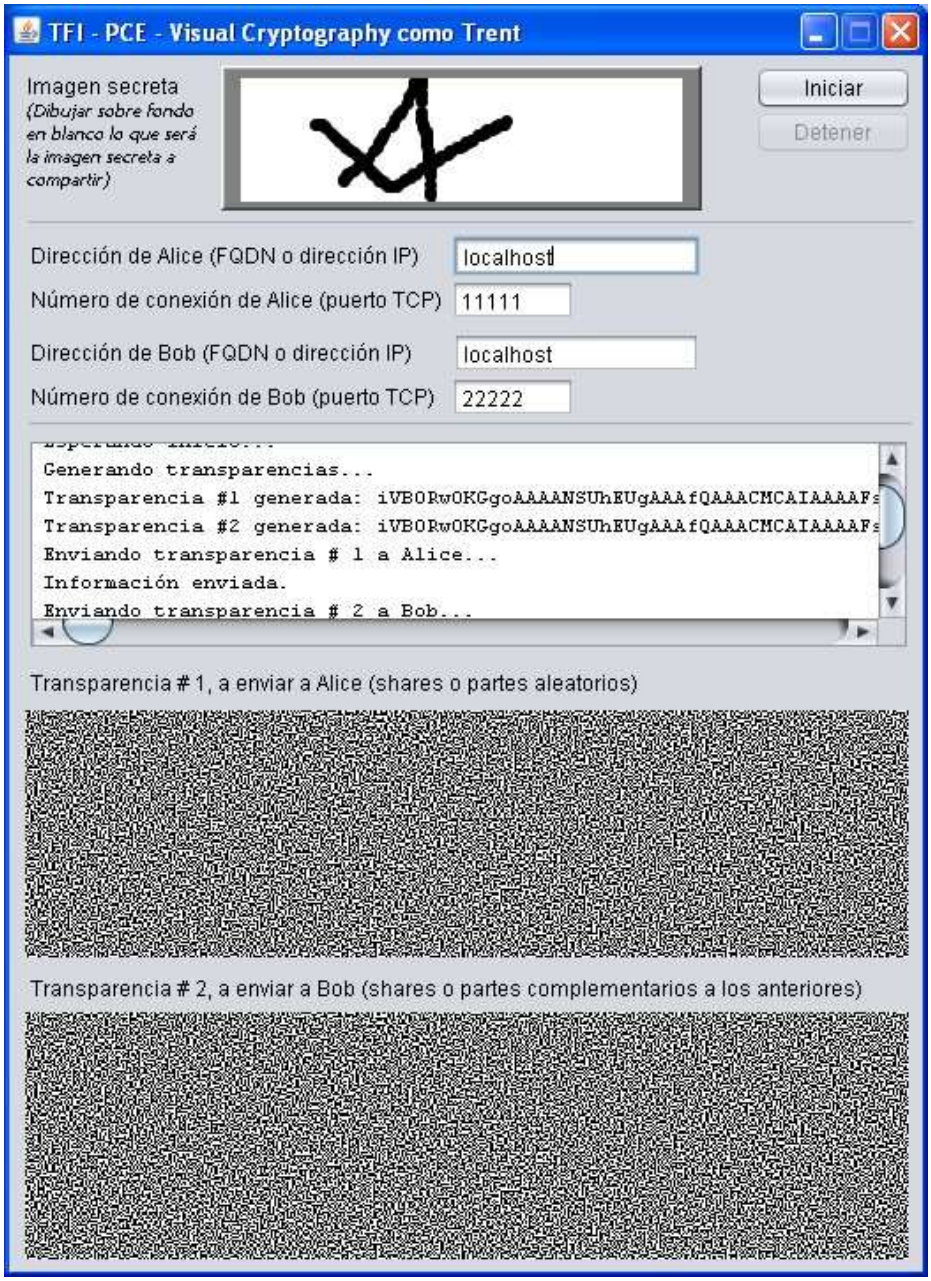
Aquí se muestra la ventana de Bob, describiendo también lo que fue su parte en el proceso:



5.7. Capturas de Visual Cryptography (Cryptografía visual)

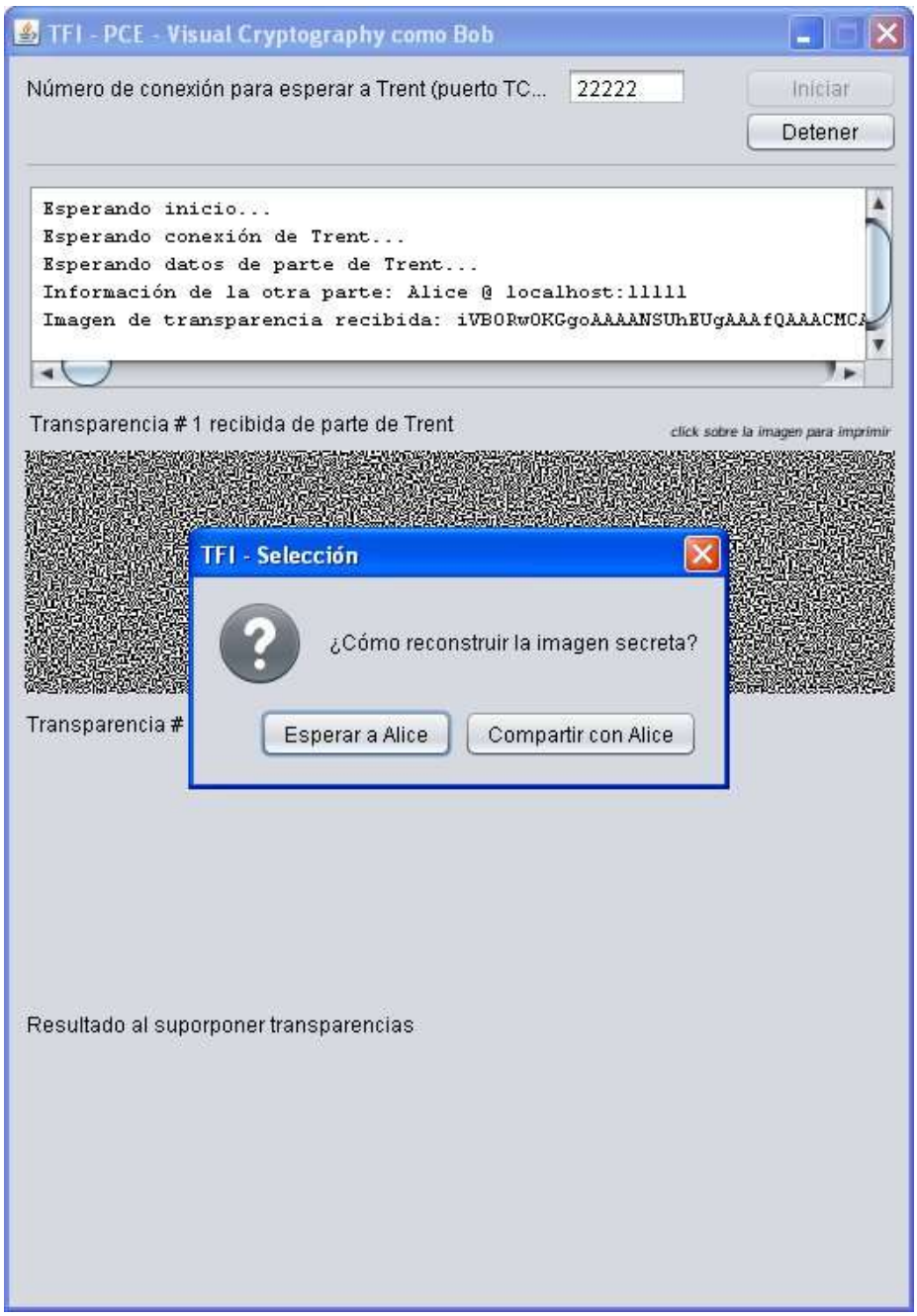
La implementación de este protocolo requirió desarrollar una interfaz de usuario algo diferente; si bien la manera de especificar la forma de contactar a las otras partes es la misma, aquí debe dibujarse lo que será la imagen secreta, mostrar las imágenes generadas (transparencias) y en el caso de Alice y Bob, mostrar el resultado de lo que sería la superposición de las imágenes o transparencias.

Aunque Alice y Bob deben estar a la espera, el primer paso aquí corresponde a Trent, quien una vez que dibuje el secreto e inicie, verá en su ventana algo similar a la captura que sigue (en el ejemplo se dibujó un garabato de prueba:

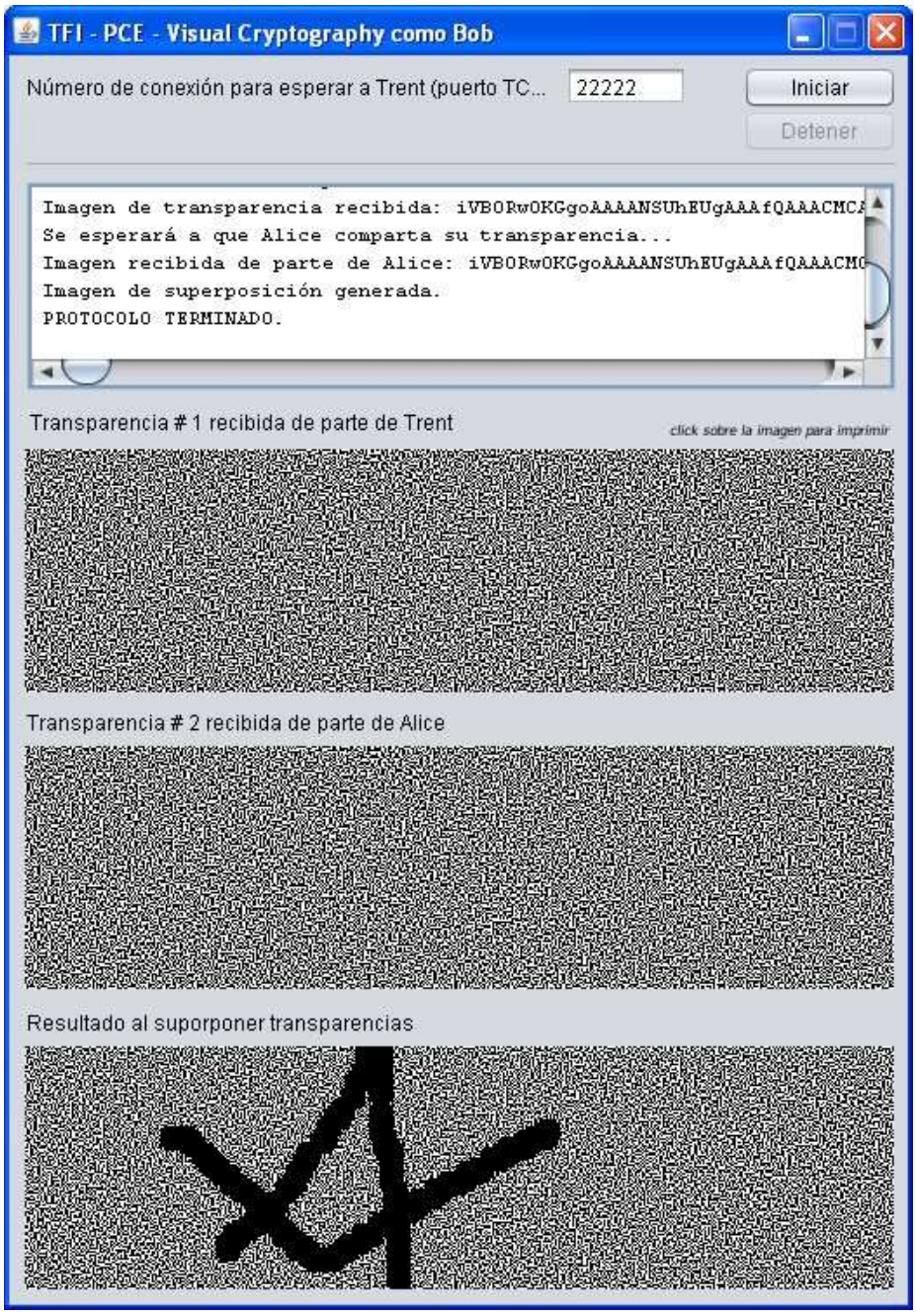


Nótese que el trazo del lápiz es grueso adrede, para luego reconocer fácilmente la imagen original (siempre algo distorsionada, al menos en sus bordes), inclusive al imprimir; funcionalidad que la aplicación permite haciendo *click* sobre las imágenes.

De manera similar al secreto compartido (de lo que en realidad esto sería una variante, como se comentó en el desarrollo del trabajo), al recibir Bob la imagen de parte Trent, debe decidir si compartirá con Alice o esperará a que ella comparta.



Habiendo Bob esperado entonces a que Alice comparta, una vez que ella lo haga, verá él algo similar a lo que puede observase en la captura siguiente:



5.8. Capturas de "MonedaAlAire" (Aplicación android de cara o seca justo)

Esta es la pantalla que se muestra al iniciar la aplicación; de esta manera entonces quedan presentadas las opciones: "Preguntar Cara o Seca vía SMS", que generará el número aleatorio y calculará su *hash* para enviarlo a la otra parte, y "Esperar recepción de SMS de pregunta", que, cómo seguramente se adivinó, esperará el SMS de una contraparte que haya elegido preguntar.

En la captura siguiente se muestra cómo, cuando el usuario eligió preguntar, se delega el envío del mensaje al sistema operativo. El *hash*, codificado, y precedido de un patrón fijo que la otra parte reconocería, estarán cargados en lo que sería el cuerpo del mensaje a enviar. Por otra parte, la selección del destinatario se haría de la misma forma que cuando se realiza el envío de un SMS estándar. Por supuesto, quien reciba el SMS deberá haber iniciado la aplicación previamente, y haber seleccionado "Esperar recepción de SMS de pregunta", lo que mostrará una pantalla como la que sigue a la siguiente.


11:33 PM
To
+++CARAOSECA?e588bc107
189f1bbc569c77655a4a50a
a7ea305d6033f035875b195
Send


11:35 PM
MonedaAlAire
Moneda al aire!
Esperando pregunta...

Una vez enviado el mensaje, la aplicación de envío del sistema operativo a la cual se delegó esta tarea se cerrará automáticamente, y quedarán visibles los detalles de los pasos llevados a cabo hasta el momento, de manera similar a como eran mostrados en la aplicación de escritorio, como puede verse en la captura siguiente:

El usuario que recibe la pregunta, por su parte, verá entonces una pantalla similar a la siguiente; como puede observarse bajo el cuadro de dialogo, también se detallan los pasos que se han realizado al momento, pero lo principal por supuesto es que debe contestar la pregunta que la contraparte –cuyo número de teléfono se ve en los detalles pero aparece informado también en el cuadro de dialogo- le hace: ¿Cara o Seca?

Pasada esta selección, se envían y verifican los valores de acuerdo a la explicado en el desarrollo del trabajo para este protocolo. Las pantallas finales, que para cada caso se muestran a continuación, indican el resultado del juego y permiten volver a Jugar.

Para el usuario que haya iniciado, o enviado la pregunta, no es necesaria verificación alguna, por lo que simplemente verá informado si ganó o perdió; en cambio, su contraparte, necesita confirmar o validar si el número aleatorio representa efectivamente el resultado (cara o seca es determinado de acuerdo a si el número aleatorio es par o impar, al igual que en la aplicación de escritorio). Por esto es que el usuario que respondió la pregunta, al ver el resultado, verá también, en caso de corresponder por supuesto, que el usuario que realizó la pregunta no ha hecho trampa.


11:39 PM
MonedaAlAire
Moneda al aire!
Tirando moneda...
- número aleatorio (16 bytes):
Resultado
¡GANASTE!
Volver a Jugar


11:39 PM
MonedaAlAire
Moneda al aire!
Esperando pregunta...
Resultado
¡PERDISTE!
* Se confirmó que la otra
parte no hizo trampa
Volver a Jugar

6. Bibliografía

1. MENEZES, Alfred J., VAN OORSCHOT, Paul C. y VANSTONE, Scott A. Handbook of Applied Cryptography. Editorial CRC Press. 5ta. Edición. EE.UU. 2001.

2. SCHNEIER, Bruce. Applied Cryptography. Editorial John Wiley & Sons, Inc. 2da. Edición. EE.UU. 1996.

3. BUCHMANN, Johannes. Cryptographic protocols. Notas de conferencia. EE.UU. 2002. Recurso online [http://www.cdc.informatik.tu-darmstadt.de/TI/Lehre/WS01_02/Vorlesung/krypto_proto/CryptoProtocols.pdf] accedido en Diciembre de 2011.

4. GOLDWASSER, Shafi y BELLARE, Mihir. Lecture Notes on Cryptography. MIT y Universidad de California. EE.UU. 2008. Recurso online [http://cseweb.ucsd.edu/~mihir/papers/gb.pdf] accedido en Diciembre de 2011.

5. P. Y. A. RYAN. The Computer Ate my Vote. Universidad de Newcastle. Inglaterra. 2006. Recurso online [http://www.cs.ncl.ac.uk/research/pubs/trs/papers/988.pdf] accedido en Diciembre de 2011.

6. BOHLI, Jens-Matthias, M#LLER-QUADE, J#rn y R#HRICH, Stefan. Bingo Voting: Secure and coercion-free voting using a trusted random number generator. Universidad de Karlsruhe. Alemania. 2006. Recurso online [http://eprint.iacr.org/2007/162.pdf] accedido en Diciembre de 2011.

7. CHAUM, David L. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Universidad de California, Berkeley. EE.UU. Recurso online [http://freehaven.net/anonbib/cache/chaum-mix.pdf] accedido en Diciembre de 2011.

8. STAJANO, Frank y ANDERSON, Ross. The Cocaine Auction Protocol: On The Power Of Anonymous Broadcast. Universidad de Cambridge. Inglaterra. Recurso online [http://www.cl.cam.ac.uk/~rja14/Papers/cocaine.pdf] accedido en Diciembre de 2011.

9. DINGLEDINE, Roger, MATHEWSON, Nick y SYVERSON, Paul. Tor: The Second-Generation Onion Router. The Free Haven Project y Naval Research Lab. EE.UU. Recurso online [http://www.onion-router.net/Publications/tor-design.pdf] accedido en Diciembre de 2011.

10. NAOR, Moni y SHAMIR, Adi. Visual Cryptography. Eurocrypt 94. Recurso online [http://www.wisdom.weizmann.ac.il/%7Enaor/PAPERS/vis.ps] accedido en Diciembre de 2011.

11. WIKIPEDIA. Cryptography. Recurso online [http://en.wikipedia.org/wiki/Cryptography] accedido en Diciembre de 2011.

12. WIKIPEDIA. Cryptographic protocols. Recurso online [http://en.wikipedia.org/wiki/Cryptographic_protocols] accedido en Diciembre de 2011.

13. WIKIPEDIA. Secret sharing. Recurso online [http://en.wikipedia.org/wiki/Secret_sharing] accedido en Diciembre de 2011.

14. WIKIPEDIA. Shamir's Secret Sharing. Recurso online [http://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing] accedido en Diciembre de 2011.
15. WIKIPEDIA. Trusted timestamping. Recurso online [http://en.wikipedia.org/wiki/Trusted_timestamping] accedido en Diciembre de 2011.
16. WIKIPEDIA. Commitment scheme. Recurso online [http://en.wikipedia.org/wiki/Commitment_scheme] accedido en Diciembre de 2011.
17. WIKIPEDIA. Undeniable signature. Recurso online [http://en.wikipedia.org/wiki/Undeniable_signature] accedido en Diciembre de 2011.
18. WIKIPEDIA. Blind signature. Recurso online [http://en.wikipedia.org/wiki/Blind_signature] accedido en Diciembre de 2011.
19. WIKIPEDIA. Oblivious transfer. Recurso online [http://en.wikipedia.org/wiki/Oblivious_transfer] accedido en Diciembre de 2011.
20. WIKIPEDIA. Key escrow. Recurso online [http://en.wikipedia.org/wiki/Key_escrow] accedido en Diciembre de 2011.
21. WIKIPEDIA. Zero knowledge proof. Recurso online [http://en.wikipedia.org/wiki/Zero-knowledge_proof] accedido en Diciembre de 2011.
22. WIKIPEDIA. Electronic voting. Recurso online [http://en.wikipedia.org/wiki/Electronic_voting] accedido en Diciembre de 2011.
23. WIKIPEDIA. Deniable encryption. Recurso online [http://en.wikipedia.org/wiki/Deniable_encryption] accedido en Diciembre de 2011.
24. WIKIPEDIA. Anonymous remailer. Recurso online [http://en.wikipedia.org/wiki/Anonymous_remailer] accedido en Diciembre de 2011.
25. WIKIPEDIA. Anonymous internet banking. Recurso online [http://en.wikipedia.org/wiki/Anonymous_internet_banking] accedido en Diciembre de 2011.
26. WIKIPEDIA. Ecash. Recurso online [http://en.wikipedia.org/wiki/Ecash] accedido en Diciembre de 2011.
27. WIKIPEDIA. Bitcoin. Recurso online [http://en.wikipedia.org/wiki/Bitcoin] accedido en Diciembre de 2011.
28. WIKIPEDIA. Bingo voting. Recurso online [http://en.wikipedia.org/wiki/Bingo_voting] accedido en Diciembre de 2011.
29. WIKIPEDIA. Prêt à Voter. Recurso online [http://en.wikipedia.org/wiki/Prêt_à_Voter] accedido en Diciembre de 2011.
30. WIKIPEDIA. Secure multiparty computation. Recurso online [http://en.wikipedia.org/wiki/Secure_multiparty_computation] accedido en Diciembre de 2011.
31. WIKIPEDIA. Trusted Computing. Recurso online [http://en.wikipedia.org/wiki/Trusted_Computing] accedido en Diciembre de 2011.

32. WIKIPEDIA. Onion routing. Recurso online [http://en.wikipedia.org/wiki/Onion_routing] accedido en Diciembre de 2011.

33. WIKIPEDIA. Tor (anonymity network). Recurso online [http://en.wikipedia.org/wiki/Tor_(anonymity_network)] accedido en Diciembre de 2011.

34. WIKIPEDIA. Visual cryptography. Recurso online [http://en.wikipedia.org/wiki/Visual_cryptography] accedido en Diciembre de 2011.