Una técnica de protección para agentes móviles contra estaciones (hosts) maliciosas

Ariel Waissbein[1,2]

[1] CoreLabs, Core Security Technologies
[2] Doctorado en Ingeniería Informática, Instituto Tecnológico de Buenos Aires (ITBA)[**]
Ariel.Waissbein [arroba] coresecurity [punto] com

Resumen Al ejecutar un agente en una estación controlada por terceros (posibles atacantes), el código del agente puede ser escrutado, y su funcionalidad conocida e incluso, pervertida; cosa que puede ser dañina para quien despliega al agente. En este trabajo presentamos un método para proteger agentes contra este tipo de amenazas basado en técnicas criptográficas seguras. Implementando nuestro método, quien despliega al agente puede fijar cuáles funcionalidades (o datos) del agente deben mantenerse confidenciales; a su vez puede especificar una condición bajo la cual es correcto y necesario develar esta información confidencial. Esto permite, por ejemplo, desarrollar agentes seguros en diversos escenarios interesantes sin la necesidad de requerimientos poco realistas de hardware o conectividad; como es el caso del problema del *purchase agent* ([1]).

Palabras clave: Secure triggers, seguridad de agentes, malicious host problem, privacidad, protección de software, reverse engineering.

1. Introducción

El uso de agentes móviles para efectuar cómputos distribuidos o transacciones en medios como Internet es tentador por la ubicuidad de esta red y el alto poder de cómputo que proporciona (vea, e.g., [2]). Sin embargo, ésto se ve limitado en muchas aplicaciones interesantes por problemas de seguridad, entre otros. Efectivamente, el dueño de una estación donde corre un agente no tiene garantías de que éste no afecte negativamente la seguridad de su sistema; y respectivamente, el dueño de un agente no puede asegurar que éste corra en las distintas estaciones sin ser corrompido, o sus detalles técnicos y secretos escrutados (vea, e.g., [3], [4], [5]).

[**] CoreLabs: Humboldt 1967, Cda. de Bs. As. (C1414CTU), Argentina. Tel./Fax: (54-11) 5032-2673.
ITBA: Av. Eduardo Madero 399 (C1106ACD), Cda. de Bs. As., Argentina

El primer problema ha sido estudiado extensivamente y solucionado en diversos contextos, e.g., por técnicas como el sandboxing ([6]), o la prometedora "trusted computing" (ver, e.g., [7]). De cualquier forma, en este trabajo nos interesa solamente el segundo problema, el de proteger a un agente de las estaciones en las que corre.

El problema de protección de agentes contra estaciones no confiables ha sido estudiado brevemente (ver, e.g., [1], [8], [9], [10]), y los resultados obtenidos son limitados tanto empírica como teóricamente (vea, e.g., [10]); cosa que se manifiesta en las escasas soluciones prácticas acutalmente en uso. La falencia de soluciones a nuestro problema no se debe solo a su falta de estudio, sino a que las propuestas conocidas pretenden resolver (instancias) del problema de ofuscación, que es sabidamente irresoluble ([11], cf. [12], [3]), o exigen requerimientos, como hardware seguro ([13]), o el uso de terceras partes confiables (trusted third parties, ver [1]), que son irreales para un alto número de aplicaciones (vea, e.g., el ejemplo a continuación).

Más en particular, nos ocupa el problema de privacidad de datos y código de agentes móviles que corren en estaciones no confiables. Los inconvenientes indicados en el párrafo precedente persisten en este subproblema. Aunque sí existen soluciones para el caso de aplicaciones particulares, como lo son la privacidad en bases de datos ([14], [15]), o los sistemas de autenticación para código móvil ([16]); ver también [17] y [18]. Pero el (sub-)problema general permanece abierto.

La intención de este trabajo es presentar una solución a este último problema aplicando las técnicas de gatillos seguros de [18]. Brevemente, somos capaces de aplicar los resultados de *op. cit.* para obtener una solución útil y de seguridad demostrable —pero relajando las hipótesis. El siguiente ejemplo debido a Hohl ([1]) ilustra pertinentemente nuestro esenario (ver también [17], [15]):

Un usuario quiere desplegar código móvil en Internet para recorrer las bases de datos de etailers[3] buscando un producto específico (marca, modelo, color, precio, etc.); requiere que quienes accedan al código del agente no sean capaces de obtener los detalles del producto buscado, ni su identidad, salvo en aquellas estaciones en las que una instancia del agente encuentra a este producto; en ese caso, el agente debe develarles la identidad de su dueño y comprar al producto (e.g., online).

Nótese que los requerimientos de privacidad que hacemos en este ejemplo son necesarios; de no existir, cualquier atacante podría conocer los gustos del usuario, obtener su número de tarjeta de crédito, o simplemen-

[3] Negocio que oferta y vende a través de Internet.

te, deducir cuál es el precio máximo que este usuario esta dispuesto a pagar por el producto y aprovechar esta información desfavoreciéndolo. Por otro lado, la versión relajada de privacidad propuesta —en la que se le permite obtener la información secreta sólo quienes acceden a la instancia del agente que ha encontrado al producto deseado— alcanza para cubrir la expectativas de un comprador. ¡Nuestra premisa principal es que estos requerimientos son realizables y a la vez útiles en términos de aplicaciones realistas!

En este trabajo, tomamos el problema de privacidad para agentes móviles bajo esta premisa. A saber, obtener un mecanismo por el cual el dueño de un agente puede establecer los comandos (o datos) del algoritmo que deberían ser confidenciales, y las condiciones bajo las cuales es correcto y necesario que dejen de serlo (vea la Sección 2 para más detalles). Presentamos una solución que protege algoritmos arbitrarios sin requerimientos *ad hoc* de hardware o terceras partes confiables.

Las técnicas de gatillos seguros (secure triggers) de [18], implementadas en [19], permiten mantener al código cifrado hasta el instante de su uso (ver Sección 3). Por ejemplo, para solucionar el problema del agente comprador descrito arriba, podemos construir un agente que analiza las descripciones de productos en sitios web una por una (en un formato estándar). En cada caso compara esta descripción con la descripción del producto buscado de manera que las comparaciones no dejan entrever información sobre el producto buscado. Una vez que las descripciones coinciden, el algoritmo *mágicamente* construye una clave simétrica, y descifra una funcionalidad escondida que se conecta al usuario y le pasa los datos del vendedor para que él efectúe la compra. En ese caso, y solo en ese caso, quien controla esta máquina puede acceder a toda la información secreta. (En la Sección 4 resolvemos un problema análogo, y damos detalles de la solución.)

2. Escenario y Resultados

Concentramos nuestros estudios en un escenario eminentemente práctico. Presentamos un proceso de protección que toma al algoritmo de un agente (e.g., el código fuente), y una descripción de "lo que se quiere proteger", y devuelve una versión protegida del agente (i.e., un ejecutable). En la Sección 4 enunciamos cuáles son los requerimientos para que nuestra protección sea efectiva, y describimos un ejemplo.

Asumimos que el atacante que controla estaciones en las que desplegamos al agente, no lo frena, y este será el caso en las aplicaciones que

consideremos (cf. Sección 4.1). Modelamos los atacantes por algoritmos que, en tiempo polinómico, intentan recuperar el secreto a partir del algoritmo que corre el agente desplegado.

Específicamente, sea dado el algoritmo para un agente móvil que queremos proteger (e.g., una sucesión de comandos en algún lenguaje fijo que conforman un algoritmo). Fíjese un número arbitrario de porciones de código de manera que cada vez que una porción se interseca con otra, sucede que una esta incluida en la otra. Por cada una de estas porciones, sea dada una condición (e.g., un predicado) sobre las variables del algoritmo. (En las aplicaciones, como en el ejemplo de la introducción, es el dueño del agente quien realiza este proceso de selección.)

Luego, somos capaces de computar una versión protegida del agente, i.e., un algoritmo con la misma funcionalidad (en términos de entradas y salidas) y con la propiedad que: el algoritmo corre normalmente ejecutando las porciones no protegidas, una vez que arriba a una porción protegida, evalua la condición asociada en los valores de las variables del algoritmo (e.g., la entrada), y descifra y ejecuta la porción protegida *siempre y cuando* ésta condición se verifique. Esta construcción se realiza mediante un algoritmo eficiente. Nuestro resultado principal (Sección 3) dice que un atacante con acceso irrestricto al agente protegido no puede obtener información semántica —en el sentido de [20]— sobre el código escondido.

Cabe aclarar, que este algoritmo de protección ha sido implementado en un marco compatible con el cíclo de desarrollo de un agente para lenguajes modernos como C, C++, o Java ([19]).

3. Gatillos Seguros (Secure Triggers)

En esta sección se introducen los "secure triggers" de [18] y [21] que forman parte integral de nuestras construcciones. Notamos que en los trabajos citados se utiliza el "UC framework" (o plataforma de composición universal; ver, e.g., [22]), mientras que en este manuscrito preferenciamos un tratamiento de seguridad *à la* indistinguibilidad. Por una cuestión de completitud describimos la funcionalidad de los gatillos seguros y reescribimos los resultados de seguridad. Referimos al trabajo original ([18]) al lector interesado en los resultados explícitos sobre el UC framework y las pruebas de seguridad.

Comenzamos por una definición formal. Sea dado un secreto $S \in \{0,1\}^*$ y un predicado $p \in P$. Un **gatillo** $T_{S,p}$ es un algoritmo que compu-

ta la siguiente función:

$$T_{S,p}(x) := \begin{cases} S & \text{si } p(x) = 1 \\ \perp & \text{si } p(x) = 0. \end{cases}$$

Una familia de gatillos $\{T_{S,p}\}_{p \in P}$ se dice **segura** si, no existe un atacante que recupere información semántica sobre el secreto S dados el algoritmo para el gatillo, la familia P, y el algoritmo de muestreo. Más precisamente, para todo atacante[4] $\mathcal{A}$, para todo polinomio univariado q, para todo par $S_1, S_2 \in \{0,1\}^*$ de tamaño $n \in \mathbb{Z}$, la probabilidad que tiene $\mathcal{A}$ de distinguir $T_{S_1,p}$ de $T_{S_2,p}$, dados $T_{S_1,p}$ y $T_{S_2,p}$, una descripción de la familia P, y el algoritmo de muestreo, es menor que $1/q(n)$ para valores de n suficientemente grandes.

En [18] los autores introducen a los gatillos seguros y construyen los primeros tres ejemplos que copiamos abajo, probando su seguridad; el cuarto ejemplo es debido a A. Juels y M. Sudan ([17]); y el quinto es nuevo. Estos ejemplos son seguros bajo hipótesis criptográficas estándar, i.e., existe un esquema de cifrado simétrico seguro contra ataque de texto plano elegido (ind-cpa). En cada caso, usaremos algoritmos de muestreo con distribución uniforme sobre la familia P. Fijemos, como parámetro de seguridad, a un entero positivo $r \in \mathbb{Z}$.

- El *gatillo simple* es un ejemplo elemental en el que familia de predicados P contiene todos los predicados de la forma $p_k(x) := 1$ si $x = k$, y $p_k(x) = 0$ sino, para $k \in \{0,1\}^r$ (ver Figura 2).
- El *gatillo de sub-sucesión* es de gran utilidad para aplicaciones. La familia P contiene a todos los predicados de la forma $p(x) = 1$, si x mirado como sucesión (finita) de bits $(x_i)_i$ contiene una sub-sucesión (secreta) $(y_{i_j})_{1 \le j \le r}$ dada, y $p(x) = 0$ sino. En la Fig. 1 abajo ilustramos con un ejemplo simplista para claves de $r = 4$ bits. El gatillo

x_1	x_2	...	x_{37}	...	x_{62}	x_{63}	x_{64}
.	$y_{i_1} = 1$	...	$y_{i_2} = 0$	...	$y_{i_3} = 0$	.	$y_{i_4} = 1$
.	$i_1 = 2$	...	$i_2 = 37$	...	$i_3 = 63$	.	$i_4 = 64$

Figura 1. Ejemplo simplista

devuelve el secreto si la entrada verifica $x_2 = 1, x_{37} = 0, x_{62} = 0$ y $x_{64} = 1$ —un atacante no puede conocer los índices i_1, i_2, i_3, i_4, ni los valores y_1, y_2, y_3, y_4 salvo que *rompa* la seguridad del gatillo.

[4] Máquina de Turing probabílistica que trabaja en tiempo polinómico.

- Dado un entero t, se define al gatillo *gatillo de t entradas múltiples* por la familia P que contiene a todos los predicados de la forma $p_{k_1,\ldots,k_t}(x) = 1$, donde $k_1, \ldots, k_t \in \{0,1\}^r$, si para todo $i, 1 \leq i \leq t$, existe una subpalabra x_i de x con $x_i = k_i$.
- Dados dos enteros positivos $t, s \in \mathbb{Z}$, con $s \leq t$, se define al *gatillo de bóvedas difusas con parámetros* s, t, tomando a P como la familia de predicados de la forma $p_{k_1,\ldots,k_t}(x^{(1)}, \ldots, x^{(t)}) = 1$ si al menos s de las entradas $x^{(1)}, \ldots, x^{(t)} \in \{0,1\}^r$ coinciden con s valores distintos dentro de $k_1, \ldots, k_t \in \{0,1\}^r$.
- El *gatillo de expresiones booleanas* que recibe una entrada x de un tamaño fijo (i.e., $x = (x_1, \ldots, x_t) \in \{0,1\}^t$) y devuelve el secreto siempre y cuando $f(x_1, \ldots, x_n) = 1$ donde f es una función booleana que se evalua por un circuito de tamaño acotado.

Para fijar ideas, mostramos como funciona el algoritmo para el gatillo simple (en la Figura 2); los algoritmos subyacentes a los otros ejemplos están detallados en [18]. Sea (G, E, D) un algoritmo de cifrado simétrico seguro contra textos-planos conocidos (ind-cpa secure, ver [23]). Supongamos fijos un secreto S y valor $k \in \{0,1\}^r$. Supongamos además que han sido computados los cifrados $E(k,0), E(k,S)$ de los valores $0 \in \{0,1\}^r$ y S. Es claro, que los gatillos simples son una instancia de gatillos segura;

almacenado: $E(k,0), E(k,S)$.
entrada: x.
salida: S o $\perp$.
compute $E(x,0)$;
If $E(x,0) = E(k,0)$[5]
then {**output** $D(x, E(k,S))$};

Figura 2. El gatillo simple

los otros ejemplos requieren algo más de atención y son analizados con justo detalle en [18].

4. Aplicaciones

Exponemos un paradigma para proteger agentes móviles de estaciones no confiables usando como herramienta principal a los gatillos seguros

[5] Nótese que es posible que $k \neq x$ y $E(x,0) = E(k,0)$, pero esto sucede solo con probabilidad despreciable en el parámetro de seguridad y luego puede ser desestimado.

descriptos en la sección anterior. El uso que proponemos se ve ilustrado en el ejemplo que mencionamos en la introducción y un segundo ejemplo que discutimos en detalle mas adelante.

Este paradigma es útil en el caso en que:

1. El agente corre en estaciones en las que el dueño del agente no puede confiar,
2. El agente contiene información (e.g., una funcionalidad o datos) que debe mantener confidencial para los observadores arbitrarios,
3. Hay una condición bajo la cual es correcto y necesario develar esta información confidencial,
4. El "meta requerimiento" de seguridad dice que esta condición debe *sumergirse* en una colección de condiciones suficientemente grande.

Las condiciones 1, 2 y 3 arriba no requieren mayor aclaración; no obstante, resulta prudente explicar en términos precisos que entendemos por "sumersión" en 4. Precisamente, que la condición de gatillo pueda ser descrita por alguno de los predicados en la Sección 3, y que la distribución de probabilidades inducida por la aplicación sea uniforme. (A grosso modo, la distribución de muestreo modela la información a priori del atacante.)

4.1. Una aplicación eminentemente práctica

Un hombre de negocios, que está por irse de vacaciones, quiere desconectarse completamente de su trabajo, pero mantenerse al tanto de las noticias para saber si hay alguna emergencia. Vale decir, necesita estar informado sólo en caso que ocurran ciertos cambios drásticos en la bolsa; por ejemplo, quiere saber si se verifica la condición de la Figura 3. Más aún, no puede confiar sus intenciones a un tercero por cuestiones de

DELL> 63 AND HPQ< 92 AND INTC> 221

Figura 3. La condición alarmante

negocios.

Usando nuestra técnica, podemos desarrollar un agente móvil que se conecta al sitio de la bolsa en Internet, revisa los valores de todas las acciones, y sólo en caso que se verifique la condición de alarma, descifra su número de pager y le avisa de esta alerta.

El agente es un Java applet "inocente" que simplemente descarga la base de acciones del sitio de NYSE cada media hora y chequea si se verifican las condiciones alarmantes. Para contrarestar denegaciones de servicio

(denial of service attacks), nuestro protagonista pude distribuir el agente en muchas máquinas y acceder a distintos *mirrors* del sitio. Éstos agentes implementan un gatillo de subsucesiones, definido en la Sección 3. Brevemente, el agente descarga una copia del estado instantáneo de la base de datos de NYSE; luego analiza cada línea de la base de datos separando nombre de valor (parsing); por cada alarma (i.e., conjunción de desigualdades), implementa un gatillo de expresiones booleanas que chequea si esta alarma ha ocurrido; en cuyo caso, llama al pager del hombre de negocios y le indica cuál es la alerta. Usamos al gatillo de subsucesiones, seleccionando como secreta a la porción del código relacionada con "la llamada al pager", y a la condición de gatillo como sigue. Por ejemplo, al verificar si se verifica la condición "DELL$>$ 63", por cada par (acrónimo, valor) en la base que acaba de bajar establece si ciertos bits del acrónimo de la entrada coinciden con los de la representación del acrónimo DELL, y para chequear si su valor es mayor a 63 debe chequear que los 7 bits menos significativos estén prendidos, y los restantes apagados[6].

El agente tardará milésimas de segundo en procesar la base de datos con unos cientos de entradas chequeando si se verifica la condición alarmante. Si un atacante quisiera atacar al agente y recuperar al procedimiento secreto, debe romper la seguridad del esquema de cifrado simétrico (esta es la reducción de [18]). En relación a los ataques de fuerza bruta que intentan adivinar cuál es la alarma, argumentaremos que dudosamente funcionan. Asumamos que la condición alarmante que describe el gatillo es la conjunción de 4 condiciones del tipo $x > valor$, supongamos que la base de acciones tiene unas 800 entradas, y que cada una de estas puede tener 500 valores; luego existen $(800 \cdot \ldots \cdot 797) \cdot 500^4 \approx 2^{74}$ posibilidades, lo que hace intratable al ataque de fuerza bruta.

5. Conclusiones

Las técnicas (teóricas) de gatillos seguros se implementan directamente a agentes móviles. En particular, en problemas prácticos como los que describimos.

Es claro que los resultados subyacentes de gatillos seguros no se aplican directamente a los ejemplos discutidos, e.g., en la práctica sucede que la información que tiene un atacante le permite reducir el espacio

[6] Hay diversos cambios que se pueden realizar en pos de mejorar la seguridad de este esquema. Por ejemplo, agregar a la conjunción alguna clausula que sea siempre cierta, de manera que los ataques de fuerza bruta necesiten tomar espacios mas grandes.

de muestreo. No obstante, hemos discutido –en el caso del problema de la Sección 4.1— que estas reducciones no son significativas y no permiten ataques tratables; y estos argumentos se generalizan (sin cambios) a otros ejemplos. Más precisamente, no se conocen ataques a las primitivas de cifrado que permitan aprovechar esta diferencia, más allá del ataque de fuerza bruta.

Por otro lado, cabe notar que esta es una limitación de la implementación que aplicamos de los gatillos seguros, y no de los gatillos *per se*. Resulta por demás interesante, estudiar implementaciones de gatillos seguros (como las cinco descriptas en la sección 3) con distribuciones no uniformes (cf. [24]).

Otra conclusión que queremos destacar es el hecho que el uso de gatillos (e.g., no simples) provee una mayor seguridad a problemas de la práctica. Por ejemplo, el usar al gatillo de subsucesiones en el problema de la sección anterior, nos exime de especificar explícitamente cuáles son las "condiciones alarmantes" y verificar desigualdades implícitamente. En ese sentido, resultaría útil tener nuevos ejemplos de gatillos seguros, así como estudiar la aplicación de los ejemplos existentes.

Agradecimientos:

El autor quisiera agradecer a A. Futoransky por los comentarios y discusiones en la elección de ejemplos.

Referencias

1. Hohl, F.: Time limited blackbox security: Protecting mobile agents from malicious hosts. In Vigna, G., ed.: Mobile Agents and Security. Volume 1419 of LNCS., Springer (1998) 92–113
2. Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@home: An experiment in public resource computing. Communications of the ACM (2002)
3. Devanbu, P.T., Stubblebine, S.G.: Software engineering for security: a roadmap. In: ICSE - Future of SE Track. (2000) 227–239
4. Hachez, G., Hollander, L.D., Jalali, M., Quisquater, J.J., Vasserot, C.: Towards a practical secure framework for mobile code commerce. In: Proceedings of the Third International Information Security Workshop (ISW 2000), Wollongong, Australia. Volume 1975 of LNCS. (2000) 164–178
5. Wang, C., Davidson, J., Hill, J., Knight, J.: Protection of software-based survivability mechanisms. In: The International Conference on Dependable Systems and Networks (DSN'01), Goteborg, Sweden (July, 2001), IEEE Press (2001) 193–205
6. Gong, L., Mueller, M., Prafullchandra, H., Schemers, R.: Going beyond the sandbox: An overview of the new security architecture in the java development kit 1.2. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems. (1997)

7. Mundie, C.: Trustworthy computing. Technical report, Microsoft Corp. (2003)
8. Sander, T., Tschudin, C.: Towards mobile cryptography. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, USA, IEEE Computer Society Press (1998)
9. Horne, B., Matheson, L., Sheehan, C., Tarjan, R.E.: Dynamic self-checking techniques for improved tamper resistance. In Sander, T., ed.: Security and Privacy in Digital Rights Management, ACM CCS-8 Workshop DRM 2001, Philadelphia, PA, USA, November 5, 2001. Volume 2320 of LNCS., Springer (2002)
10. de Carvalho Ferreira, L., Uto, N., Dahab, R.: Combining techniques for protecting mobile agents. In: Congreso Iberoamericano de Seguridad Informática, México D. F., Méjico (2003)
11. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In Kilian, J., ed.: Advances in Cryptology - CRYPTO 2001. Volume 2139 of LNCS., UCSB, Santa Barbara (CA), Springer (2001) 1–18
12. van Oorschot, P.C.: Revisiting software protection (invited talk). In Boyd, C., Mao, W., eds.: Information Security, 6th International Conference, ISC 2003. Volume 2851 of LNCS., Bristol, UK, Springer (2003) 1–13
13. Goldreich, O.: Towards a theory of software protection and simulation by oblivious rams. In: Proceedings of the nineteenth annual ACM conference on Theory of computing, ACM Press (1987) 182–194
14. Yee, B.: A sancturary for mobile agents. In Vitek, J., Jensen, C., eds.: Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Volume 1603 of Lecture Notes in Computer Science., Springer-Verlag (1999) 263–275
15. Bohannon, P., Jakobsson, M., Srikwan, S.: Cryptographic approaches to privacy in forensic dna databases. In Imai, H., Zheng, Y., eds.: Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings. Volume 1751 of Lecture Notes in Computer Science., Springer (2000) 373–390
16. Rezgui, A., Ouzzani, M., Bouguettaya, A., Medjahed, B.: Preserving privacy in web services. In: The 4th International ACM Workshop on Web Information and Data Management, Virginia, USA (2002)
17. Juels, A., Sudan, M.: A fuzzy vault scheme. In: Proceedings of IEEE Internation Symposium on Information Theory, Lausanne, Switzerland, IEEE Press (2002) 408–426
18. Futoransky, A., Kargieman, E., Sarraute, C., Waissbein, A.: Foundations and applications for secure triggers. To be published in ACM Transactions on Information and System Security (TISSEC) (2004)
19. Bendersky, D., Futoransky, A., Notarfrancesco, L., Sarraute, C., Waissbein, A.: Advanced software protection now. Corelabs Technical Report, available at `http://www.coresecurity.com/corelabs/projects/software_protection.php` (2003)
20. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences (JCSS) **28** (1982) 270–299
21. Futoransky, A., Kargieman, E., Sarraute, C., Waissbein, A.: Foundations and applications for secure triggers. Cryptology ePrint Archive, Report 2005/284 (2005) `http://eprint.iacr.org/`.
22. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, Proceedings,, Las Vegas, Nevada, USA, 14th-17th October 2001, IEEE Computer Society (2001) 136–145

23. Goldreich, O.: Foundations of Cryptography (Vol. 2). Cambridge University Press (2004)
24. Goldreich, O., Krawczyk, H.: Sparse pseudorandom distributions. In Brassard, G., ed.: CRYPTO 1989. Volume 435 of LNCS., Springer (1989) 113–127