D. Morales, and I. Agudo, "Prueba de concepto de Autoridad de Certificación usando Computación Segura Multiparte", *XIV Jornadas de Ingeniería Telemática*, pp. 50-53, 2019.
http://doi.org/10.26754/uz.978-84-09-21112-8
NICS Lab. Publications: https://www.nics.uma.es/publications

XIV Jornadas de Ingeniería Telemática.
JITEL 2019.
Universidad de Zaragoza.

Actas de las XIV Jornadas de Ingeniería Telemática (JITEL 2019), Zaragoza (España), 22-24 de octubre de 2019.

ISBN: X-XXXX-XX-XXX-X
DOI: XXXXXX

Prueba de concepto de Autoridad de Certificación usando Computación Segura Multiparte

Daniel Morales, Isaac Agudo.
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Edificio de Investigación Ada Byron, 29010
d.moralesescalera@lcc.uma.es, isaac@lcc.uma.es

***Resumen*—Este trabajo pretende analizar el paradigma de la Computación Segura Multiparte y sus posibles aplicaciones en el campo de la criptografía. Se plantea como modelo alternativo, más escalable y seguro al uso de módulos hardware de seguridad para aplicaciones que requieran de Terceras Partes Confiables. Concretamente, se ha integrado un protocolo de criptografía RSA multiparte con la librería *certbuilder*, para la creación de certificados X.509. De esta forma se asegura que la creación de los certificados raíz de la Infraestructura de Clave Pública se realiza de forma que la generación de claves y firma de éste se ejecute íntegramente sobre el sistema multiparte, con un modelo de tres partes que trabaja con circuitos aritméticos, sin que ninguna de ellas, de forma aislada, tenga posibilidad de comprometer la clave privada correspondiente. Para comprobar la viabilidad del sistema se han realizado pruebas de generación de certificados con diferentes longitudes de clave, siendo el proceso determinante la creación de las claves. Los elevados tiempos hacen que una aplicación como esta no sea asumible en otros escenarios, pero creemos que para el caso de la creación de los certificados raíz de una infraestructura de clave pública las garantías avanzadas de seguridad compensan el tiempo extra.**

***Palabras Clave*—Computación Segura Multiparte, Criptografía, RSA, Secreto Compartido, Autoridad de Certificación.**

I. INTRODUCCIÓN

Uno de los problemas a los que se enfrenta el modelo actual de Internet es la confiabilidad. Otros problemas como la confidencialidad o la integridad han sido resueltos gracias a la criptografía moderna, mediante protocolos criptográficos simétricos y asimétricos. Sin embargo, en contadas ocasiones, se necesita algún mecanismo que verifique la autenticidad de un actor en Internet.

Este problema se ha solucionado tradicionalmente mediante la integración de Terceras Partes Confiables, actores intermediarios que proporcionan algún tipo de servicio en un entorno no confiable. Sin embargo, esta solución traslada el problema, pues la autenticidad de la Tercera Parte Confiable puede no estar garantizada.

A. *Modelos de Certificación Jerárquica*

Para solucionar el problema presentado, aparecen los modelos de Certificación Jerárquica. Un certificado es un documento digital que identifica a un usuario como tal, proporcionando su información relevante y verificado por una autoridad de certificación. El modelo más extendido es el de certificados de clave pública del estándar X.509. Este tipo de certificado contiene la clave pública del agente a verificar, además de estar firmado digitalmente por una autoridad de certificación mediante su clave privada.

De esta forma, distintas entidades de certificación pueden certificarse unas a otras, dando lugar a una cadena de confianza jerárquica, en cuya cumbre se hallan las autoridades de certificación con potestad para auto-firmar sus propios certificados. Estos certificados se conocen como certificados raíz.

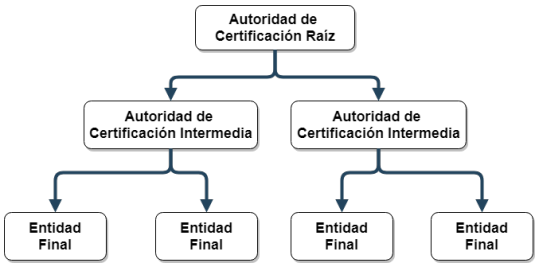

Figura 1. Jerarquía de Autoridades de Certificación

La problemática presentada en este trabajo es que dichas autoridades pueden suponer un punto simple de fallo, de manera que exponer su seguridad puede exponer la seguridad de toda la cadena de confianza que se construye bajo ellas.

B. *Módulos de seguridad hardware*

Una solución extendida para proteger los objetos criptográficos sensibles son los Módulos de Seguridad Hardware. Estos dispositivos protegen las claves dentro de un entorno seguro a nivel de hardware, lo que presenta una capa extra de seguridad ante una posible intrusión en el sistema.

Sin embargo, estos dispositivos pueden presentar diversos inconvenientes, como su elevado coste, con soluciones que parten de los 20000$ [1], además de la exposición a ataques de canal lateral o la pérdida de flexibilidad y escalabilidad. Es por esto que han surgido propuestas a nivel de software, con la problemática de que su nivel de seguridad no alcanza al de la solución hardware.

La propuesta de este trabajo es incluir la funcionalidad de un Módulo de Seguridad, a nivel de software y en un entorno descentralizado, mediante CSM (Computación Segura Multiparte). La ventaja de esta solución es que CSM garantiza la seguridad mediante modelos matemáticos probados.

Aunque las aplicaciones de CSM están a penas empezando a ver la luz, casos como el de Unbound Tech [2] corroboran la viabilidad de este tipo de soluciones. Su propuesta de gestión de claves mediante CSM ha obtenido los niveles de certificación 1 y 2 del estándar FIPS 140-2, que se encarga de acreditar los módulos criptográficos.

II. PARADIGMA DE LA COMPUTACIÓN SEGURA MULTIPARTE

A. *Origen y evolución*

CSM nace como un modelo para evaluar una función entre varios participantes, sustituyendo a una Tercera Parte Confiable por un protocolo, de forma que ninguno conozca los datos de entrada de los demás pero todos obtengan el resultado de la computación.

Sus inicios se remontan a la aparición del protocolo *Garbled Circuit* [3], basado en circuitos booleanos, aunque posteriormente surgieron otras opciones basadas en circuitos aritméticos.

B. *Modelos de seguridad*

A la hora de diseñar un protocolo multiparte se han de cumplir una serie de requisitos para garantizar la seguridad, como privacidad, exactitud, independencia de las entradas, etc. Para abordar estos requisitos se diseñan unos modelos genéricos que definen los umbrales máximos permitidos sobre los que se garantiza la seguridad de un protocolo [4].

B1. Modelo de comunicación: El modelo de comunicación determina las características de los canales por los que los participantes intercambian la información entre ellos. Estos canales pueden ser *unicast* o *broadcast*. Además, pueden asumirse como *seguros*, *autenticados* o *inseguros*. También se determina la temporalidad de la información, que se puede enviar de forma *síncrona* o *asíncrona*.

B2. Modelo de adversario: El objetivo de los adversarios es que la computación no finalice correctamente. Se clasifican en función de las capacidades de acción que tienen sobre el resto de los participantes de la computación. Un *adversario pasivo* es aquel que trata de obtener información de uno o más participantes, pero no puede desviar el flujo de acción natural del protocolo. Un *adversario activo*, por su parte, tiene total control sobre algún participante, por lo que puede modificar su funcionamiento respecto a la ejecución del protocolo. Además, puede ser *estático* o *dinámico* en función de si los participantes corruptos se definen previamente a la computación, o pueden variar a lo largo de la misma.

B3. Modelo de computación: Un protocolo de CSM se define sobre un lenguaje específico o modelo matemático sobre el que se sustentan las operaciones. Los modelos tradicionales de la literatura definen circuitos sobre *campos booleanos*, aunque cada vez es más habitual adoptar soluciones basadas en circuitos aritméticos, que se definen sobre *campos finitos* $(F, +, *)$. Lo bueno de este modelo es que cualquier función computable puede ser expresada como un circuito.

III. ESQUEMAS DE SECRETO COMPARTIDO

Los protocolos que trabajan con circuitos aritméticos suelen basarse en esquemas de secreto compartido. Estos algoritmos criptográficos permiten que un secreto S pueda ser compartido con diversos participantes de forma fragmentada, necesitando de la colaboración de varios de ellos para su reconstrucción. Por ello se conocen como esquemas de umbral (t, n), siendo t el mínimo de fragmentos necesarios para la reconstrucción y n el total de fragmentos existentes.

El caso más extendido de esquema de secreto compartido es el de Shamir [5]. Este esquema construye polinomios aleatorios para dividir el secreto en fragmentos y utiliza la interpolación de Lagrange para su reconstrucción. Aprovecha la propiedad de que para definir un polinomio de grado k se necesitan $k + 1$ puntos. La construcción de estos polinomios se presenta en la Ec.1, donde los $t - 1$ coeficientes $\{a_1, ..., a_{t-1}\}$ se eligen al azar, mientras que a_0 representa al secreto S.

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1} \qquad (1)$$

El polinomio asociado al secreto se utiliza para generar puntos de la función y a cada participante se le entrega el valor de un punto. Para reconstruir el secreto se interpola el polinomio con la fórmula de Lagrange (Ec.2) y se calcula su valor para el caso particular $x = 0$.

$$f(x) = \sum_{i=1}^{t} s_i \prod_{j=1}^{t, i \neq j} \frac{x - x_j}{x_i - x_j} \qquad (2)$$

A. *Suma y multiplicación*

Estas operaciones se definen como una serie de cálculos sobre los coeficientes de los polinomios que representan a los secretos compartidos.

En el caso de la suma se trata de una operación sencilla ya que no necesita compartir secretos adicionales. Para construir el polinomio suma que represente la suma de dos secretos, simplemente se suman los coeficientes de los polinomios previamente compartidos que tienen el mismo grado. En la Ec. 3 se presenta el polinomio $h(x)$ como resultado de sumar dos polinomios $f(x)$ y $g(x)$.

$$h(x) = (s_f + s_g) + (r_{1_f} + r_{1_g})x + ... + (r_{t-1_f} + r_{t-1_g})x^{t-1} \quad (3)$$

En el caso de la multiplicación se necesita compartir un secreto adicional, lo que eleva el coste de la computación. El motivo es que la multiplicación de dos polinomios de grado $t-1$ da como resultado un polinomio de grado $2t-2$ (Ec. 4), el cual ha de reducirse a grado $t-1$ para poder interpolarse. Para conseguir esto, cuando cada participante multiplica los dos fragmentos originales de los secretos construye con el resultado un nuevo polinomio aleatorio, el cual conduce a otra ronda de compartición de secretos.

$$h(x) = (s_f s_g) + r_1 x + r_2 x^2 + ... + r_{2t-2} x^{2t-2} \quad (4)$$

IV. VIRTUAL IDEAL FUNCTIONALITY FRAMEWORK

VIFF es un framework para desarrollar prototipos de aplicaciones CSM, escrito en Python. La comodidad de utilizar un framework como éste es que oculta la complejidad matemática subyacente. De esta forma, el desarrollador puede centrarse exclusivamente en la funcionalidad que desea implementar.

VIFF presenta una arquitectura de tres capas, en la que cada capa da servicio a la capa superior, siendo la capa de las operaciones CSM (basada en el esquema de Shamir) la que da servicio a las aplicaciones que se pretenden desarrollar, tal y como se muestra en la Fig. 2.

Figura 2. Pila de implementación de VIFF

Para implementar la capa de comunicación de red, VIFF hace uso de una librería de Python llamada *Twisted*, la cual facilita el desarrollo de modelos de comunicación asíncronos mediante un modelo dirigido por eventos.

Algunas consideraciones de seguridad del modelo desarrollado en VIFF son que el modelo de adversario es pasivo, por lo que puede corromper hasta un máximo de 1/2 de los participantes, y que está limitado computacionalmente a un tiempo polinómico.

V. RSA DISTRIBUIDO PARA INFRAESTRUCTURA DE CLAVE PÚBLICA

El objetivo final del trabajo consiste en integrar un protocolo RSA distribuido que implementa las funciones de generación, cifrado y firma, desarrollado en una tesis [6] e implementado en VIFF, con una librería para la generación de certificados de clave pública X.509.

Por simplicidad de integración, se ha optado por usar la librería *certbuilder* [1], que además de estar desarrollada también en Python, tiene un diseño modular que permite fácilmente engarzar los protocolos de CSM para RSA distribuido. Por consiguiente, tanto para generar las claves RSA como para realizar la firma, la librería *certbuilder* delega en un conjunto de nodos CSM (3 nodos para esta solución concreta).

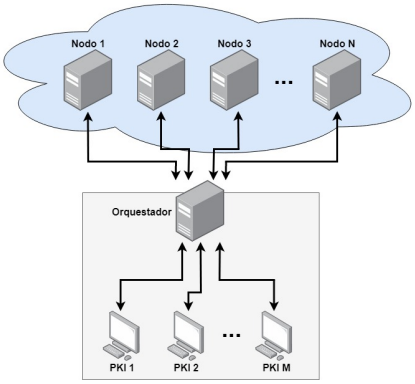

Figura 3. Arquitectura del sistema *certbuilder* multiparte

La topología resultante emula un sistema distribuido en la nube, al que el cliente accede desde una red externa mediante un Orquestador (Fig. 3). Este elemento intermedio es necesario para operaciones de coordinación entre los diferentes nodos, los cuales se encargan de realizar las operaciones CSM.

El trabajo realizado implementa la interfaz necesaria para que el cliente *certbuilder* pueda comunicarse de forma coherente con los nodos CSM, como puede apreciarse en la Fig. 4. Los nodos se despliegan en uno o varios entornos de prestadores de servicios, asignando tres de ellos para cada operación particular.

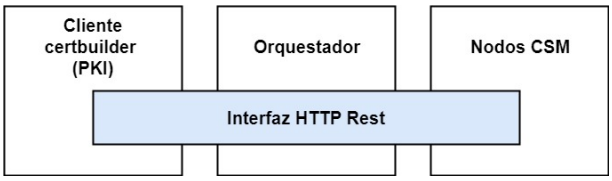

Figura 4. Integración de certbuilder con CSM

A. *RSA distribuido con VIFF*

Para la generación de las claves se producen varias rondas de generación de los parámetros correspondientes de forma distribuida (Fig. 5), de los cuales se va comprobando

[1] https://github.com/wbond/certbuilder

su validez por los requisitos de primalidad. Si en una determinada fase no se cumplen los requisitos, se vuelve a alguna fase anterior. Si por el contrario, se cumplen, se avanza de fase. Este proceso constituye el principal cuello de botella del protocolo, ya que la generación de los parámetros se hace de forma aleatoria y se ha de volver numerosas veces al inicio del procedimiento, con el gran coste computacional que ello supone por la inclusión del modelo CSM.

En la Ec. 5 puede apreciarse un ejemplo del parámetro público N, generado a partir de los valores privados de cada participante. La seguridad reside en la realización de las operaciones de suma y multiplicación de los parámetros privados mediante CSM, por lo que se necesita comprometer a más de un participante para conocer sus valores.

$$N = (p_1 + p_2 + p_3)(q_1 + q_2 + q_3) \tag{5}$$

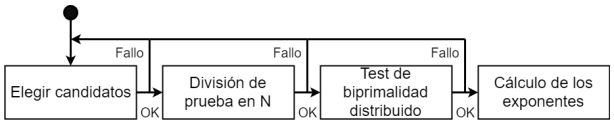

Figura 5. Fases del protocolo RSA distribuido

B. *Extensión CSM para certbuilder*

Certbuilder hace uso librerías criptográficas para la generación de los certificados. Para no inhabilitar el funcionamiento original de la librería, la solución propuesta ha integrado dos nuevas funciones implementadas en el código fuente, *generate_pair_mpc()* y *build_mpc()*. La primera función se encarga de solicitar al orquestador que inicie el procedimiento de generación de claves en los nodos, entregando como respuesta un identificador asociado al par de claves a generar. El procedimiento es asíncrono, y para solicitar la clave se necesita consultar al orquestador mediante un método HTTP adicional, cuya implementación en el cliente se ha dejado libre. La segunda función calcula el hash del certificado a firmar y se lo envía al orquestador, quien solicita a los nodos que posean la clave a emplear que firmen el hash de forma distribuida. Como resultado, el cliente obtiene el valor de la firma que ha de incorporar al certificado.

C. *Interfaz orquestador-nodos*

El orquestador coordina a los nodos necesarios para CSM. Pese a que no maneja información crítica en sí misma, presenta un punto de fallo respecto a gestión de acceso a las claves y privilegios. Este aspecto de seguridad se ha omitido en un primer momento, priorizando la funcionalidad del sistema y la seguridad ofrecida en la parte distribuida.

Para realizar varios procesos simultáneos se ha utilizado un sistema de gestión de hebras, donde cada una se encarga de generar un par de claves o de firmar un hash. Cada operación CSM se realiza vinculada a un puerto concreto en cada nodo, por tanto, el orquestador se encarga de solicitar a los nodos sus puertos disponibles, previamente al inicio de cada operación.

Respecto a la generación de las claves, los tres nodos invocan un script de Python en el que se ejecuta el protocolo RSA distribuido previamente expuesto. De esta forma generan los parámetros de las claves, que son guardados localmente en cada nodo. Al finalizar el proceso se genera la clave pública, que es el objeto que se entrega al cliente en formato ASN.1 DER.

El proceso de firma lo implementa otro script de Python basado en el mismo protocolo distribuido. En este caso, el orquestador envía el hash a los tres nodos correspondientes, que devuelven el valor de la firma obtenida.

Cuadro I
GENERACIÓN DE CLAVES RSA CON CSM [6].

Nº bits	Tiempo promedio	Ratio
256	0,97 min	3,83
512	3,78 min	3,89
1024	32,61 min	8,64
2048	120,87 min	3,71

VI. CONCLUSIONES

Pese a que el sistema está en desarrollo[2][3], se han obtenido resultados aceptables en pruebas realizadas en un entorno local.

La ventaja es que ofrece un modelo más escalable, mediante uso por demanda, que no necesita de largos tiempos de amortización del sistema.

Los elevados tiempos de generación de claves pueden ser aceptables para un modelo en el que CSM se utiliza solo en la capa de certificación raíz (Tabla I). Los tiempos de firma, por otro lado, no superan los diez segundos, lo que hace que, una vez generadas las claves, no suponga un coste adicional muy elevado.

Por ello, se contempla la opción de modificar el protocolo CSM empleado para tratar de obtener un sistema más ligero y versátil.

En conclusión, CSM ofrece un modelo de altas garantías de seguridad, pero a costa de grandes tiempos de computación. Es por ello que sus aplicaciones parecen tender a protocolos de computación ligera (tipo IOT) o a procesos críticos que no dependan en gran medidad del tiempo.

REFERENCIAS

[1] Logan Harbaugh. Thales nShield Connect offers enterprise-class key management. September 2009. Available: https://www.networkworld.com/article/2246758/thales-nshield-connect-offers-enterprise-class-key-management.html
[2] Lindell. Unbound receives FIPS 140-2 Level 1 and FIPS 140-2 Level 2 certification, May 2019. Available: https://www.unboundtech.com/unbound-receives-fips-140-2-certification/
[3] A. Chi-Chih Yao, How to Generate and Exchange Secrets (Extended Abstract), 1986, pp. 162-167.
[4] Hirt Martin, Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting, 2001.
[5] Shamir Adi, How to Share a Secret, November 1979.
[6] Mauland Atle, Realizing Distributed RSA using Secure Multiparty Computations, July 2009.

[2]https://github.com/dmoralesescalera/RSA-MPC-server
[3]https://github.com/dmoralesescalera/certbuilder