Implementation of a Distributed Algorithm for Threshold RSA Key Generation

Caterina Muñoz Vildósola
caterina@niclabs.cl
NIC Chile Research Labs
University of Chile

Abstract

In this work, we present an implementation of a distributed algorithm for threshold RSA key generation. The implemented algorithm allows to generate threshold RSA keys without a trusted dealer. Moreover, the algorithm is designed in such a way that none of the participants is capable of deducing the private key with their known information.

We provide a brief explanation of the algorithm in question and explain the corresponding implementation details.

1 Introducción

El sistema criptográfico propuesto por Rivest, Shamir y Adleman [RSA78], más conocido como RSA, es uno de los sistemas criptográficos más usados hoy en día, tanto para cifrado como para firmado.

Sin embargo, hay veces en las que RSA simple no es suficiente, y se debe usar RSA con umbral. En los sistemas criptográficos con umbral, la operación privada (descifrar, por ejemplo), está distribuida entre n participantes llamados nodos. Para realizar una operación privada exitosa se requiere de la participación de t de los n nodos. Esto último permite que algunos de los nodos fallen. Por ejemplo, si se tiene un sistema de criptografía umbral con tres nodos y umbral dos, se debe contar con dos de los tres nodos para realizar una operación privada. Uno de los nodos puede dejar de funcionar y el sistema sigue siendo operativo.

Un problema complejo en un sistema RSA con umbral es la generación de llaves. Es muy difícil generar las llaves de manera tal que ningún participante tenga suficiente información para vulnerar el sistema, como la factorización del módulo RSA o los valores de las llaves privadas.

En el artículo [BF01], Boneh y Franklin propusieron un algoritmo que logra generar llaves RSA para criptografía umbral de manera distribuida y segura. En este trabajo, introducimos una implementación del algoritmo propuesto por Boneh y Franklin.

2 El Algoritmo

2.1 Parámetros de Entrada

Hay algunos parámetros que se deben entregar al algoritmo para que se pueda ejecutar: La cantidad de nodos que participarán: n, con $n \geq 3$, el parámetro umbral: t, con $t > \frac{n}{2}$, el tamaño en bits del módulo RSA: b y el exponente público: e.

2.2 Resultados del Algoritmo

Al terminar la ejecución del algoritmo, se tiene lo siguiente:

- Una llave pública RSA compuesta por el exponente público definido anteriormente (e) y el módulo RSA N. N debe ser el producto de dos números primos y tener un tamaño de a lo más b bits y a lo menos $b-4$ bits.
- Una llave privada RSA d distribuida entre los n nodos participantes. La llave está distribuida de tal manera que se puede realizar criptografía umbral (basta que un subconjunto cualquiera de t nodos realice la operación privada parcial para poder realizar una operación privada exitosa). Así, el nodo $i-$ésimo guarda la llave privada parcial d_i.

2.3 Descripción General del Algoritmo

La siguiente es una descripción de muy alto nivel de los pasos que sigue el algoritmo. La descripción detallada del algoritmo está en el artículo de Boneh y Franklin [BF01].

1. El nodo i-ésimo elige p_i y q_i. Al hacer esto, se debe tener en cuenta el tamaño deseado del módulo RSA N

2. Los nodos usan sus p_i y q_i para calcular un N candidato de acuerdo a la siguiente fórmula:

$$N = p \cdot q = \left(\sum_{i=1}^{n} p_i\right) \cdot \left(\sum_{i=1}^{n} q_i\right)$$

   Este cálculo se realiza sin exponer los p_i y q_i gracias a una versión simplificada de la técnica BGW [BOGW88].

3. Los nodos le realizan un test de biprimalidad a N. Si N es biprimo (producto de dos primos), se avanza al paso siguiente; si no, el algoritmo vuelve a comenzar desde el paso 1. Cabe destacar que no es necesario calcular p ni q explícitamente para determinar si N es producto de dos primos. El test de biprimalidad está especificado y probado en el artículo [BF01].

4. A partir de N y e se calcula un conjunto de llaves preliminar $\{d'_i\}$. Estas llaves cumplen que:

$$d = \sum_{i=1}^{n} d'_i$$

   El valor d sería la llave privada en un sistema RSA simple con el mismo módulo N y el mismo exponente público e. Con este conjunto preliminar no se puede hacer criptografía umbral.

5. Usando el conjunto $\{d'_i\}$, se calcula el conjunto final de llaves $\{d_i\}$. A diferencia del conjunto preliminar, este conjunto final sí permite realizar criptografía umbral. La técnica que se ocupa para lograr el comportamiento umbral viene del trabajo de Rabin expuesto en el artículo [Rab98].

La figura 1 ilustra el orden en el que ocurren los pasos del algoritmo.

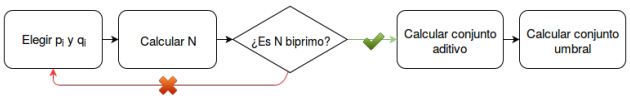

Figure 1: Esquema de los Pasos del Algoritmo

3 Detalles de Implementación

Luego de especificar el algoritmo implementado, se procedió a implementar la solución. En esta sección se describe cómo se llevó a cabo la implementación y algunos detalles particularles del software.

3.1 Módulos de Pasos

Cada uno de los pasos del algoritmo explicados en la sección 2.3 está implementado en un módulo. A su vez, cada módulo contiene funciones que ejecutan las tareas necesarias para completar cada paso. Muchas de esas funciones son distribuidas e involucran que los nodos intercambien mensajes entre ellos para poder obtener el resultado de la función. Además, cada una de las tareas de la función se ejecuta en un proceso distinto.

Así, si una función involucra calcular un valor, mandarle ese valor al resto de los nodos, recibir los valores respectivos del resto de los nodos y luego calcular un resultado final, todas esas tareas se ejecutan en procesos distintos. Cuando las tareas se deben ejecutar serialmente, se lanza un proceso para la primera tarea y ese proceso se encarga de lanzar un proceso para la segunda tarea antes de terminar su ejecución.

Cuando una de las funciones distribuidas se ejecuta, se debe ejectuar en todos los nodos del sistema a la vez. Los nodos trabajan cooperativamente para lograr resultados conjuntos utilizando estas funciones.

3.2 Módulo del Algoritmo Completo

El módulo del algoritmo completo se encarga de componer las funciones principales de los módulos de los pasos (explicados anteriormente en la sección 3.1) para obtener el flujo representado en la figura 1.

Para lograr lo anterior, tiene dos funciones: una función auxiliar que se encarga de generar un `N` candidato y una función principal recursiva que implementa el *loop* de la figura 1 y el comportamiento que sigue al *loop*.

3.3 Capa de Comunicación

El módulo de comunicación es el que está encargado de administrar toda la comunicación que ocurre entre los nodos y un cliente que es el que solicita la generación de llaves. Se eligió ocupar un modelo de comunicación central en el que toda la comunicación entre nodos es vía el cliente.

Dado lo anterior, este módulo tiene funciones para que los nodos puedan enviar y recibir mensajes y para que el cliente pueda distribuir correctamente los mensajes entre los nodos.

La figura 2 explica cómo fluje la comunicación entre procesos y nodos.

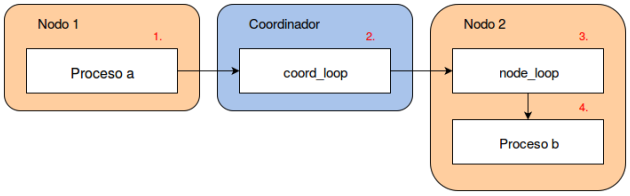

Figure 2: Flujo de Comunicación entre procesos

1. El proceso `a` del nodo 1 le quiere enviar un mensaje al proceso `b` del nodo 2. Ocupa la función `send_to_node`, especificando el nodo al que le quiere enviar el mensaje e incluyendo el nombre del proceso destinatario dentro del mensaje.
2. La función le envía un mensaje al proceso que está corriendo `coord_loop` en el cliente.
3. El cliente le redirige el mensaje al proceso que está corriendo la función `node_loop` en el nodo destinatario.
4. El proceso de `node_loop` le envía el mensaje al proceso destinatario, que está indicado dentro del mensaje.

Así, el cliente se encarga de determinar a qué nodo entregarle los mensajes y dentro de cada nodo, el proceso que ejecuta la función `node_loop` se encarga de determinar a qué proceso del nodo entregarle los mensajes.

4 Resultados

Primero, se realizaron varias pruebas del algoritmo completo para comprobar que estuviera correctamente implementado. Se verificó que todos los nodos llegaran al mismo valor de N, que con todas las combinaciones posibles de t llaves privadas se obtuviera el mismo valor de d y que el valor de d obtenido fuera equivalente al de un sistema RSA normal con módulo N y exponente públio e. Además, cuando el tamaño lo permitía, se comprobó que el módulo N obtenido fuera producto de dos primos. Todas las verificaciones anteriores fueron positivas, por lo que se concluyó que la implementación del algoritmo está correcta.

Además en cada experimento se midió la cantidad de intentos de generación de módulo N que se debieron realizar. Luego, se calculó el promedio de intentos agrupados por tamaño de módulo y se comprobó con un valor esperado. Dicho valor esperado se obtuvo del teorema de los números primos [CP05], que indica que se deben realizar en promedio $(\frac{b}{2}\ln 2)^2$ intentos para obtener un módulo biprimo (donde b es el tamaño deseado del módulo). El gráfico de la Figura 3 muestra el resultado obtenido, donde se puede observar que el promedio de intentos obtenido fue mucho menor al esperado.

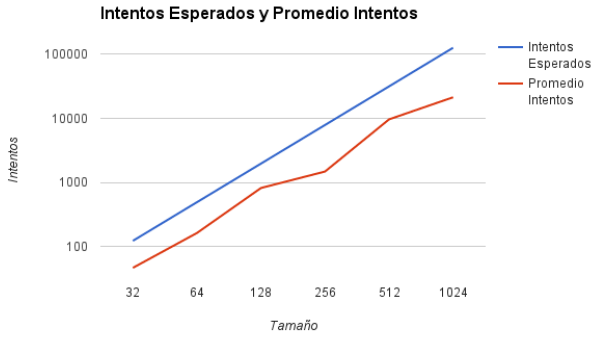

Figure 3: Comparación entre Promedio Obtenido y Promedio Esperado

5 Trabajo Futuro

- Implementar las optimizaciones del algoritmo propuestas en el artículo [BF01].
- Agregar una capa de seguridad que cifre el tráfico de datos entre los nodos. Para esto, lo ideal sería usar cifrado simétrico (donde todos los nodos comparten una llave).
- Probar un esquema distinto de comunicación.

Referencias

[BF01] Dan Boneh and Matthew Franklin. Efficient generation of shared rsa keys. *Journal of the ACM (JACM)*, 48(4):702–722, July 2001.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault tolerant distributed computation. In *Proc. STOC*, pages 1–10, 1988.

[CP05] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective, 2nd ed.* Springer-Verlag, 2005.

[Rab98] T. Rabin. A simplified approach to threshold and proactive rsa. In *Advances in Cryptology — CRYPTO*, pages 89–104, 1998.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.