UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

ALGORITMOS DE ENCRIPTACIÓN DE CLAVE ASIMÉTRICA

Trabajo final integrador de

Especialización en Redes de Datos

Autor: Mario Rodrigo Franchi

Director: Lic. Paula Venosa

Octubre 2012

INDICE

"Be careful what you water your dreams with. Water them with worry and fear and you will produce weeds that choke the life from your dream. Water them with optimism and solutions and you will cultivate success. Always be on the lookout for ways to turn a problem into an opportunity for success. Always be on the lookout for ways to nurture your dream."

Lao Tzu

Introducción

La criptografía es un tema que ha fascinado y ocupado a la humanidad desde la época de los egipcios. La transmisión segura de la información ha encontrado una gran variedad de aplicaciones en las áreas política, militar y económica. Hoy más que nunca, en un mundo caracterizado por la ubicuidad de Internet y basado fuertemente en datos digitales, se hace imperiosa la necesidad de garantizar la seguridad de estos; dado que la información confidencial puede quedar expuesta a ser vista o manipulada por terceros no deseados.

La criptografía asimétrica (o de clave pública) es una tecnología que se ocupa de la comunicación sobre redes abiertas, transformándose en la piedra fundamental de todos los negocios que se basen en transacciones electrónicas.

La criptografía de clave pública se ancla en la dificultad de resolver ciertos problemas matemáticos y en las funciones de una sola vía. Los dos problemas más conocidos, y que han ganado la mayor aceptación, son el de la factorización de números enteros y el de la solución del logaritmo discreto. Ante la omnipresente amenaza de un ataque al sistema criptográfico, la pregunta que surge es: ¿Se puede evitar por completo la ruptura de clave? La respuesta lamentablemente es NO. Lo único que se puede hacer es desarrollar métodos que permitan aumentar la protección de los sistemas criptográficos haciéndolos seguros por más tiempo. Con esto conseguimos dificultar cada vez más la ruptura de la clave pero no evitarla por completo. Vienen a colación las palabras de Edgar Allan Poe: *"Es dudoso que el género humano logre crear un enigma que el mismo ingenio humano no resuelva"*.

El presente trabajo está estructurado de la siguiente manera. En el capítulo 1 se introducen las bases matemáticas y criptográficas necesarias para el entendimiento de los capítulos posteriores. El segundo capítulo presenta a la criptología y al criptoanálisis, este último como una rama de la primera, detallando en particular el funcionamiento de la criptografía simétrica y la criptografía asimétrica. El capítulo 3 está dedicado a la descripción de los algoritmos de clave pública que actualmente son más utilizados. Cabe mencionar que esta es un área de la ciencia con constantes avances, no obstante los algoritmos más seguros son siempre los que más tiempo han estado expuestos al dominio público y que aún no han podido ser atacados exitosamente. El capítulo 4 es una revisión resumida de las aplicaciones más relevantes de los algoritmos de clave pública. Se diseca el funcionamiento de un protocolo emblemático, como es SSH, a través de un análisis con Wireshark de la secuencia de establecimiento de sesión. El quinto capítulo expone algunos puntos de comparación ente las distintas variantes de la criptografía de clave pública. Para finalizar, en el capítulo 6 se presentan las conclusiones.

Capitulo 1

Principios Matemáticos

"Lo importante no es lo que han hecho de nosotros, sino lo que hacemos con lo que han hecho de nosotros."
Jean-Paul Sartre

1 Principios Matemáticos

Para lograr un entendimiento de los algoritmos de cifrado, tanto de clave pública (asimétricos) como de clave privada (simétricos) se hace necesaria una somera introducción a algunos conceptos básicos de ciertas áreas de la matemática, como son la teoría de números y el algebra lineal. El objetivo de este capítulo es presentar estos temas para facilitar la comprensión y lectura de los capítulos subsiguientes. No es intención dar una demostración formal de cada teorema, sino exponer los conceptos básicos de los mismos. Una buena fuente de consulta es [FOROUZAN 2007].

1.1 Aritmética Entera

Si bien este tipo de aritmética nos es familiar, se hace una revisión que servirá de punto de comparación con la aritmética modular.

Los números enteros se definen como el conjunto de los números **$\mathbb{Z}$={...,-2,-1,0,1,2,3,...}.** Dentro de este conjunto está el subconjunto de los números naturales, $\mathbb{N}$ = {1,2,3,4,...}. Es decir, el subconjunto de los números enteros positivos (mayores que 0).

Operaciones binarias

Una operación binaria toma dos entradas y genera una salida. Se pueden definir tres operaciones binarias para enteros: suma, resta y multiplicación. Estas operaciones toman dos elementos del conjunto de enteros (*a*,*b*) y crean un elemento (*c*) en el mismo conjunto. Cabe destacar que la división no cumple con este requisito dado que produce dos salidas (cociente y resto).

Suma: -1 + 5 = 4
Resta: 1 -3 = -2
Multiplicación: 2 .-3 = -6

División Entera

Veamos ahora el algoritmo de la división entera. Si dividimos un número *a* (el dividendo) por *n* (el divisor) obtenemos dos números: *q* (el cociente) y *r* (el resto).

$$\mathbf{a = q \cdot n + r}$$

Ejemplo:

```
10 |  3        a = 10    n = 3
 9   3        q = 3     r= 1
 1
```

En criptografía se imponen dos restricciones a la división:

1. El resto es un entero no negativo, $r \geq 0$.
2. El divisor es un entero positivo, $n > 0$.

Divisibilidad

Dados dos números enteros *a* y *n* (con *n* distinto de 0), se dice que *n* divide a *a*, y lo escribimos como $n|a$, si existe un $q \in \mathbb{Z}$ tal que $a = q \cdot n$.

También se dice que *n* es un factor o divisor de *a*, y que *a* es un múltiplo de *n*. Si el resto no es cero, entonces *n* no divide a *a* y la relación se representa como $n \nmid a$.

Algunas propiedades derivadas de la definición anterior:

- Si $a|1 \rightarrow a = \pm 1$.
- Si $a|b$ y $b|a \rightarrow a = \pm b$.
- Cualquier $b \neq 0$ divide a 0.
- Si $a|b$ y $b|c \rightarrow a|c$.
- Si $b|g$ y $b|h \rightarrow b \mid (m \cdot g + n \cdot h)$ para cualquier m y n

El algoritmo de Euclides

Dados dos enteros *a* y *b* distintos de 0, decimos que el entero $d > 1$ es un máximo común divisor, o mcd, de *a* y *b* si d|a, d|b y para cualquier otro $c \in \mathbb{Z}$ tal que c|a y c|b, entonces c|d. Es decir, *d* es un entero positivo, divisor común de *a* y *b*, y cualquier otro divisor común es también un divisor de *d*. Con estas condiciones, el máximo común divisor es único. Se denota como d = mcd(a,b).

Sean *a*,*b* enteros no nulos. Entonces mcd(a,b) = mcd(b,r) donde *r* es el único $0 \leq r < b$ tal que existe un entero *q* con $a = bq + r$ (es decir, que *r* es el resto de la división de *a* por *b*).

Esta proposición nos indica que es igual de válido calcular el mcd(a,b) que el mcd(b,r), con la ventaja de que *r* es un entero de menor tamaño que el original *a*. Esto se aprovecha en el algoritmo de Euclides para el cálculo del máximo común divisor de dos números enteros.

Para calcular el mcd de dos enteros *a* y *b* (ambos > 0, suponemos a > b) se definen *q* y *r* recursivamente mediante las ecuaciones:

$a = bq_1 + r_1 \quad (0 \leq r_1 < b)$

$b = r_1 q_2 + r_2 \quad (0 \leq r_2 < r_1)$

$r_1 = r_2 q_3 + r_3 \quad (0 \leq r_3 < r_2)$

....

$r_{k-3} = r_{k-2} q_{k-1} + r_{k-1} \quad (0 \leq r_{k-1} < r_{k-2})$

$r_{k-2} = r_{k-1} q_k \quad (r_k = 0)$

Y de la proposición anterior, tenemos que:

$\text{mcd}(a,b) = \text{mcd}(b,r_1) = \text{mcd}(r_1,r_2) = \ldots = \text{mcd}(r_{k-2},r_{k-1}) = r_{k-1}$

El algoritmo extendido de Euclides

La extensión del algoritmo de Euclides es de aplicación en algunos algoritmos de cifrado como RSA. Este algoritmo además de dar el mcd, calcula al mismo tiempo y con la misma cantidad de pasos otros dos números enteros, ***x*** e ***y***, tal que:

$$x \cdot a + y \cdot b = mcd\ (a,b)$$

Una posible implementación en pseudocódigo es la siguiente:

```
r1 = a;  r2 = b;
x1 = 1;  x2 = 0;
y1 = 0;  y2 = 1;

while (r2 > 0)
{
        q = r1 / r2;

        r = r1 –q . r2;
        r1 =r2;   r2= r;

        x = x1 –q . x2;
        x1 = x2;   x2= x;

        y = y1 –q . y2;
        y1 = y2;   y2= y;
}
```

Los valores de r_1 y r_2 son los mismos que en el caso anterior. Inicialmente r_1 y r_2 toman los valores de a y b, respectivamente. Las variables x_1 y x_2 se inicializan a 1 y 0 respectivamente. Las variables y_1 e y_2 se inicializan a 0 y 1 respectivamente. Los cálculos de r,x e y son semejantes con una salvedad: aunque r es el resto de dividir r_1 por r_2 no existe la misma relación con x e y. Se calcula solo un cociente q ($r_1 \mid r_2$) que se utiliza también para los cálculos de x e y.

Ejemplo: Calcular el mcd (456,138) y los valores de x e y de acuerdo al pseudocódigo dado:

q	r_1	r_2	r	x_1	x_2	x	y_1	y_2	y
3	456	138	42	1	0	1	0	1	-3
3	138	42	12	0	1	-3	1	-3	10
3	42	12	6	1	-3	10	-3	10	-33
2	12	6	0	-3	10	-23	10	-33	29
	6	0		10	-23		-33	29	

(10 . 456) + (-33 . 138) = 4560 – 4554 = 6

1.2 Aritmética Modular

El algoritmo de la división generaba dos salidas, el cociente q y el resto r. En la aritmética modular solamente nos interesamos en el resto. Es decir que podemos transformar el algoritmo de la división en una operación binaria.

El módulo

Este operador binario se denomina *operador módulo* y se representa con *mod.* Como entradas al operador binario tenemos un número del conjunto de los enteros ($\mathbb{Z}$) y otro valor llamado módulo (*n*). La salida es el residuo.

a mod n = r

Dos enteros *a* y *b* se dice que son ***congruentes módulo n*** si (a mod n) = (b mod n). Esto se expresa como: a ≡ b (mod n). Hay que tener en cuenta que se puede utilizar el operador *mod* tanto como operador binario como relación de congruencia. Observar que en este último caso no se obtiene un significado de igualdad. (El operador igualdad es uno-a-uno, el operador congruencia es muchos-a-uno)

- 56 ≡ 11 (mod 15) → 56 mod 15 = 11 y 11 mod 15 = 11
- 32 ≡ -8 (mod 10) → 32 mod 10 = 2 y -8 mod 10 = 2
- -8 ≡ 2 ≡ 12 ≡ 22 (mod 10)

Conjunto de Residuos: $\mathbb{Z}_n$

El operador módulo con módulo *n* genera siempre un valor entero comprendido entre 0 y n-1, o sea $0 \leq r \leq n-1$. Es decir el operador módulo genera un conjunto circular $\mathbb{Z}_n$, donde una vez alcanzado el valor máximo vuelven a repetirse los mismos valores.
Algunos ejemplos pueden ser:

- $\mathbb{Z}_3 = \{0,1,2\}$
- $\mathbb{Z}_7 = \{0,1,2,3,4,5,6\}$
- $\mathbb{Z}_n = \{0,1,2,3,4,5,6,7,8,\ldots\ldots, (n-1)\}$

Operaciones binarias en $\mathbb{Z}_n$

Las tres operaciones que se aplican al conjunto $\mathbb{Z}_n$ son: suma, resta y multiplicación. El resultado se mapea al conjunto $\mathbb{Z}_n$ con el operador mod. El conjunto de origen puede ser tanto $\mathbb{Z}_n$ como $\mathbb{Z}$.

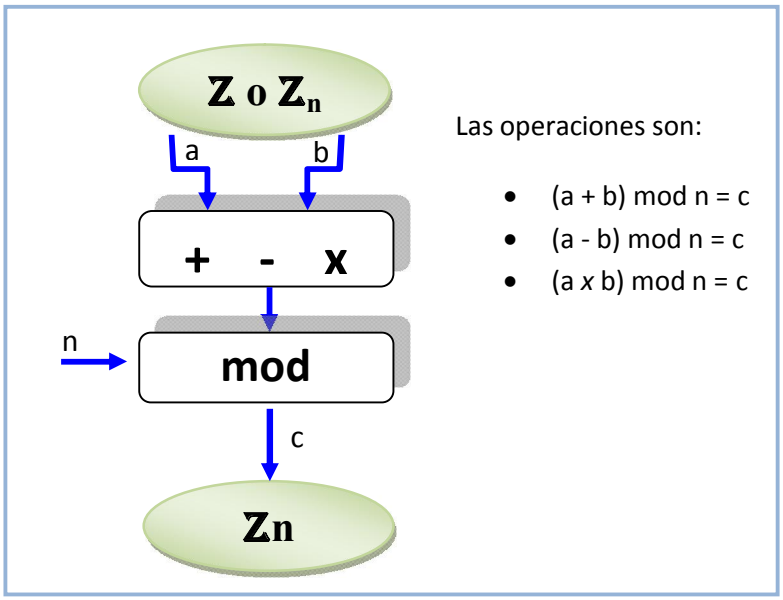


Figura 1 – Operaciones binarias en $\mathbb{Z}_n$

Inversos aditivos y multiplicativos

El ***inverso aditivo*** en la aritmética modular, se define de tal manera que la suma de un entero y su inverso aditivo es congruente a 0 mod n. Todos los enteros poseen un inverso aditivo, y lo podemos expresar como:

$$\mathbf{a + b \equiv 0 \ (mod\ n)}$$

Si estamos en $\mathbb{Z}_n$ el inverso aditivo de *a* se calcula simplemente como b = n –a.

El ***inverso multiplicativo*** en la aritmética modular, se define de tal manera que la multiplicación de un entero y su inverso multiplicativo es congruente a 1 mod n. No todos los enteros poseen un inverso multiplicativo. Para que lo tenga se debe cumplir que *a* y *n* sean primos relativos, es decir mcd (n,a) = 1. Lo podemos expresar como:

$$\mathbf{a \,.\, b \equiv 1 \ (mod\ n)}$$

Es posible hallar el inverso multiplicativo aplicando el teorema extendido de Euclides:

$$\mathbf{(x \,.\, a) + (y \,.\, b) = mcd\ (a, b) = 1}$$

El valor de *y* es el inverso multiplicativo de *b* en $\mathbb{Z}_n$.

En las siguientes tablas se pueden ver las relaciones de suma y multiplicación en $\mathbb{Z}_9$. Cada elemento de la tabla suma tiene su inverso resaltado en celeste; con el mismo color se resaltan los inversos en la tabla de multiplicación donde se observa que no todos los enteros tienen su correspondiente inverso.

+	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8	0
2	2	3	4	5	6	7	8	0	1
3	3	4	5	6	7	8	0	1	2
4	4	5	6	7	8	0	1	2	3
5	5	6	7	8	0	1	2	3	4
6	6	7	8	0	1	2	3	4	5
7	7	8	0	1	2	3	4	5	6
8	8	0	1	2	3	4	5	6	7

X	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8
2	0	2	4	6	8	1	3	5	7
3	0	3	6	0	3	6	0	3	6
4	0	4	8	3	7	2	6	1	5
5	0	5	1	6	0	7	3	8	4
6	0	6	3	0	2	3	0	6	3
7	0	7	5	3	1	8	6	4	2
8	0	8	7	6	5	4	3	2	1

En criptografía a veces es necesario que un número tenga su inverso para poder usar uno de ellos para cifrar y el otro para descifrar. Con el inverso aditivo no existen problemas dado que todos los elementos de $\mathbb{Z}_n$ tienen el suyo. Pero en el caso de necesitar la existencia del inverso multiplicativo se debe definir un nuevo conjunto, $\mathbb{Z}_n^*$,

tal que esté compuesto solo de los elementos de $\mathbb{Z}_n$ que tienen un inverso multiplicativo. Por ejemplo, para el caso dado en las tablas de arriba:

- $\mathbb{Z}_9 = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
- $\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$

Se pueden definir además dos conjuntos nuevos: $\mathbb{Z}_p$ y $\mathbb{Z}_p^*$. En estos conjuntos el módulo es un número primo. En el conjunto $\mathbb{Z}_p$ todos los números tienen su inverso aditivo y también el inverso multiplicativo excepto el 0, al que le falta este último. Para ello se define $\mathbb{Z}_p^*$ que contiene todos los enteros desde 1 hasta p-1, de esta manera todos los elementos tienen tanto inverso aditivo como multiplicativo.

1.3 Números Primos

- Un número entero n > 1 se dice que es *primo* si sus únicos divisores son 1 y n (dos divisores).
- Un entero n > 1 se dice que es *compuesto* si admite más de dos divisores.
- De acuerdo a estas definiciones el número 1 no es primo (un divisor). El número 2 es el primo más chico y todos los demás números primos son impares.
- Los números primos son infinitos. Ver demostración en [GAVALA].

Dos números, *a* y *b*, se dicen que son primos relativos o coprimos si el mcd (a,b) =1. Es útil notar que todos los elementos de los conjuntos $\mathbb{Z}_n^*$ y $\mathbb{Z}_p^*$ son primos relativos con respecto al módulo del conjunto.

Función de Euler

La función de Euler es de vital importancia para el cálculo de potencias grandes y juega un papel preponderante en la criptografía. Básicamente esta función devuelve la cantidad de números enteros que son menores y coprimos a un número n. Podemos decir que calcula la cantidad de elementos que componen al conjunto $\mathbb{Z}_n^*$.
Algunas propiedades de la función de Euler:

1. $\Phi(1) = 0$.
2. $\Phi(p) = p-1$, con p primo.
3. $\Phi(m \cdot n) = \Phi(m) \cdot \Phi(n)$ si m y n son coprimos.
4. $\Phi(p^e) = (p^e - p^{e-1})$ si p es primo

Con las propiedades anteriores podemos hallar el valor de $\Phi(n)$. Considerando a:

$$\mathbf{n = p_1^{e1} \times p_2^{e2} \times \ldots p_i^{ei}}$$

entonces podemos encontrar:

$$\mathbf{\Phi(n) = (p_1^{e1} - p_1^{e1-1}) \times (p_2^{e2} - p_2^{e2-1}) \times \ldots\ldots \times (p_i^{ei} - p_1^{ei-1})}$$

La complejidad de hallar el valor de $\Phi(n)$ para un *n* grande es directamente dependiente de la complejidad de factorización de n.

n	1	2	3	4	5	6	7	8	9	10	11	12
Φ(n)	1	1	2	2	4	2	6	4	6	4	10	4

Teorema de Euler y pequeño Teorema de Fermat
Estos dos teoremas son también de fundamental importancia en la criptografía.
Según el teorema de Euler si mcd (a,n) = 1, entonces:

$$\mathbf{a^{\phi(n)} \equiv 1 \bmod n}$$

Otra versión del teorema de Euler (usada en el criptosistema RSA) establece que si n = p . q, a < n y k es un entero, entonces:

$$\mathbf{a^{k \cdot \phi(n)+1} \equiv a \bmod n}$$

Un caso particular del teorema de Euler sucede si consideramos que *n* es un número primo. El pequeño teorema de Fermat establece que si *p* es un número primo y *a* un entero tal que mcd (p,a) = 1, entonces:

$$\mathbf{a^{p-1} \equiv 1 \bmod p}$$

Otra versión establece que si *p* es un número primo y *a* es un entero, entonces:

$$\mathbf{a^{p} \equiv a \bmod p}$$

1.4 Verificación de Primalidad

Dado que los métodos para encontrar un número primo (números de Fermat, números de Mersenne) no han dado resultado, la manera de generar números primos de gran tamaño es elegir un número al azar y verificar posteriormente si es primo. Esta no es una tarea sencilla y para ello existen dos tipos de algoritmos: determinísticos y probabilísticos.

Los algoritmos probabilísticos no siempre aciertan al determinar la primalidad, pero son más eficientes computacionalmente. Los algoritmos determinísticos siempre dan el resultado correcto a expensas de un mayor procesamiento, a veces inaceptable.

Los primeros no garantizan la exactitud del resultado pero el error es tan pequeño que son aceptables. El orden de complejidad del algoritmo es polinomial si se tolera cierto margen de error, ver anexo L de [FOROUZAN 2007]. El algoritmo devuelve ya sea un número primo o un número compuesto siguiendo las siguientes reglas:

- Si el entero bajo prueba es primo, el algoritmo acierta.
- Si el entero es compuesto, el algoritmo devuelve un compuesto con probabilidad $1 - \varepsilon$, y un primo con probabilidad ε.

Cambiando ciertos parámetros del algoritmo se pueden hacer M pasadas de manera de reducir el error a ε^M.

El test más utilizado para verificar la primalidad es el de Miller-Rabin que puede consultarse en [MENEZES 2001].
Una forma efectiva de elegir un número primo se muestra en la figura 2.

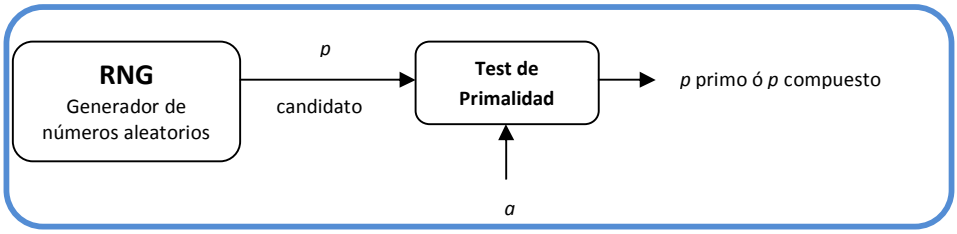


Figura 2– Procedimiento del test de primalidad

1.5 Factorización

La factorización es difícil. A pesar de centurias de esfuerzo por parte de matemáticos y expertos en computación, los métodos más rápidos para factorizar un número N necesitan de tiempo exponencial en el número de bits de N. Existen dos funciones populares de una sola vía que se usan en esquemas prácticos de clave pública. La primera es el problema de la factorización de enteros, en la cual se basa RSA. Dados dos primos grandes, es fácil computar el producto, sin embargo es difícil factorizar el producto resultante. La otra función de una sola vía que se utiliza es el problema del logaritmo discreto.
El Teorema Fundamental de la Aritmética establece que cualquier entero positivo mayor que uno se puede expresar como factor de número primos:

$$n = {p_1}^{e_1} x\ {p_2}^{e_2} x \dots \dots \dots x {p_k}^{e_k}$$

División por tentativa – Fuerza Bruta
El algoritmo de división por tentativas es el más simple para factorizar un entero. Supongamos que *a* y *b* son factores no triviales (ni ±1, ni ± N) de N tal que a . b = N y a ≤ b. Para llevar a cabo el algoritmo de división por tentativa, simplemente se verifica si a|N para a = {2…. √N}. Cuando se encuentra un divisor *a*, entonces b = N / a es también un factor, y de esta manera se ha hallado una factorización para N. Si es necesario se aplica el mismo algoritmo en forma recursiva a los factores resultantes.
Evidentemente esta no es la manera óptima de factorizar un número. Existen otros métodos como el *"Fermat"*, *"Pollard p-1"* y el *"Pollard rho"* que se pueden consultar en la bibliografía listada al final de este trabajo.

1.6 Teorema Chino del Resto

En su forma básica, el Teorema Chino del resto determina un número *x* que si se lo divide por ciertos divisores, da como resultado ciertos restos. Por ejemplo, ¿cuál es el menor número que cuando se lo divide por 3 da como resto 2, cuando se lo divide por 5 da como resto 3, y cuando se lo divide por 7 da como resto 2? Una analogía muy común cuenta sobre una mujer que le dice a un policía que perdió su canasta con huevos, y que si de la canasta sacaba de a tres huevos por vez le quedaban 2, si sacaba de a cinco por

vez le quedaban 3 y si sacaba de a siete por vez le quedaban 2. Entonces la mujer le pregunta al policía: ¿cuál es la mínima cantidad de huevos que tenía en la canasta? La respuesta es 23. La situación se puede expresar como:

$$\mathbf{x \equiv a_1 \pmod{m_1}} \qquad \mathbf{x \equiv a_2 \pmod{m_2}} \quad \ldots\ldots \quad \mathbf{x \equiv a_3 \pmod{m_3}}$$

El teorema chino del resto establece que las ecuaciones de arriba tienen una única solución si los módulos son primos relativos.
Ejemplo:
$x \equiv 2 \pmod 3$
$x \equiv 3 \pmod 5$ $\longrightarrow$ $x = 23$
$x \equiv 2 \pmod 7$
Procedimiento:

1. Encontrar $M = m_1 \times m_2 \times \ldots \times m_k$
2. Encontrar $M_1 = M/m_1$. $M_2 = M/m_2$ …….$M_k = M/m_k$
3. Encontrar el inverso multiplicativo de $M_1, M_2, \ldots. M_k$ usando los correspondientes módulos ($m_1, m_2, \ldots.. m_k$). Los llamamos: $M_1^{-1}, M_2^{-1}, \ldots\ldots.., M_k^{-1}$
4. La solución a las ecuaciones simultáneas es:

$$x = (a_1 \, x \, M_1 \, x \, M^{-1}{}_1 + a_2 \, x \, M_2 \, x \, M^{-1}{}_2 + \cdots .. a_k \, x \, M_k \, x \, M^{-1}{}_k) \, mod \, M$$

$M = 105$
$M_1 = 35, M_2 = 21, M_3 = 15$
$M_1^{-1} = 2 \quad M_2^{-1} = 1 \quad M_3^{-1} = 1$
$x = (2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1) = 23 \bmod 105$

1.7 Exponenciación y Logaritmo

La exponenciación y el logaritmo son la inversa el uno del otro:

$$\mathbf{y = a^x \quad \leftrightarrow \quad x = \log_a y}$$

En criptografía es común usar la operación modular exponenciación, para lo cual calculamos:

$$\mathbf{y = a^x \bmod n}$$

Una característica del algoritmo RSA es que usa la exponenciación con grandes exponentes tanto para cifrar como para descifrar. Como estos cálculos son bastante intensivos computacionalmente se usa un algoritmo más eficiente que la simple multiplicación, llamado de exponenciación rápida.
Para la exponenciación rápida se usa el método de elevar al cuadrado y multiplicar. La idea es expresar el exponente como un número binario de n bits (x_0 a x_{n-1}). Por ejemplo, $x = 19 = (10011)_2$. Para lo cual podemos escribir x como:

$$\mathbf{x = x_{n-1} \, x \, 2^{k-1} + x_n \, x \, 2^{k-2} + \ldots\ldots\ldots + x_2 \, x \, 2^2 + x_1 \, x \, 2^1 + x_0 \, x \, 2^0}$$

Y gráficamente podemos representar $y = a^x$

$$y = a^{\boxed{x = x_{n-1}\, x\, 2^{k-1} + x_n\, x\, 2^{k-2} + \ldots\ldots\ldots\ldots + x_2\, x\, 2^2 + x_1\, x\, 2^1 + x_0\, x\, 2^0}}$$

$$y = \boxed{a^{2^{n-1}} \text{ ó } 1} \times \boxed{a^{2^{n-2}} \text{ ó } 1} \text{ x.....x } \boxed{a^2 \text{ ó } 1} \times \boxed{a \text{ ó } 1}$$

$y = a^{19} = a^{10011} = a^{16} \,.\, 1 \,.\, 1 \,.\, a^2 \,.\, a$

Se aprecia que *y* es el producto de *n* términos. Cada término es 1 (si el bit correspondiente es 0) o a^{2^i} (si el bit correspondiente es 1). Es decir el término a^{2^i} se incluye en la multiplicación si el bit es 1 y no se incluye si el bit es 0.
El método usa *n* iteraciones. En cada iteración verifica el valor del bit correspondiente. Si el valor es 1, multiplica la base actual con el valor anterior. Luego eleva al cuadrado la base para la próxima iteración. Para más detalles se puede consultar [STALLINGS 2011].

Logaritmo

Si se usa exponenciación ($a^x \equiv y$) para cifrar o descifrar, el adversario puede usar el logaritmo para atacar. Verificaremos si la exponenciación es una función de una sola vía.
Lo primero que se puede hacer es resolver $x = \log_a y \pmod n$, implementando un algoritmo (de búsqueda exhaustiva) que calcule continuamente $y = a^x \bmod n$ hasta que halla el valor de *y* dado. Esto definitivamente es muy ineficiente y la complejidad del algoritmo es de orden exponencial $O(2^n)$.

Logaritmo Discreto

Otra forma es usar la idea del logaritmo discreto que implica ciertos conceptos de grupos multiplicativos. Se recomienda ver la bibliografía al final del trabajo para una introducción a este tema.

Grupo multiplicativo finito

Se puede definir un grupo multiplicativo finito: $G = \langle \mathbb{Z}_n^*, x \rangle$ en el cual la operación es la multiplicación. Este grupo contiene los enteros desde 1 a n-1 que son primos relativos con n. El elemento identidad es e = 1.

Orden del grupo

El orden de un grupo finito |G| es el número de elementos en el grupo G. En G=<$\mathbb{Z}_n$*,x> el orden del grupo es Φ (n).

Orden de un elemento

El orden de un elemento ord(a) es el entero más chico i tal que $a^i \equiv e \pmod n$. El elemento identidad "e" es 1 en este caso.

Ejemplo: Consideremos G = <$\mathbb{Z}_8$*, x>, por lo que Φ (8) = 4, y los elementos: {1,3,5,7}. Los órdenes de estos son:

- $1^1 \equiv 1 \bmod (8)$ → ord(1) = 1
- $3^1 \equiv 3 \bmod (8)$; $3^2 \equiv 1 \bmod (8)$ → ord(3) = 2
- $5^1 \bmod (8)$; $5^2 \equiv 1 \bmod (8)$ → ord(5) = 2
- $7^1 \bmod (8)$; $7^2 \equiv 1 \bmod (8)$ → ord(7) = 2

	i=1	i=2	i=3	i=4	i=5	i=6	i=7
a=1	x:1	x:1	x:1	x:1	x:1	x:1	x:1
a=3	x:3	x:1	x:3	x:1	x:3	x:1	x:3
a=5	x:5	x:1	x:5	x:1	x:5	x:1	x:5
a=7	x:7	x:1	x:7	x:1	x:7	x:1	x:7

$a^i \equiv x \pmod 8$ para el grupo G = <$\mathbb{Z}_8$*, x>

Raíces primitivas

En el grupo G = <$\mathbb{Z}_n$*, x> cuando el orden de un elemento es igual a Φ (n), ese elemento es llamado una raíz primitiva del grupo. La tabla siguiente muestra el resultado de $a^i \equiv x \pmod 7$ para el grupo G = <$\mathbb{Z}_7$*, x>, en este grupo Φ (7) = 6.

		i=1	i=2	i=3	i=4	I=5	i=6
	a=1	x:1	x:1	x:1	x:1	x:1	x:1
	a=2	x:2	x:4	x:1	x:2	x:4	x:1
Raíz primitiva →	a=3	x:3	x:2	x:6	x:4	x:5	x:1
	a=4	x:4	x:2	x:1	x:4	x:2	x:1
Raíz primitiva →	a=5	x:5	x:4	x:6	x:2	x:3	x:1
	a=6	x:6	x:1	x:6	x:1	x:6	x:1

Grupo cíclico

Si un grupo G = <$\mathbb{Z}_n$*, x> tiene raíces primitivas, es cíclico. Cada raíz primitiva es un "generador" y puede ser usada para crear todo el conjunto. Es decir si *g* es una raíz primitiva del grupo, se puede generar $\mathbb{Z}_n$* como:

$$\mathbb{Z}_n^* = \{g^1, g^2, g^3, \ldots\ldots, g^{\phi(n)}\}$$

La idea del logaritmo discreto

Algunas propiedades del grupo G = <$\mathbb{Z}_p$*, x>:

1. Incluye todos los elementos desde 1 hasta p-1
2. Siempre tiene raíces primitivas
3. Es cíclico. Los elementos se pueden crear usando g^x donde x es un entero desde 1 hasta Φ (n) = p-1.
4. Las raíces primitivas se pueden pensar como la base del logaritmo. Si el grupo tiene k raíces primitivas, los cálculos se pueden hacer en k bases diferentes. Dado $x = \log_g y$ para cualquier elemento *y* en el conjunto, hay otro elemento *x* que es el logaritmo de y en base g. Este tipo de logaritmo se llama logaritmo discreto. Se usará la notación L_g para mostrar que la base es g (el módulo se sobreentiende).

Solución a los logaritmos modulares usando logaritmos discretos

Se quiere resolver el problema del tipo $y = a^x \pmod n$ donde *y* es dato y se desea calcular *x*.

Una manera de hacerlo es usar una tabla para cada $\mathbb{Z}_p$* y diferentes bases. Estas tablas se pueden precalcular y salvar. La tabla siguiente muestra los logaritmos discretos para $\mathbb{Z}_7$*. Hay dos raíces primitivas en ese conjunto, por lo tanto dos bases distintas para el logaritmo (3 y 5).

y	1	2	3	4	5	6
$x = L_3\, y$	6	2	1	4	5	3
$x = L_5\, y$	6	4	5	2	1	3

Las tablas y algunas propiedades de los algoritmos discretos (no enunciadas aquí) no pueden usarse para resolver $y \equiv a^x \pmod n$ cuando n es muy grande. Se han propuesto muchos algoritmos para resolver este problema (más eficientes que el de búsqueda exhaustiva) pero ninguno tiene complejidad de orden polinomial, $O(n^c)$ y tienen generalmente el mismo nivel de complejidad que el problema de factorización.

Capitulo 2

Criptografía

- *Por mi naturaleza estoy destinado a envejecer. No hay forma de huir el envejecimiento.*
- *Por mi naturaleza estoy destinado a enfermarme. No hay forma de huir de la enfermedad.*
- *Por mi naturaleza estoy destinado a morir. No hay forma de huir de la muerte.*
- *Todo lo que quiero y las personas que amo tienen la naturaleza del cambio. No hay forma de evitar tener que separarme de ello.*
- *Mis acciones son lo único que realmente me pertenece. No puedo huir de las consecuencias de mis acciones. Ellas son la base que me soporta*

Siddhartha Gautamá

2 Criptografía

En este capítulo se hace una exposición de los conceptos básicos de la criptografía como una rama de la criptología. Se analizan los dos esquemas fundamentales de cifrado: de clave pública y de clave privada. Además se expone la utilidad del criptoanálisis como una herramienta de verificación de validez de los algoritmos de cifrado.

2.1 Criptografía como una rama de la criptología

El término criptografía proviene del griego kryptós, "escondido", y graphos, "escritura"; y es el arte de ocultar los mensajes compuestos de signos convencionales, de manera que solo develen su significado a la luz de una clave secreta. Es la ciencia y el arte de reescribir el texto escrito, de manera que sea indescifrable, para quien no posee la clave.

La criptografía parece estar estrechamente relacionada a las comunicaciones electrónicas modernas. Sin embargo, la criptografía es una cuestión bastante antigua, como lo demuestran los jeroglíficos secretos usados en el año 2000 AC en el antiguo Egipto. Desde aquella época la criptografía ha estado presente de alguna forma en casi todas las culturas de lengua escrita.

En la figura 1 se observan los campos en los que se divide la disciplina científica de la criptología, donde podemos observar que la criptografía es una rama de esta.

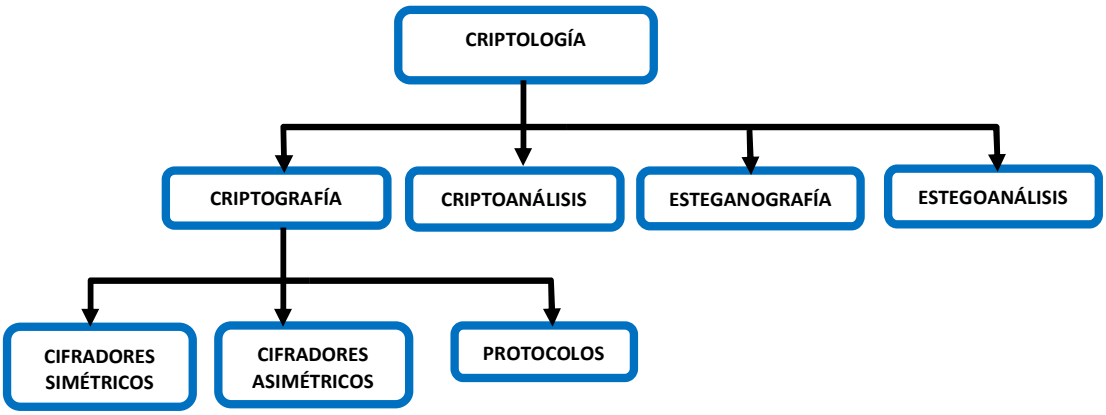


Figura 1 – Áreas de la Criptología

Criptografía: Se ocupa del estudio de los algoritmos, protocolos y sistemas que se utilizan para proteger la información y dotar de seguridad a las comunicaciones y a las entidades que se comunican.

Criptoanálisis: Se ocupa de intentar capturar el significado de los mensajes construidos mediante la criptografía sin tener autorización para ello. Podríamos decir que el criptoanálisis tiene un objetivo opuesto al de la criptografía. Su propósito es buscar el

punto débil de las técnicas criptográficas para explotarlo y así reducir o eliminar la seguridad que teóricamente aportaba esa técnica criptográfica. A cualquier intento de criptoanálisis se lo llama ataque. Un ataque tiene éxito, y se dice que el sistema se ha quebrado, cuando el atacante consigue romper la seguridad que la técnica criptográfica aportaba al sistema.

Criptosistemas

Un criptosistema, o sistema criptográfico, se puede definir como los fundamentos y procedimientos de operación que participan en el cifrado y descifrado de un mensaje.
Todo sistema criptográfico es una tupla: {P, C, K, E y D} con las siguientes propiedades:

- P es el conjunto de todos los mensajes a transmitir. Es el espacio de textos planos.
- C es el conjunto de todos los mensajes cifrados. Es el espacio de textos cifrados.
- K es el conjunto de claves a utilizar. Es el espacio de claves.
- E es la familia de todas las funciones de cifrado. $E = \{E_k : k \in K\}$ y $E_k : P \rightarrow C$.
- D es la familia de todas las funciones de descifrado. $D = \{D_k : k \in K\}$ y $D_k : C \rightarrow P$.
- Para cada $e \in K$, hay un $d \in K$ tal que $D_d(E_e(p)) = p$ para todo $p \in P$. (e puede ser igual a d).

Todo criptosistema debe cumplir, al menos, tres requisitos básicos:

1. Todas las transformaciones de cifrado y descifrado, E_k y D_k, han de ser fácilmente calculables.
2. Los algoritmos de las transformaciones E_k y D_k tienen que ser fácilmente implementables.
3. La seguridad del sistema debe depender solo del secreto de las claves k y no de los algoritmos de las transformaciones E y D (Principio de Kerchoff).

Además, un buen criptosistema tiene que tener las siguientes cualidades: seguridad, autenticidad y no repudio:

- Seguridad: es la incapacidad para un criptoanalista de determinar el texto original, a partir del texto cifrado que haya podido interceptar.
- Autenticidad e integridad: es la incapacidad para un criptoanalista de improvisar, sustituir o modificar un texto cifrado C por un texto C', sin que el receptor lo detecte.
- No repudio: es la propiedad de que el emisor, después de haber enviado un mensaje, no pueda afirmar que este no es suyo. Esto se realiza por otros medios, como la firma digital, que se adaptan al criptosistema utilizado.

Algoritmos Simétricos (o de clave privada)

Es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar los mensajes. Las dos partes que se comunican tienen un método compartido de cifrado/descifrado y han de ponerse de acuerdo de antemano sobre la clave a utilizar. Una vez que ambas partes tienen acceso a esta clave, el remitente la usa para cifrar un mensaje, luego se lo envía al destinatario, y éste lo descifra con la misma clave. Toda la criptografía desde tiempos antiguos y hasta 1976 se basó exclusivamente en métodos simétricos. Estos aun están en vigencia, especialmente para cifrado de datos y verificación de integridad de mensajes.

Algoritmos Asimétricos (o de clave pública)

La criptografía asimétrica, es por definición, aquella que utiliza dos claves diferentes para cada usuario, una para cifrar que se la llama clave pública y otra para descifrar que es la clave privada. El nacimiento de la criptografía asimétrica se dio al estar buscando un modo más práctico de intercambiar las claves simétricas. En 1976, Diffie y Hellman propusieron una manera para hacer esto, sin embargo la criptografía asimétrica no tomó forma hasta que el popular método RSA de Rivest, Shamir y Adleman (publicado en 1978) se hiciera conocido. Su funcionamiento está basado en la imposibilidad computacional de factorizar números enteros grandes. Los algoritmos asimétricos se pueden emplear para aplicaciones como firma digital y establecimiento de clave, y también para el cifrado clásico.

Esquemas híbridos

En criptografía, los criptosistemas de clave pública son convenientes en cuanto a que no requieren que el emisor y el receptor compartan un secreto común a fin de comunicarse de forma segura (entre otras propiedades útiles). Sin embargo, a menudo dependen de complicados cálculos matemáticos y por ello son, en general, mucho más ineficientes que los criptosistemas simétricos. En muchas aplicaciones que usan criptografía de clave pública, el alto costo de la encriptación de mensajes largos puede ser prohibitivo. Un criptosistema híbrido es aquel que combina la conveniencia de un sistema de cifrado de clave pública con la eficiencia de un sistema de cifrado de clave simétrica.

Un sistema criptográfico híbrido se construye combinando los dos sistemas criptográficos:

- Un *sistema de encapsulación de clave*, que es un sistema de cifrado de clave pública.
- Un *sistema de encapsulación de datos*, que es un sistema de cifrado de clave simétrica.

Cabe observar que para mensajes muy largos la mayor parte del trabajo de cifrado/descifrado de datos se realiza por parte del sistema de clave simétrica (generalmente más eficiente en términos computacionales), mientras que el sistema de clave pública (generalmente más ineficiente en términos computacionales), se utiliza solo para cifrar/descifrar un valor de clave corto. La razón para usar ambas familias de algoritmos es que cada uno tiene sus fortalezas y debilidades específicas.

Protocolos Criptográficos

A grandes rasgos, los protocolos criptográficos tienen que ver con la implementación de algoritmos criptográficos. Los algoritmos simétricos y asimétricos pueden verse como los bloques constructivos con los cuales se implementan ciertas funcionalidades, como por ejemplo la comunicación segura a través de Internet. Son ejemplos de protocolos criptográficos el esquema TLS (Transport Layer Security), que se utiliza en todos los navegadores web y él cual aplica al algoritmo RSA en su implementación; por otro lado el protocolo SSH (Secure Shell) se utiliza para realizar sesiones de consola seguras e implementa en su estructura al algoritmo de Diffie-Hellman.

2.2 Criptografía simétrica

Se pude hacer una introducción a la criptografía simétrica con un ejemplo sencillo. Hay dos usuarios, Alice y Bob, que se quieren comunicar sobre un canal no seguro como puede ser Internet, una WLAN, etc. (figura 2). El problema es cuando entra en juego un tercer actor, Eve, que tiene acceso al canal, por ejemplo, comprometiendo un router o escuchando las señales de radio de una comunicación inalámbrica. Este último tipo de escucha no autorizada se llama "eavesdropping". Existen numerosas situaciones en las que Alice y Bob preferirían comunicarse sin que nadie escuche sus conversaciones.

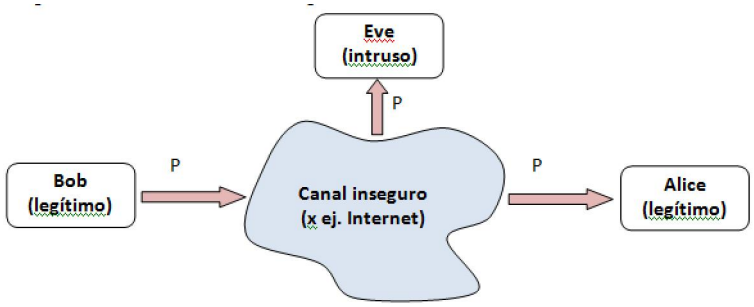


Figura 2 – Comunicación con canal inseguro

En el esquema de la figura 3, Alice puede enviar un mensaje a Bob, sobre el canal inseguro con la certeza de que el adversario, Eve, no podrá comprender el contenido del mensaje enviado.

Texto plano (Plaintext): Mensaje **P** enviado por Alice a Bob

Texto Cifrado (Ciphertext): Mensaje **C** enviado por el canal inseguro.

El cifrado simétrico utiliza una sola clave tanto para el cifrado como para el descifrado. A su vez los algoritmos de cifrado y descifrado son la inversa el uno del otro. Si P es el texto plano, C es el texto cifrado, y *k* es la clave, entonces el algoritmo de cifrado $E_k(P)$ crea el texto cifrado del texto plano y el algoritmo de descifrado $D_k(C)$ crea el texto plano del texto cifrado.

Cabe mencionar que Alice y Bob necesitarán otro canal, que sea seguro, para intercambiar las claves secretas; como por ejemplo el intercambio de una clave secreta temporal usando un cifrador de clave pública.

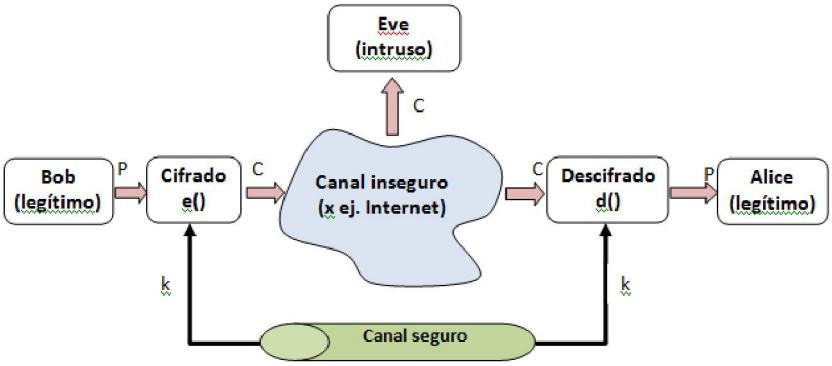


Figura 3 – Sistema de cifrado simétrico

Longitud de la clave

La discusión de la longitud de la clave para los algoritmos simétricos tiene sentido si solo hablamos de ataques de fuerza bruta. Si existe un ataque de tipo analítico que funciona bien, no nos será de mucha ayuda un espacio de claves grande. Además están también las posibilidades de ataques de ingeniería social y de implementación, donde un clave larga tampoco brinda ayuda extra. Por otro lado, el largo de clave para algoritmos simétricos y asimétricos es dramáticamente diferente. Por ejemplo, una clave simétrica de 80 bits proporciona la misma seguridad que una clave de 1024 bits en RSA.

Un espacio de claves grande es una condición necesaria, pero no suficiente, para un cifrador simétrico seguro. Este debe ser además resistente ante ataques de tipo analítico.

Largo de clave	Estimación de seguridad
56-64 bits	Corto plazo: horas o días.
112-128 bits	Largo plazo: décadas en ausencia de computadoras cuánticas.
256 bits	Largo plazo: décadas aun con computadoras cuánticas.[1]

2.3 Criptografía asimétrica

Los algoritmos de clave pública son muy diferentes a los algoritmos simétricos como AES o DES. La mayoría de los algoritmos de clave pública están basados en funciones teóricas. Esto es diferente a los cifradores simétricos, donde el objetivo no es tener una descripción matemática compacta entre la entrada y la salida. Aunque dentro de los cifradores simétricos se usan estructuras matemáticas para implementar bloques pequeños (por ejemplo la S-Box en AES), esto no significa que el cifrador en sí mismo constituya una descripción matemática compacta.

Mientras que la criptografía de clave simétrica se basa en sustituciones y permutaciones de símbolos, la criptografía de clave pública se basa en aplicar funciones matemáticas a números. En la criptografía de clave simétrica, el texto plano y el texto cifrado se toman como una combinación de símbolos. El cifrado y descifrado permutan o sustituyen un símbolo por otro.

[1] Las computadoras cuánticas tienen el potencial de resolver ciertos tipos de problemas mucho más rápido que los ordenadores clásicos. Con ellas, se gana en velocidad y eficacia.

En la criptografía de clave asimétrica, el texto plano y el texto cifrado son números; el cifrado y el descifrado son funciones matemáticas que se aplican a números para crear otros números. Veamos algunas cuestiones asociadas con los algoritmos simétricos.

Problema de distribución de clave: La clave entre Alice y Bob se debe establecer utilizando un canal seguro. El canal de comunicación del mensaje, *per se*, no es seguro; por lo que no es factible el envío de la clave directamente a través del mismo (lo que hubiera sido la manera más conveniente de hacerlo).

Número de claves: Una vez resuelto el problema de la distribución de las claves, tenemos el problema de la gran cantidad de claves. Cada par de usuarios necesita un par de claves separadas. En una red de *n* usuarios tenemos:

$$\frac{n \times (n-1)}{2}$$

pares de claves, y cada usuario tiene que almacenar n-1 claves de forma segura. Para una red corporativa de 1000 personas, se necesitan medio millón de pares de claves que deben ser generadas y transportadas sobre canales seguros.

No es posible el no repudio: Alice y Bob poseen las mismas facultades debido a que tienen la misma clave. Por lo tanto la criptografía simétrica no nos defiende de que Alice o Bob se hagan trampa entre sí. Un ejemplo típico es el caso donde Alice envía una orden de compra a Bob por una cantidad *x* de cañas de pescar, luego Alice se arrepiente y puede tranquilamente decir que ella no fue quien emitió esa orden de compra, ya que Bob también pudo haberla generado, dado que él conoce la clave. De esta manera no hay prueba legal con la que se pueda dirimir el conflicto. Por lo tanto, precisamos un mecanismo para identificar por separado a cada uno de los actores comerciales de este caso. La prevención de esta situación se denomina *no repudio* y se puede conseguir a través de la implementación de la firma digital usando algoritmos de clave asimétrica.
Con el fin de superar estos inconvenientes, Diffie, Hellman y Merkle tuvieron una propuesta revolucionaria basada en la siguiente idea: no es necesario que la clave que tiene la persona que cifra el mensaje sea secreta. La parte crucial es que solo el receptor puede descifrar el mensaje con una clave secreta. Para implementar este sistema, una de las partes, Bob, publica una clave pública de cifrado que es conocida por todo el mundo. A su vez retiene una clave secreta que se complementa con la que hizo pública, y la utilizará para el descifrado. Por lo tanto, la clave *k* de Bob consiste de dos partes, una pública, k_{pub}, y una privada, k_{pr}. En la figura 4 se muestra un protocolo básico de cifrado de clave pública.

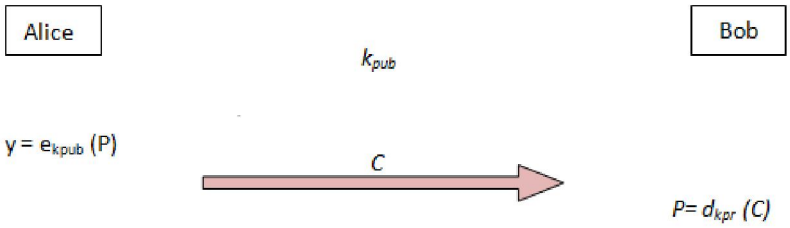


Figura 4 – Esquema comunicación con algoritmo asimétrico

En la figura 5 se muestra como usar un algoritmo asimétrico para transportar una clave simétrica de AES. Lo que se hace es cifrar la clave simétrica de AES con algún algoritmo de clave pública. Una vez que Bob descifró la clave simétrica, ambas partes pueden usarla para cifrar y descifrar aplicando cualquier algoritmo simétrico (en este caso AES). La principal ventaja con el esquema de la figura 4 es que los datos se cifran con un algoritmo de cifrado simétrico, lo cual es mucho más rápido.

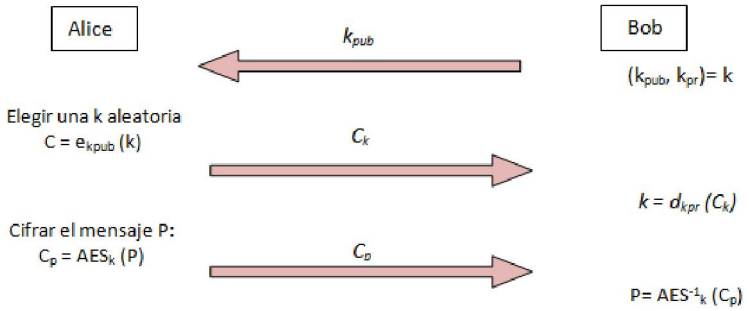


Figura 5– Algoritmo asimétrico para intercambio de claves

Los algoritmos de cifrado asimétricos se construyen sobre un principio común, las funciones de una sola vía (OWF = One Way Function)

Función de una sola vía: Una función f () es una función de una sola vía si:

1. y = f (x) es computacionalmente sencilla, y
2. x = f^{-1}(y) es computacionalmente inviable

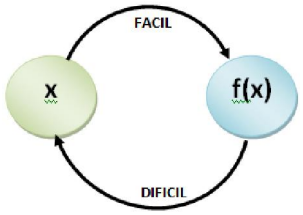


Figura 6 – Funciones de una sola vía

Una función trampa de una sola vía (TOWF = Trapdoor One Way Function), es aquella que es fácil de calcular en una dirección pero inviable de calcular en la otra dirección, a no ser que se conozca cierta información extra. Con la información adicional la inversa puede calcularse en tiempo polinomial.

Resumiendo:

1. $y = f_k(x)$ es computacionalmente sencilla si se conoce *k* y *x*.
2. $x = f_k^{-1}(y)$ es computacionalmente sencilla si se conoce *k* e *y*.
3. $x = f_k^{-1}(y)$ es inviable si se conoce *y*, pero no se conoce *k*.

Una función es fácil de calcular si puede ser evaluada en tiempo polinomial, es decir, el tiempo de corrida es una expresión polinomial; para mayor detalle ver anexo L de [FOROUZAN 2007]. Para que estas funciones sean útiles en los esquemas prácticos de cifrados, el cálculo de $y = f(x)$ debe ser suficientemente rápido como para que no lleve a una aplicación a tiempos de ejecución inaceptablemente lentos. El cálculo inverso $x = f^{-1}(y)$, debe ser computacionalmente tan intenso que no es posible evaluarlo en un periodo de tiempo razonable, digamos 10.000 años, aun cuando se usen los mejores algoritmos conocidos.

Mecanismos de Seguridad

Los principales servicios de seguridad que la criptografía de clave pública puede brindar son:

Establecimiento de claves. Hay protocolos para establecer claves secretas sobre un canal inseguro. Por ejemplo, el protocolo "Diffie–Hellman key Exchange" (DHKE) o protocolos de transporte de clave basados en RSA.

No repudio. Con los algoritmos de firma digital se puede proveer no repudio e integridad de mensaje (RSA, DSA)

Identificación. Se pueden identificar usuarios usando protocolos de desafío y respuesta o con firma digital.

Cifrado. Se pueden cifrar mensajes con algoritmos como RSA o Elgamal

Por un lado los algoritmos de clave simétrica pueden manejar el cifrado/descifrado pero se les complica con el manejo de la administración de las claves. Por otro lado parecería que los algoritmos de clave pública pueden proveer todas las funcionalidades necesarias de seguridad. El problema es que el cifrado/descifrado es computacionalmente demasiado intenso (100 a 1000 veces más lentos que los cifradores simétricos) por lo cual no se los utiliza para esta función. Asimismo los algoritmos de clave privada no son buenos para el manejo de clave y para brindar no repudio. ¿Solución?, usar lo mejor de cada mundo, la mayoría de los protocolos reales son protocolos híbridos que incorporan tanto algoritmos simétricos como asimétricos. Un ejemplo de esto es el protocolo SSL/TLS que se usa para conexiones Web seguras.

Una cuestión que queda pendiente con los algoritmos asimétricos es la autenticidad de las claves públicas que son distribuidas libremente. ¿Cómo sabemos que realmente pertenecen a la persona que dice ser y no estamos siendo engañados por un tercero? Básicamente el problema se resuelve con certificados, que asocian una clave pública a una identidad y con una Autoridad de Certificación (CA) en la que uno confía, y que es la emisora de dichos certificados.

Algoritmos de clave pública más utilizados

Hay tres grandes familias de algoritmos de clave pública que son de relevancia práctica. Se pueden clasificar en base a su problema computacional subyacente.

Esquemas de Factorización de Enteros. Se basan en la dificultad de factorizar números enteros de gran tamaño. El ejemplo clásico es RSA.

Esquemas de Logaritmo Discreto. Se basan en el problema del logaritmo discreto en campos finitos. Ejemplos: Intercambio de claves de Diffie-Hellman (DHKE), cifrado Elgamal y el Algoritmo de Firma Digital (DSA = Digital Signature Algorithm).

Esquemas de Curva Elíptica (EC - Elliptic Curve). Es una generalización del algoritmo de logaritmos discretos. Ejemplos: Intercambio de claves con implementación de Diffie–Hellman en curvas elípticas (ECDH) e implementación de DSA en curvas elípticas (ECDSA).

Largo de las claves y nivel de seguridad

Algoritmo	Criptosistema	Nivel de Seguridad (bits)			
		80	128	192	256
Factorización de enteros	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Logaritmo discreto	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Curvas Elípticas	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Clave simétrica	AES, 3DES	80 bit	128 bit	192 bit	256 bit

Los algoritmos de clave pública requieren operandos y claves de gran tamaño. Cuanto más grande son éstos más seguros se vuelven los algoritmos. Un parámetro de comparación entre los distintos algoritmos es el *nivel de seguridad*. Se dice que un algoritmo tiene un nivel de seguridad de n bits si el ataque requiere 2^n pasos. Para los algoritmos de clave simétrica esta relación es natural, dado que con n bits tenemos un espacio de claves de 2^n. Con los algoritmo de clave pública la relación no es directa y generalmente se precisan muchos más bits para lograr el mismo nivel de seguridad que en un algoritmo de clave privada. En la tabla se muestran la cantidad de bits necesarios para cada algoritmo a fin de alcanzar el nivel de seguridad deseado.

Los ataques de fuerza bruta se contrarrestan, como siempre, eligiendo el nivel de seguridad adecuado (cantidad de bits de la clave).

Otra forma de ataque es averiguar alguna manera de calcular la clave privada de la clave pública. Todos los algoritmos son susceptibles a este tipo de ataque. En la historia del criptoanálisis se ha visto numerosas veces que lo que parecía irresoluble desde una perspectiva, se transformaba en un cuestión sencilla cambiando la forma de abordar el problema.

Capitulo 3

Algoritmos Asimétricos

- *Cuida tus pensamientos, porque se transformarán en actos;*
- *Cuida tus actos, porque se transformarán en hábitos;*
- *Cuida tus hábitos, porque determinarán tu carácter;*
- *Cuida tu carácter, porque determinará tu destino;*
- *¡Y tu destino es tu vida!*

Mahatma Gandhi

3 Algoritmos asimétricos

En este capítulo se analizan los fundamentos de los algoritmos de clave asimétrica más populares y de aplicación práctica más difundida. Comenzamos con el reconocido algoritmo RSA, útil tanto para la firma digital como para el intercambio de claves, y base del protocolo SSL. Seguimos con el algoritmo de intercambio de claves de Diffie-Hellman utilizado, entre otros, en el protocolo SSH. Luego analizamos el algoritmo DSA como una aplicación práctica de ElGamal. Para finalizar veremos una introducción a la criptografía de curva elíptica.

3.1 RSA – Rivest, Shamir y Adleman

Se puede decir que RSA es el algoritmo de clave asimétrica que más se utiliza. Mayormente se lo emplea para:

- Cifrado de pequeñas cantidades de datos, por ejemplo claves.
- Firmas digitales.

Obviamente el algoritmo RSA no está pensado para reemplazar a los algoritmos simétricos dada su lentitud de cómputo y el aumento en el tamaño del mensaje cifrado. Su utilidad radica en lograr un intercambio de claves seguro para que luego algoritmos como AES puedan llevar a cabo el pesado trabajo de cifrado de manera más eficiente.
La función de una sola vía en la que se basa RSA es el problema de factorización de números enteros. Multiplicar dos números primos grandes es computacionalmente fácil, no así la operación inversa de factorización. Para comprender su funcionamiento es necesario tener en claro el teorema de Euler y la función phi de Euler dado que juegan un papel muy importante.

Procedimiento para cifrar y descifrar

El cifrado y descifrado se realiza en el anillo entero $\mathbb{Z}_n$. Para una explicación detallada puede consultarse [FOROUZAN 2007]. RSA cifra el texto plano P que es un elemento en $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. Por lo tanto el valor binario de P tiene que ser menor que *n*. Lo mismo sucede con el descifrado.

Cifrado: Dada la clave pública (n,e) = k_{pub} y el texto plano P, la función de cifrado es:

$$\mathbf{C = ek_{pub}(P) \equiv P^e \bmod n} \quad \text{donde P y C} \in \mathbb{Z}_n$$

Descifrado: Dada la clave privada d = k_{pr} y el texto cifrado C, la función de descifrado es:

$$\mathbf{P = dk_{pr}(C) \equiv C^d \bmod n} \quad \text{donde P y C} \in \mathbb{Z}_n$$

- *P*, C y *n* son números grandes ≥ 1024 bits.
- *e*: exponente de cifrado o exponente público.
- *d*: exponente de descifrado o exponente privado.

Si Bob le quiere enviar un mensaje cifrado a Alice, necesita la clave pública (n,e) de esta última y Alice, a su vez, descifra con su clave privada *d.*
Algunos requerimientos del criptosistema RSA:

- Dados *e* y *n* debe ser inviable obtener *d*.
- No se pueden cifrar más de *n* bits. En caso que sea necesario debe hacerse por bloques.
- Se necesita un método de exponenciación rápida de manera de poder obtener fácilmente tanto P^e mod n, como C^d mod n (algoritmo de elevar al cuadrado y multiplicar).
- Para un *n* dado debe haber muchos pares de claves públicas/privadas, sino sería viable un ataque de fuerza bruta.

Generación de claves RSA

1. Elegir dos primos grandes tal que p ≠ q.
2. n = p.q
3. Φ(n) = (p-1).(q-1)
4. Elegir el exponente público *e* tal que 1 < e < Φ(n) y mcd (e, Φ(n)) = 1.
5. Calcular el exponente privado d = e^{-1} mod Φ(n). (d es el inverso de e mod Φ(n))
6. Clave pública → (e, n)

- La condición que el mcd (e, Φ(n)) = 1 asegura que existe el inverso multiplicativo de *e* módulo *n* y por lo tanto hay una clave privada *d*.
- Los números primos del paso 1 se calculan de acuerdo a lo visto en el capítulo 2.
- El cálculo de *d* y *e* se puede hacer al mismo tiempo usando el algoritmo extendido de Euclides.
- El tamaño recomendado de cada primo *p* y *q* es de 512 bits (aprox. 152 dígitos decimales); por lo que el tamaño del módulo *n* es de 1024 bits (309 dígitos).
- Los valores *p*, *q* y Φ(n) podrían ser descartados luego de los cálculos.

Una demostración del algoritmo RSA se puede ver en [FOROUZAN 2007].

Ejemplo

Si p = 47 y q = 71, entonces se calcula:

$n = p.q = 47 \times 71 = 3337$ $\Phi(n) = (p-1).(q-1) = 46 \times 70 = 3220$

Se elige aleatoriamente el exponente público e = 79 de manera tal que el mcd (e, Φ(n)) = mcd (79, 3220) = 1, es decir *e* y Φ(n) son primos relativos. Usando el algoritmo extendido de Euclides (mcd (e, Φ(n)) = 1 = (e.d + Φ(n).s), se calcula el exponente privado *d* tal que:

$e.d \equiv 1 \pmod{\Phi(n)}$ $79.d \equiv 1 \pmod{3220}$

Aplicando el algoritmo extendido de Euclides se calcula *d*:

q	r_1	r_2	r	x_1	x_2	x	y_1	y_2	y
40	3220	79	60	1	0	1	0	1	-40
1	79	60	19	0	1	-1	1	-40	41
3	60	19	3	1	-1	4	-40	41	-163
6	19	3	1	-1	4	-25	41	-163	1019
3	3	1	0	4	-25	79	-163	1019	-3220
	1	0		**-25**	79		**1019**	-3220	

mcd (e, Φ(n)) = 1 = 79 × **1019** − 3220 × 25

(79) x (1019) ≡ 1 (mod 3220) **d = 1019 (clave privada)**

Para cifrar un mensaje P = 688 con e = 79, se calcula:

$C \equiv P^e \pmod{n} \equiv 688^{79} \pmod{3337} \equiv 1570 \pmod{3337}$

Para descifrar el mensaje, se hace el mismo proceso de exponenciación usando el exponente privado.

d = 1019 por lo cual:

$P \equiv C^d \pmod{n} \equiv 1570^{1019} \pmod{3337} \equiv 688$

Y por lo tanto se recupera el mensaje.

Utilizando el programa openssl [http://www.openssl.org/] se puede generar y verificar la composición de las claves pública y privada obtenidas de acuerdo a [PKCS#1] (Apéndice A – RSA Key Representation). También se usará este programa para cifrar un mensaje de texto corto tanto con AES como con RSA y notaremos la diferencia de tamaño del texto cifrado, a equivalentes niveles de seguridad.

1. Generación clave privada.

```
OpenSSL> genrsa -out privada.pem 1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.....++++++
e is 65537 (0x10001)
```

2. Generación clave pública.

OpenSSL> rsa -in privada.pem -out publica.pem -pubout

writing RSA key

3. Verificación estructura clave privada.

OpenSSL> rsa -text -in privada.pem

Private-Key: (1024 bit)

modulus:

00:99:e8:10:4c:2d:39:a2:84:3c:0d:9f:8f:92:dd:

…..

8d:2f:f1:a0:62:9b:b6:a0:69

publicExponent: 65537 (0x10001)

privateExponent:

00:90:77:e3:84:7e:17:51:d4:2a:c2:e6:fa:50:73:

….

86:f1:42:68:35:76:75:bc:39

prime1:

00:cb:26:33:04:59:bf:b8:c0:03:54:2c:c7:1a:f5:

.....:

78:ee:ed:1a:4b

prime2:

00:c1:f2:48:be:30:d4:f8:a9:25:6f:27:d9:4f:7a:

….

45:d4:85:ff:9b

exponent1: d mod (p − 1)

00:96:da:64:e4:04:29:64:6d:e1:8a:ee:3d:63:fd:

….

89:a6:12:97:85

exponent2: d mod (q − 1)

26:ec:9a:5e:d9:6c:14:cf:d5:bc:7e:51:e5:da:8b:

…..

b1:56:87:5b

coefficient: es el coeficiente CRT q^{-1} mod p.

00:c6:95:0d:34:c6:0f:f1:d6:6e:38:d1:5d:df:73:

…..

1d:67:d6:ec:5c

writing RSA key

-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQCZ6BBMLTmihDwNn4+S3UTQSFOij1OC3WlfqkBX1zTfNUu8rmyt
……
LJQvzI+Wp/1OtFKkxDkqMMRMohpyA5RMBkoBWDYdZ9bsXA==
-----END RSA PRIVATE KEY-----

3. Verificación de la estructura de la clave pública.

OpenSSL> rsa -text -pubin -in publica.pem

Public-Key: (1024 bit)

Modulus:

00:99:e8:10:4c:2d:39:a2:84:3c:0d:9f:8f:92:dd:

……

8d:2f:f1:a0:62:9b:b6:a0:69

Exponent: 65537 (0x10001)

writing RSA key

-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCZ6BBMLTmihDwNn4+S3UTQSFOi
….
QSSjMp7GBUJ4mdZeSILNCljETWKnJwzfpKH2NZwISZi/rqjwv4wHAbqxNzJMAauV
yU60uo0v8aBim7agaQIDAQAB
-----END PUBLIC KEY-----

Ahora cifraremos un archivo de texto de 3 bytes tanto con AES de 128 bits como con RSA de 3072 bits para ver la diferencia en tamaño de la salida encriptada. Los valores de clave se tomaron para respetar el mismo nivel de seguridad relativo entre protocolos.

1_ RSA: ***rsautl -in hola.txt -out rsahola.txt -inkey autofirmado.pem -certin –encrypt***
Tamaño de salida: 384 bytes, es decir 384 bytes x 8 bits = 3072 bits que es el módulo elegido.
2_ AES: ***aes-128-cbc -in hola.txt -out aeshola.txt -pass pass:190372***
aes-128-ecb -in hola.txt -out aeshola.txt -pass pass:190372
aes-256-cbc -in hola.txt -out aeshola.txt -pass pass:190372
aes-256-ecb -in hola.txt -out aeshola.txt -pass pass:190372
Para todos los casos el tamaño de salida es: 32 bytes, es decir 32 bytes x 8 bits = 256 bits.
3_ 3DES: ***des3 -in hola.txt -out 3des.txt -pass pass:190372***
Tamaño de salida: 24 bytes, es decir 24 bytes x 8 bits = 192 bits

Algoritmo de Cifrado	Tamaño archivo entrada	Tamaño archivo salida
aes-128-ecb; aes-128-cbc aes-256-ecb; aes-256-cbc	3 bytes / 24 bits	32 bytes / 256 bits
3des	3 bytes / 24 bits	24 bytes / 192 bits
rsa (3072bits)	3 bytes / 24 bits	384 bytes / 3072 bits

Cifrado rápido con exponentes públicos cortos
En este caso la clave pública *e* se elige para que sea de un valor pequeño. Hay tres valores de particular importancia: e = 3, e = 17 y e = $2^{16} + 1$.

Clave pública e	e en formato binario	Cantidad de sumas y multiplicaciones
3	11_2	3
17	10001_2	5
$2^{16} + 1$	$1\ 0000\ 0000\ 0000\ 0001_2$	17

Todos esos exponentes tienen un peso de Hamming bajo, es decir la cantidad de unos en su representación binaria. Esto es favorable dado que se reduce la cantidad de operaciones necesarias para la exponenciación. De todos modos, aun con estos valores de *e*, RSA continúa siendo seguro. La clave privada *d* mantiene su largo total igual a la longitud de *n* aunque *e* sea corto. Un análisis del problema de elegir una clave privada corta se pueda ver en [WIENER 1989].
Como consecuencia se obtiene el beneficio de poder cifrar mensajes y verificar las firmas digitales RSA de manera más rápida. De hecho RSA se transforma, de este modo, en el esquema de clave pública más rápido. Sin embargo la mayor complejidad de cálculo sigue quedando del lado donde está la clave privada *d*, de esta manera el

descifrado y la generación de firmas son más lentos. No obstante, hay que tener en cuenta que esta asimetría está del lado favorable ya que generalmente la firma digital se genera una vez pero se verifica muchas más veces.

Descifrado rápido aplicando el Teorema Chino de Resto

Dado que *p* y *q* son primos, cualquier mensaje P < n = p.q se representa en forma unívoca por la tupla $[P_p, P_q]$, donde $P_p = P \bmod p$ y $P_q = P \bmod q$. Por lo tanto es posible obtener P calculando P_p y P_q y combinándolos de acuerdo al Teorema Chino del Resto visto en el capítulo 2, en vez de usar la forma teórica $P = C^d \bmod n$.

$P_p = P \bmod p = (C^d \bmod n) \bmod p$
$= C^d \bmod p$ (dado que n = p.q)
$= C^{d \bmod (p-1)} \bmod p$ (*)
$= C^{dP} \bmod p$ con $d_P = d \bmod (p-1)$

(*) De acuerdo con [GROßSCHÄDL 2000] escribimos *d* como un múltiplo de Φ(p) más un resto, d = k Φ(p) + d mod Φ(p), donde k es un entero. Entonces:
$C^d = C^{k\Phi(p) + d \bmod \Phi(p)} = (C^{\Phi(p)})^k . C^{d \bmod \Phi(p)}$.
El Teorema de Euler establece: $C^{\Phi(p)} \equiv 1 \bmod p$. Por lo tanto:
$C^d \equiv 1^k . C^{d \bmod \Phi(p)} \equiv C^{d \bmod \Phi(p)} \bmod p$.
Finalmente, como p es primo, entonces Φ(p) = p-1.

Además, el texto cifrado C se puede reducir módulo p antes de calcular P_p, de manera que todos los operandos se escalan a la mitad. Con las cantidades $C_p = C \bmod p$ y $C_q = C \bmod q$, así como también $d_p = d \bmod (p-1)$ y $d_q = d \bmod (q-1)$, se obtienen las siguientes ecuaciones para Pp y Pq:

$$\mathbf{Pp = C_p^{dp} \bmod p \ \ y \ \ Pq = C_q^{dq} \bmod q}$$

La combinación de Pp y Pq para obtener P se puede hacer de acuerdo a lo visto en el capítulo 2. Para el caso especial de k = 2, $m_1 = p$, $m_2 = q$ y M = n = p.q, obtenemos $M_1 = M/m_1 = p.q/p = q$ y $M_2 = M/m_2 = p.q/q = p$.

$$\begin{aligned} P &= (P_p q(q^{-1} mod\ p) + P_q p(p^{-1} mod\ q)) mod\ n \\ &= (P_p q(q^{p-2} mod\ p) + P_q p(p^{q-2} mod\ q)) mod\ n\ (*) \\ &= (P_p (q^{p-1} mod\ n) + P_q (p^{q-1} mod\ n)) mod\ n\ (**) \end{aligned}$$

(*) De acuerdo al Pequeño Teorema de Fermat $a^{p-1} \equiv 1 \bmod p \rightarrow a^{p-2} \equiv a^{-1} \bmod p$.
(**) a(b mod c)=(ab) mod (ac)
Los valores de d mod (p-1) y d mod (q-1) se pueden tener precalculados (como se ve en la estructura de la clave privada mostrada más arriba). El resultado es una aceleración de cómputo en un factor de cuatro, comparado con la forma teórica pura.

Una justificación del elemento "coefficient" en la estructura de la clave RSA expuesta más arriba implica comprender la aplicación de la fórmula de Garner como se puede ver en [MENEZES 2001] y en [http://www.di-mgt.com.au/crt_rsa.html].
Son útiles para practicar y comprender el funcionamiento del algoritmo RSA las calculadoras disponibles en línea que se mencionan en la bibliografía.

3.2 DH – Diffie y Hellman

Una de las formas de intercambiar una clave simétrica es a través de un centro de distribución de claves (KDC = Key Distribution Center). Sin embargo existe una alternativa para que Alice y Bob puedan intercambiar esa clave sin recurrir a un KDC. Para establecer la clave ambas partes deben elegir dos números p y α. El número p es un número primo grande del orden de 300 dígitos decimales (1024 bits). El segundo número α es una raíz primitiva en el grupo $<Z_{p^*}, x>$. Tanto p como α deben hacerse públicos para que tomen conocimiento ambos participantes.
Los pasos son los siguientes:

1. Alice elige un número grande al azar a tal que $2 \leq a \leq p-2$ y calcula $A = K_{pub,A} \equiv \alpha^a \bmod p$.
2. Bob elige un número grande al azar b tal que $2 \leq b \leq p-2$ y calcula $B = K_{pub,B} \equiv \alpha^b \bmod p$.
3. Alice envía el valor A pero retiene el valor a.
4. Bob envía el valor B pero retiene el valor b.
5. Alice calcula $K_{AB} = k_{pub,B}{}^{k_{pr,A}} \equiv B^a \bmod p$.
6. Bob también calcula $K_{AB} = k_{pub,A}{}^{k_{pr,B}} \equiv A^b \bmod p$.

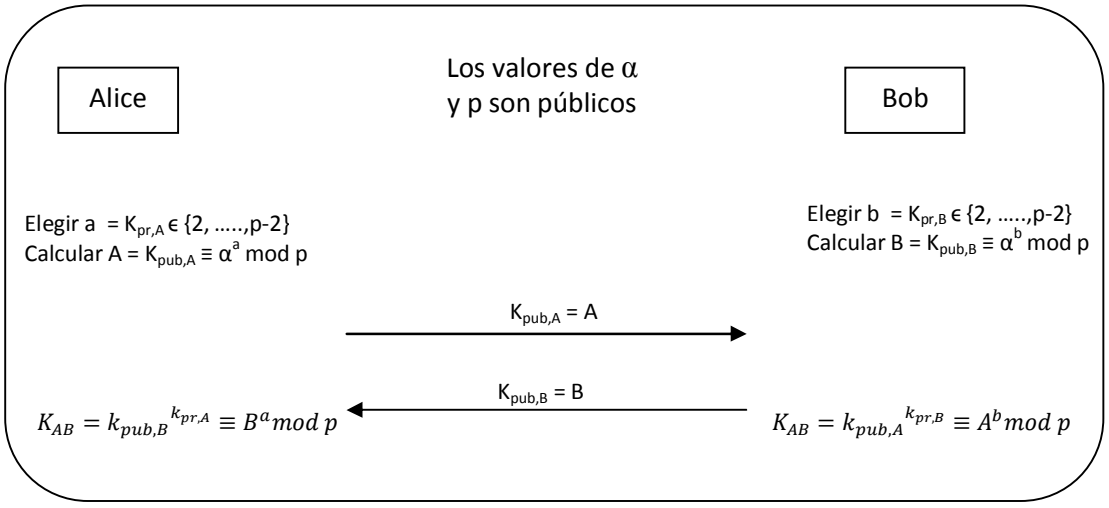


Figura1 – Intercambio de claves DH

La prueba de que el protocolo es correcto, es decir, que Alice y Bob calculan la misma clave de sesión k_{AB} es la siguiente:
Alice calcula: $B^a \equiv (\alpha^b)^a \equiv \alpha^{ab} \bmod p$
y Bob calcula: $A^b \equiv (\alpha^a)^b \equiv \alpha^{ab} \bmod p$

Y por lo tanto Alice y Bob comparten la misma clave de sesión $k_{AB} \equiv \alpha^{ab} \bmod p$. Ahora ambos pueden establecer una comunicación segura usando k_{AB} como clave para un algoritmo simétrico, por ejemplo 3DES.

Ejemplo:
Se asume p = 17 y α = 2. Los pasos son los siguientes:

1. Alice elige a = 11 y calcula $A = 2^{11} \bmod 17 = 8$.
2. Bob elige b = 13 y calcula $B = 2^{13} \bmod 17 = 15$.
3. Alice le envía el número 8 a Bob.
4. Bob le envía el número 15 a Alice.
5. Alice calcula la clave simétrica $K_{AB} = 15^{11} \, mod \, 17 = 9$
6. Bob calcula la clave simétrica $K_{AB} = 8^{13} \, mod \, 17 = 9$

El algoritmo de Diffie-Hellman es vulnerable a dos tipos de ataques: de logaritmo discreto y de hombre en el medio (MITM = Man In The Middle). Para un detalle de estos y otros tipos de ataque se puede ver [RAYMOND].

3.3 DSA – Digital Signature Algorithm

Este algoritmo viene a dar entidad al estándar DSS (Digital Signature Standard) que es el adoptado por el gobierno de los Estados Unidos para implementar la firma digital.
Vamos a analizar el algoritmo con una longitud de bit de1024 bit. Se trabaja con dos grupos cíclicos, uno más grande Z_p^*, de 1024 bit de longitud y otro más pequeño que es un subgrupo de 160 bit de Z_p^*, de esta manera se obtienen firmas de longitud más corta. Otras combinaciones posibles de *p* y *q* se muestran en la siguiente tabla:

p	q	firma
1024	160	320
2048	224	448
3072	256	512

Generación de las claves en DSA:
1. Generar un primo p tal que $2^{1023} < p < 2^{1024}$.
2. Encontrar un divisor primo q de p-1 tal que $2^{159} < q < 2^{160}$.
3. Encontrar un elemento α tal que ord(α)= q, es decir, α genera el subgrupo con q elementos.
4. Elegir un entero aleatorio *d* tal que 0 < d < q.
5. Calcular $\beta \equiv \alpha^d \bmod p$.
Las claves son:

$$\mathbf{k_{pub} = (p,q,\alpha,\beta)}$$
$$\mathbf{k_{pr} = (d)}$$

Generación de la firma

La firma digital consiste en un par de enteros (r,s). Cada uno de estos parámetros es de 160 bits de longitud, dando una longitud total de la firma de 320 bits. Dadas las claves pública y privada y un mensaje P, la firma se calcula como sigue:

1. Elegir un entero como clave efímera aleatoria k_E tal que $0 < k_E < q$.
2. Calcular $r \equiv (\alpha^{kE} \bmod p) \bmod q$.
3. Calcular $s \equiv (SHA(P) + d.r).k_E^{-1} \bmod q$.

Como se ve el mensaje P debe pasarse por una función de hash SHA-1 con una longitud de salida de 160 bits.

Verificación de la firma

1. Calcular el valor auxiliar $w \equiv s^{-1} \bmod q$.
2. Calcular el valor auxiliar $u_1 \equiv w.SHA(P) \bmod q$.
3. Calcular el valor auxiliar $u_2 \equiv w.r \bmod q$.
4. Calcular $v \equiv (\alpha^{u1} \cdot \beta^{u2} \bmod p) \bmod q$.
5. La verificación $ver_{kpub}(P,(r,s))$ es la siguiente:

$v \equiv r \bmod q$ → firma válida

$v \not\equiv r \bmod q$ → firma inválida

Si $v \not\equiv r \bmod q$, entonces la verificación falla. Esto significa que la firma o el mensaje fueron alterados o que el verificador no tiene la clave pública correcta. De otro modo la firma se considera válida.

Utilizando el programa openssl podemos generar y verificar la composición de las claves pública y privada.

OpenSSL> dsaparam -out paramdsa.pem 1024
OpenSSL> gendsa -des3 -out privkey.pem paramdsa.pem
OpenSSL> dsa -text -in privkey.pem

```
read DSA key
Private-Key: (1024 bit)
priv:
    69:a3:5e:74:6a:03:af:8a:86:ae:d9:73:12:ba:1a:
    ac:86:d1:c8:28
pub:
    00:ba:ef:ef:59:98:0c:5b:f8:ce:a1:d2:51:00:5f:
    ……..
    86:99:74:61:ac:c4:82:f3:5c
P:
    00:f7:a6:8b:17:e9:bc:29:a0:4e:28:fa:03:4f:c8:
    …..
    24:c8:b7:1f:c3:e6:f7:d1:a1
Q:
    00:8b:c6:b8:53:f9:86:a4:9a:e3:94:1e:80:98:8f:
    97:f1:6d:b8:d3:5d
G:      5c:a4:31:a1:44:96:03:12:78:35:40:70:22:bc:47:
```

```
    …..
   6a:94:79:96:02:06:a1:20
writing DSA key
-----BEGIN DSA PRIVATE KEY-----
MIIBuwIBAAKBgQD3posX6bwpoE4o+gNPyC6p6H78vvceiRG42/3LXqa7W5x91Z62
…….
A6+Khq7ZcxK6GqyG0cgo
-----END DSA PRIVATE KEY-----
```

3.4 Criptografía con Curvas Elípticas – ECC (Eliptic Curve Cryptography)

Aunque RSA es un criptosistema de clave asimétrica seguro, el precio a pagar por esa seguridad es la longitud de las clave. La criptografía de curva elíptica (ECC) es el miembro más nuevo de las tres familias de algoritmos de clave pública que tienen relevancia práctica.

La ECC brinda el mismo nivel de seguridad que RSA o DH con operandos mucho más cortos (160-256 bits contra 1024-3072 bits). La ECC se basa en el problema del logaritmo discreto, por lo cual los protocolos que se usan en este tipo de función de una sola vía (DHKE por ejemplo) se pueden implementar usando curvas elípticas. En ciertos casos la ECC ofrece ventajas de rendimiento (menos cómputos) y ancho de banda (firmas y claves más cortas) sobre las otras dos familias. Sin embargo, RSA con exponentes públicos cortos es todavía mucho más rápido que las operaciones sobre curvas elípticas.

La teoría matemática de las curvas elípticas es compleja, por lo cual se hace una introducción conceptual al tema. Para un análisis detallado se puede consultar [WASHINGTON 2008] o un análisis más simplificado en [FOROUZAN 2007].

Curvas Elípticas sobre Números Reales

La ECC se basa en el problema del logaritmo discreto, lo que hace necesario encontrar un grupo cíclico sobre el que se pueda construir el criptosistema. Ese grupo cíclico debe ser computacionalmente complicado, es decir debe tener propiedades de una sola vía robustas. Las curvas elípticas son ecuaciones cúbicas en dos variables, cuya forma general es:

$$\boldsymbol{y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3}$$

Por ejemplo, si se grafica la ecuación $x^2 + y^2 = r^2$ sobre los números reales, se obtienen todos los pares de puntos que satisfacen la ecuación de una circunferencia:

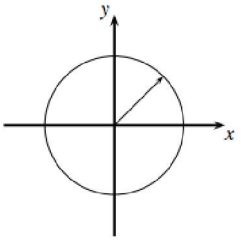


Figura 2 – Puntos que satisfacen la ecuación de una circunferencia

Las curvas elípticas sobre números reales usan una clase especial de curvas elípticas de la forma:

$$y^2 = x^3 + ax + b$$

Para los fines criptográficos estamos interesados en considerar la curva sobre un campo finito y no sobre los números reales. La elección más común es el campo de primos GF(p) donde todas la aritmética se realiza módulo el primo p.

Una curva elíptica sobre $\mathbb{Z}_p$, con p > 3, es el conjunto de todos los pares (x,y) $\in \mathbb{Z}_p$ que satisfacen:

$$y^2 \equiv x^3 + a.x + b \bmod p$$

junto con:

- un punto imaginario en el infinito **O.**
- a,b $\in \mathbb{Z}_p$.
- la condición: $4.a^3 + 27.b^2 \neq 0 \bmod p$ (curva no singular).

Por definición se analiza la curva sobre un campo finito. Si se intenta graficar una curva elíptica sobre $\mathbb{Z}_p$ no se obtiene nada parecido a una curva. No obstante, a los fines conceptuales su puede graficar sobre el conjunto de los números reales.
Por ejemplo la curva elíptica no singular (es decir, que la traza no se interseca a sí misma) $y^2 \equiv x^3 + 3x + 3$:

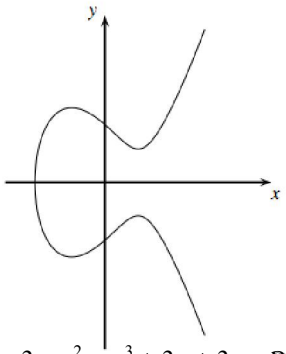


Figura 3 – $y^2 \equiv x^3 + 3x + 3$ en $\mathbb{R}$

La cuestión es encontrar una curva con un grupo cíclico grande, el cual es necesario para construir un problema de logaritmo discreto. Para ello se necesita identificar el conjunto de elementos, que en el caso de la curva elíptica son los puntos que satisfacen la ecuación de la curva. Lo siguiente es definir las operaciones permitidas sobre esos puntos. Siempre teniendo en cuenta que se cumplan las propiedades de los grupos para esas operaciones, las cuales son: cierre, asociación, conmutativa, existencia de la identidad y existencia del inverso; puede verse una concisa y clara explicación en [FOROUZAN 2007].

Operaciones sobre Curvas Elípticas

Se define una operación de grupo con el símbolo arbitrario de suma "+" (esta operación de suma es distinta de la que se definió para los enteros. La adición de dos puntos sobre la curva en este caso nos tiene que dar otro punto sobre la curva). Dados dos puntos, P = (x_1, y_1) y Q = (x_2, y_2) se tiene que calcular las coordenadas de un tercer punto R tal que:

$$P + Q = R$$
$$(x_1,y_1)+(x_2,y_2)=(x_3,y_3)$$

Esta operación de suma se puede interpretar geométricamente si se considera la curva definida sobre los números reales. Hay dos casos, la suma de dos puntos distintos (llamado suma de puntos) y la suma de un punto consigo mismo (llamado duplicación de punto).

Suma de puntos: Se calcula R = P + Q con P ≠ Q. Para ello se traza una línea que pase por P y Q y se obtiene el punto de intersección entre esta línea y la curva. Luego se hace el espejo de este punto con respecto al eje x. El punto reflejado es, por definición, R.

Duplicación de punto: Se calcula P + Q con P = Q, por lo tanto R = P + P = 2P. Se dibuja una línea tangente al punto P y obtenemos un segundo punto de la intersección entre esta línea y la curva elíptica. Luego se hace el espejo de este punto con respecto al eje x, obteniendo el punto R.

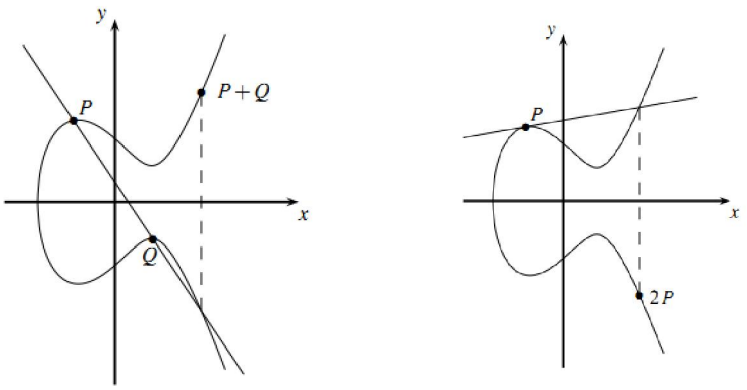


Figura 4– Suma y duplicación de puntos sobre una curva elíptica en ℝ

Estas dos construcciones geométricas se pueden pasar a expresiones analíticas. Para ello se toma la ecuación de la curva elíptica pero se la considera sobre el campo primo GF(p) en vez de los números reales. Las coordenadas del punto R (x_3, y_3) se obtienen de la siguiente forma:

$$x_3 = s^2 - x_1 - x_2 \bmod p$$
$$y_3 = s(x_1 - x_3) - y_1 \bmod p$$

donde:

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \ mod\ p; si\ P \neq Q\ (suma\ de\ puntos) \\ \\ \frac{3{x_1}^2 + a}{2y_1} \ mod\ p; si\ P = Q\ (duplicación\ de\ punto) \end{cases}$$

El parámetro *s* es la pendiente de la línea que pasa a través de P y Q en un caso o la pendiente de la tangente al punto P en el otro caso.
Se debe definir todavía al elemento identidad O, tal que P + O = P, para todos los puntos P en la curva elíptica. Como no existe ningún punto (x, y) que satisfaga esta condición, se define un punto abstracto en el infinito como el elemento identidad O. Este punto se lo puede visualizar como localizado hacia ± ∞ a lo largo del eje y.
Consistente con la definición de grupo se puede definir ahora el inverso –P de cualquier elemento del grupo P como: P + (-P) = O. Donde -P es el punto reflejado sobre el eje x, es decir si P = (x, y) entonces –P = (x, -y). En el caso de curvas elípticas sobre un campo primo GF(p), dado que $-y_p \equiv p - y_p \bmod p$, entonces: - P = ($x_p$. p – y_p).

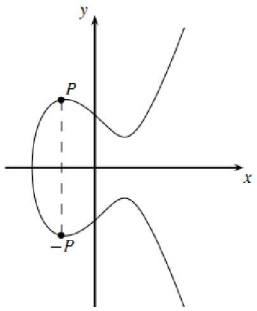


Figura 5– Inversa de un punto P en una curva elíptica

Ejemplo: Se tiene una curva sobre el campo $\mathbb{Z}_{13}$, E: $y^2 \equiv x^3 + x + 1 \bmod 13$.

Se duplica el punto P = (4,2).
2P = P + P = (4,2) + (4,2) = (x_3,y_3)

$$s = \frac{3{x_1}^2 + a}{2y_1} = \frac{3.4^2 + 1}{2.2} = 10.49 \equiv 490 \equiv 9\ mod\ 13$$

$x_3 = s^2 - x_1 - x_2 = 81 - 4 - 4 = 28 \equiv 8 \bmod 13$
$y_3 = s(x_1 - x_3) - y_1 = 9(4 - 8) - 2 = -38 \equiv 1 \bmod 13$
2P = (4,2) + (4,2) =(8,1)

Se verifica si el resultado 2P = (8,1) es realmente un punto sobre la curva introduciendo los valores en la ecuación:
$y^2 \equiv x^3 + x + 1 \bmod 13$
$1 \equiv 512 + 8 + 1 \bmod 13$
$1 = 521 \equiv 1 \bmod 13$

Encontrando los puntos sobre la curva
Un pseudocódigo para encontrar los puntos sobre la curva $E_p(a,b)$ es:

```
Entrada: p, a y b    // p es el módulo
x = 0;
while x < p
{
        w = (x^3 + ax + b) mod  p
        si w es cuadrado perfecto en  Zp → return  (x,√w) y (x, - √w)
        x = x +1
}
```

Hasta ahora se vieron las operaciones de grupo (suma de puntos y duplicación), se estableció un elemento identidad, y se vio como hallar el inverso de un punto en la curva. Se puede definir entonces el siguiente teorema (sin demostrarlo):

"Los puntos sobre una curva elíptica junto con $\mathcal{O}$ *tienen subgrupos cíclicos. Bajo ciertas condiciones todos los puntos en una curva elíptica forman un grupo cíclico*".

Ejemplo: Hallar todos los puntos sobre la curva $E_{17}(2,2)$, es decir: E: $y^2 \equiv x^3 + 2x + 2$. Todos los puntos sobre la curva forman un grupo cíclico y el orden es #E = 19. Para este grupo el orden es primo por lo tanto todos los elementos son raíces primitivas.
Comenzando con el elemento primitivo P = (5,1) se calculan todas las potencias de P, es decir, P, 2P, …….(#E)P. Esta es la lista de los elementos que se obtienen:

2P = (5, 1) + (5,1) = (6,3)	11P = (13, 10)
3P = 2P + P = (10, 6)	12P = (0, 11)
4P = (3, 1)	13P = (16, 4)
5P = (9, 16)	14P = (9, 1)
6P = (16, 13)	15P = (3, 16)
7P = (0, 6)	16P = (10, 11)
8P = (13, 7)	17P = (6, 14)
9P = (7, 6)	18P = (5, 16)
10P = (7, 11)	19P = $\mathcal{O}$ (Es decir s = ∞)

A partir del último punto se ve la estructura cíclica:

$$20P = 19P + P = \mathcal{O} + P = P$$
$$21P = 2P$$

(0,6)	(0,11)
(3,1)	(3,16)
(5,1)	(5,16)
(6,3)	(6,16)
(7,6)	(7,11)
(9,1)	(9,16)
(10,6)	(10,11)
(13,7)	(13,10)
(16,4)	(16,13)

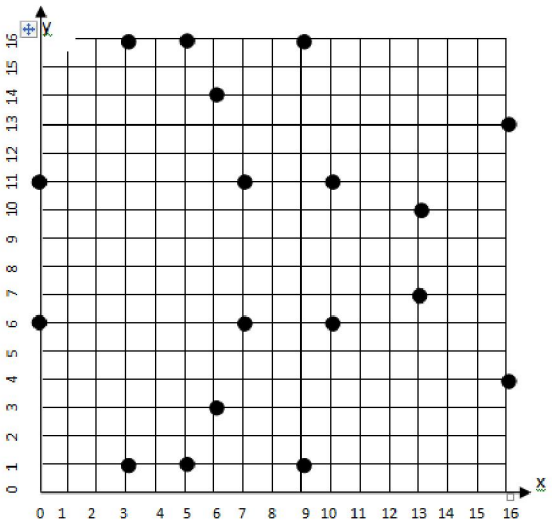


Figura 6 – Puntos sobre una curva elíptica GF(17)

Para establecer un criptosistema de logaritmo discreto es importante conocer el orden del grupo. El teorema de Hasse determina un número aproximado de puntos:
"Dada una curva elíptica E mod p, el número de puntos sobre la curva es representado como #E y está limitado por:

$$p + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}"$$

Esto nos permite deducir que si se precisa una curva elíptica con 2^{160} elementos, entonces se tiene que usar un número primo de aproximadamente 160 bits.

Intercambio de claves Diffie-Hellman con Curvas Elípticas (ECDH)

Es posible realizar un intercambio de claves usando curvas elípticas aplicando el mismo concepto que en Diffie-Hellman (DHKE). Se lo denomina intercambio de clave Diffie-Hellman sobre curvas elípticas (ECDH). Es necesario definir los parámetros de dominio, es decir, una curva elíptica apropiada sobre la cual se podrá trabajar y una raíz primitiva de esa curva:

- Se elige un primo p y una curva elíptica:

  $$E_p(a,b): y^2 \equiv x^3 + a \cdot x + b \bmod p$$

- Se elige una raíz primitiva $P = (x_p, y_p)$
- Los parámetros del dominio son: El primo p, la curva determinada por los coeficientes a y b y la raíz primitiva P.

El intercambio de claves se efectúa como se muestra en la figura de manera similar al protocolo DH convencional.

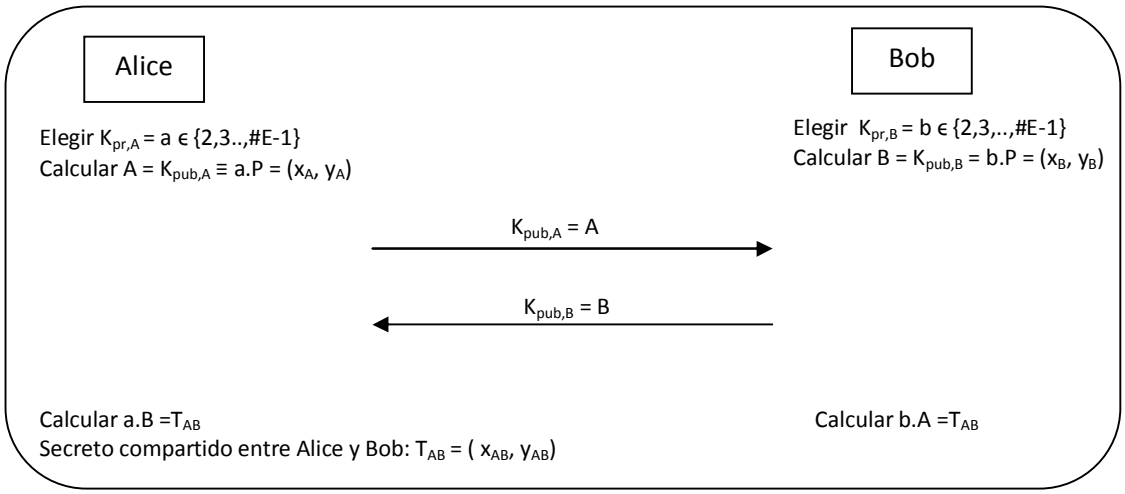


Figura 7 – Intercambio de clave DH sobre curva elíptica

Para probar que el protocolo es correcto se verifica:
Alice calcula: a.B = a.(b.P)
Bob calcula: b.A = b.(a.P)

Dado que la suma de puntos es asociativa, ambos lados calculan el mismo resultado, el punto llamado T_{AB} = a.b.P. El procedimiento completo es el siguiente:

1. Alice y Bob eligen sus claves privadas a y b, respectivamente, las cuales son dos enteros grandes.
2. Con las claves privadas ambos generan las respectivas claves públicas A y B, las cuales son puntos en la curva. Las claves públicas se calculan con multiplicación de puntos.
3. Se intercambian las claves públicas.
4. Se calcula el secreto compartido T_{AB} realizando una segunda multiplicación de puntos entre la clave pública que reciben y su propia clave privada.
5. Esta clave compartida secreta T_{AB} se usa para derivar una clave de sesión como entrada a un algoritmo simétrico.
6. Las dos coordenadas (x_{AB}, y_{AB}) no son independientes una de la otra. Dada x_{AB} la otra coordenada (y_{AB}) se calcula con la ecuación de la curva. Por lo tanto se usa solo una de las coordenadas para derivar la clave de sesión.

Capitulo 4

Aplicaciones

*"It is only with the heart that one can see rightly;
what is essential is invisible to the eye"*

Antoine de Saint-Exupéry

4 Aplicaciones de la Criptografía asimétrica

En este capítulo veremos algunas aplicaciones en las cuales tiene lugar la aplicación de la criptografía asimétrica. El campo es mucho más amplio de lo aquí presentado, pero por cuestiones de extensión no se han desarrollado temas como los generadores de números aleatorios basados en criptografía asimétrica, que se pueden ver en [MENEZES 2001], y los servicios de marca horaria (time stamping) desarrollados en la RFC 3161.

4.1 Firma Digital

La firma digital es una de las herramientas criptográficas más importantes que se usa hoy en día. Algunas de sus aplicaciones son: los certificados digitales para el comercio electrónico seguro, la firma legal de contratos y la actualización segura de software. Junto con el establecimiento de claves sobre canales inseguros, son las instancias más importantes de la criptografía de clave pública.

Fundamentos de la firma digital

En el mundo real para verificar que una persona generó un determinado mensaje nos basamos en la firma hológrafa. De la misma manera, solo la persona que generó un mensaje digital debe ser capaz de generar una firma digital para éste. Para lograr esta funcionalidad nos basamos en la criptografía de clave pública. En este esquema el remitente del mensaje usa una clave privada y el receptor usa la clave pública complementaria a la anterior. En el ejemplo de la figura 1, Bob quiere firmar un mensaje *M*. El resultado del algoritmo de firma depende tanto de la clave privada k_{pr} como del mismo mensaje *M*. Es de vital importancia que Bob mantenga su clave secreta bien asegurada, ya que es la única garantía de que él generó el mensaje. Luego de esto, tanto la firma *s* como el mensaje *M* son enviados a Alice. Para implementar la función de verificación Alice precisa el mensaje *M*, la firma *s* y la clave pública de Bob k_{pub}. Aunque la función de verificación tiene entradas grandes genera solo una salida binaria: verdadero/falso. De esta manera a cualquier mensaje firmado se le puede seguir la pista hasta llegar a su remitente, dado que la firma correspondiente solo pudo haber sido generada por el poseedor de la clave privada asociada.

Para generar una firma digital se puede utilizar cualquiera de las tres familias de algoritmos de clave pública: factorización de enteros, algoritmos discretos o curvas elípticas.

Los criptosistemas de clave pública son muy ineficientes a la hora de tratar con mensajes muy largos. En los sistemas de firma digital, de hecho los mensajes pueden ser largos, sin embargo hay que usar esquemas de clave pública. Para salvar este

problema lo que se hace es firmar un hash del mensaje que es mucho más corto que el mensaje entero.

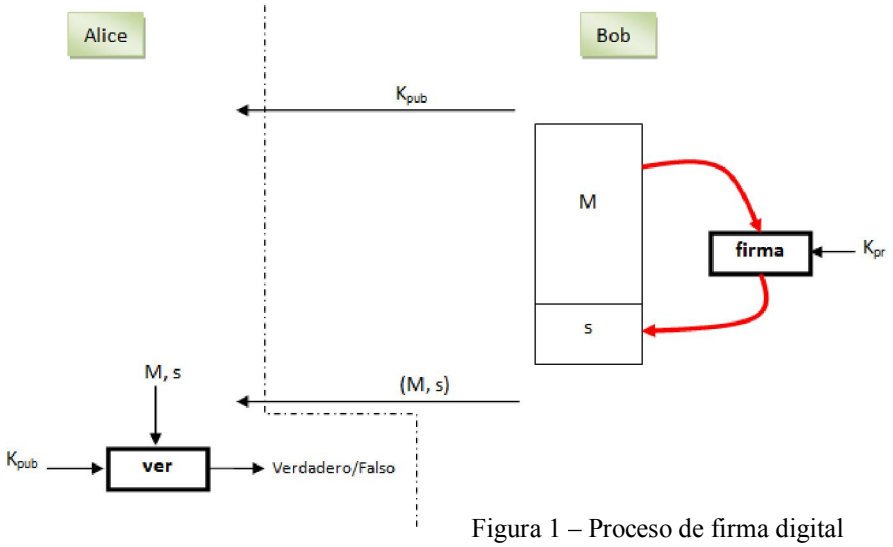


Figura 1 – Proceso de firma digital

El remitente firma el hash del mensaje y el receptor compara el hash recibido (luego de descifrarlo) con el hash que él mismo computa sobre el mensaje en texto plano recibido. El resultado final del proceso es el mismo que si se hubiese firmado todo el mensaje con la clave privada del remitente.

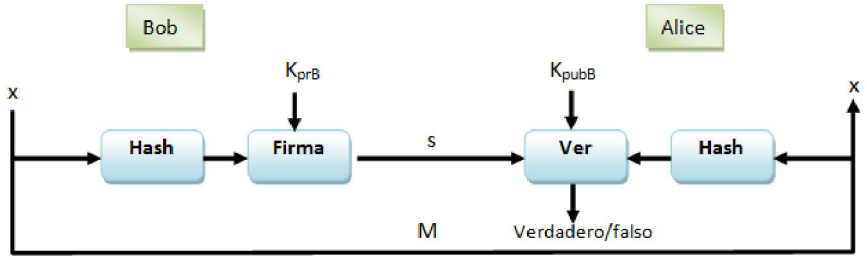


Figura 2 – Firma del hash

A continuación se muestra un protocolo de firma digital genérico:

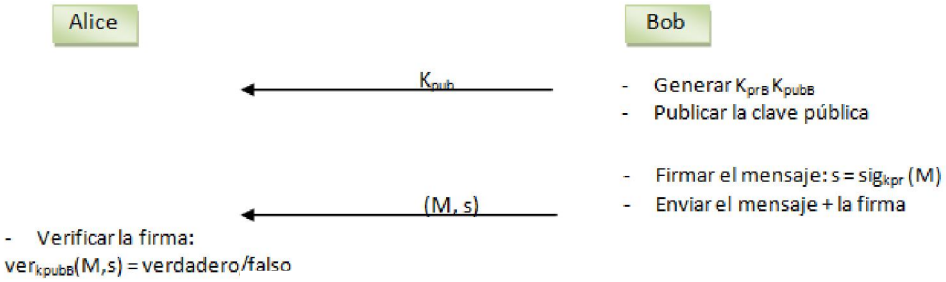


Figura 3 – Protocolo de firma digital básico

Los sistemas criptográficos vistos tenían dos objetivos fundamentales: el cifrado de datos (por ejemplo, RC4, RSA, AES) o establecer una clave compartida (por ejemplo, DH). Se podría pensar que con esto satisfacemos todas las necesidades de seguridad, sin embargo hay aspectos de la seguridad, el no repudio en particular, que están más allá del cifrado y el intercambio de claves.

Aún con los esquemas de clave asimétrica puede haber problemas para asegurar el no repudio, dado que el emisor pudo haber cambiado su par de claves al momento de querer verificar el mensaje, como también puede alegar que las claves no le pertenecen. La solución a este inconveniente es usar un tercero confiable, como por ejemplo las Autoridades de Certificación (CA), que evitarán que el emisor niegue haber enviado el mensaje.

4.2 Autenticación de entidad

Se verán técnicas (basadas en algoritmos de clave pública) diseñadas para permitir que una parte (el verificador) pueda estar segura que la identidad de la otra parte (el solicitante) es la que dice ser; esto impedirá la suplantación de identidad. En la técnica más común el verificador comprueba que un mensaje sea correcto (posiblemente en respuesta a un mensaje anterior) lo que demuestra que el solicitante conoce el secreto asociado por diseño a una entidad autorizada.

De acuerdo a la RFC 2828, la autenticación es el proceso de verificación de una identidad solicitada por o para una entidad del sistema. El proceso de autenticación consiste en dos pasos:

1. **Identificación**: Se le presenta un identificador al sistema de seguridad. (Los identificadores deberían asignarse con cuidado, porque las entidades autenticadas son la base para otros servicios de seguridad, como el servicio de control de acceso).
2. **Verificación**: Se genera o presenta información de autenticación que corrobora la unión entre la entidad y el identificador.

Veamos para aclarar conceptos, la diferencia entre autenticación de entidad y de mensaje:

Autenticación de Mensaje	Autenticación de Entidad
No sucede en tiempo real. La otra parte puede no estar presente en el proceso de comunicación. Se usa, por ejemplo, para enviar un mail, donde una parte está conectada y la otra no. Se implementa por ejemplo con HMAC[1].	Sucede en tiempo real. Ambas partes deben estar presentes. Solo se cursaran mensajes luego que los participantes se hayan autenticado. Como ejemplo podemos tomar la autenticación en redes wireless con el estándar 802.1x entre el usuario y un servidor RADIUS.

En la tabla de abajo se resumen varios métodos de autenticación de entidad expuestos en [FOROUZAN 2007], nosotros nos concentramos en el método de desafío-respuesta con clave asimétrica.
Las técnicas de clave pública pueden ser usadas para autenticación basada en el método de desafío-respuesta, con un solicitante que demuestra su conocimiento de la clave

[1] HMAC significa "Hash Message Authentication Code" que puede verse en detalle en [STALLING 2011] o en varios de los libros citados en la bibliografía.

privada de una de las siguientes formas:

a. El solicitante descifra un mensaje cifrado con su clave pública.
b. El solicitante firma digitalmente un desafío.

Es aconsejable que el par de claves públicas usadas en estos mecanismos no se use para otros propósitos, dado que los usos combinados pueden comprometer la seguridad.

METODOS DE AUTENTICACION DE ENTIDAD		
METODO	IMPLEMETACION	DESCRIPCION
PASSWORD	CONTRASEÑA FIJA	El solicitante usa siempre la misma contraseña para autenticarse.
	CONTRASEÑA DE UNA SOLA VEZ (ONE TIME)	En este esquema la contraseña se usa una sola vez, es decir se cambia con cada autenticación.
DESAFÍO-RESPUESTA	CLAVE SIMÉTRICA	En una de las variantes el verificador envía un núnico (nonce) al solicitante. Este último lo devuelve cifrado con la clave simétrica compartida, y de esta manera se autentica. Otras opciones contemplan usar marcas horarias.
	HMAC	Un ejemplo es utilizar una función de HMAC de manera que el solicitante envía una marca horaria en texto plano junto con el hash de la marca horaria concatenada con el secreto compartido. De esta manera el verificador compara el hash recibido con el hash computado sobre la marca horaria recibida (en texto plano) a la que le concatena previamente la clave compartida.
	CLAVE PÚBLICA	El verificador cifra el desafío usando la clave pública del solicitante. El solicitante descifra el mensaje usando su clave privada. La respuesta al desafío es el desafío descifrado.
	FIRMA DIGITAL	El solicitante usa su clave privada para firmar el desafío.
ZERO-KNOWlEDGE	------	Tanto en el método de clave como en el de desafío-respuesta es posible que se revele directa o indirectamente información sobre la clave del solicitante. En el método de zero-knowledge el solicitante demuestra que conoce un secreto pero sin revelarlo. Las interacciones están diseñadas de tal manera que no pueden llevar a revelar ni adivinar el secreto.
BIOMETRICO	------	Es la medición de características fisiológicas o comportamentales que identifican a una persona. La biometría mide características que no pueden adivinarse, robarse o compartir.

Una segunda precaución es que el sistema de clave pública usado no debe ser susceptible a ataques de "texto cifrado elegido", dado que un atacante pude intentar extraer información haciéndose pasar por el verificador y eligiendo desafíos estratégicos en vez de aleatorios. Incorporando un número aleatorio auto generado dentro del dato

sobre el cual se calcula la respuesta puede solucionar ambos problemas. Este dato debe estar disponible para el verificador en texto plano para que pueda llevar a cabo la verificación.

Desafío-respuesta basado en cifrado de clave pública

Consideremos el siguiente protocolo:

A → B: A (1)
A ← B : KA_{pub} (nonce; B) (2)
A →B : nonce (3)

En el paso (1) Alice se identifica. En el paso (2) Bob elige un nonce y junto con su ID se lo envía a Alice cifrado con la clave pública de ella. En el paso (3) Alice descifra el mensaje con su clave privada, extrae el nonce y se lo envía a Bob. A partir de este momento Alice está autenticada.

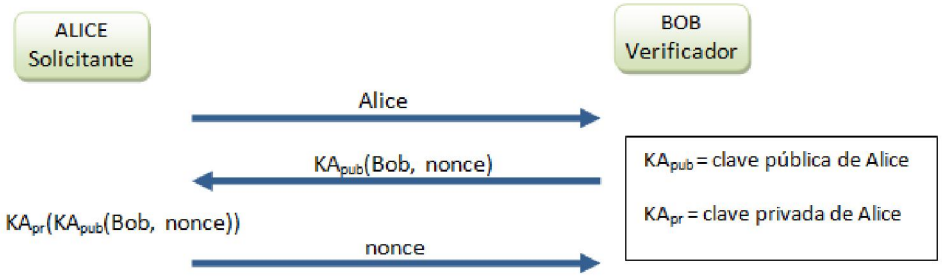


Figura 4 - Autenticación unilateral

Si la autenticación debe ser bidireccional se intercambian nonces en ambos sentidos, cifrados con las claves públicas correspondientes.

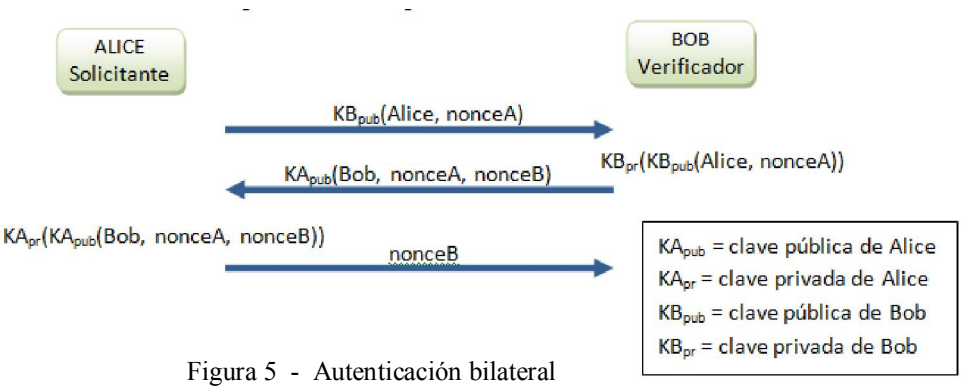


Figura 5 - Autenticación bilateral

Desafío-respuesta basado en firma digital

Veamos tres mecanismos de identificación de desafío-respuesta basados en firma digital.

1_ Autenticación unilateral con marca de tiempo

A → B : $cert_A$; t_A ;B; KA_{pr} (t_A ;B) (1)

Después de la recepción, Bob verifica que la marca de tiempo sea aceptable, que el identificador recibido B sea él y verifica (usando la clave pública de A extraída de $cert_A$) que la firma sobre esos dos campos sea correcta.

t_A: Marca de tiempo generada por A.
$cert_A$: Certificado de A que contiene su clave pública. Se puede omitir si el verificador ya posee esta información.

2_ Autenticación unilateral con nonces (fig. 6)

La dependencia a las marcas de tiempo puede reemplazarse con números aleatorios (nonces), asumiendo el costo de un mensaje adicional:

A ← B : nonceB (1)

A → B : $cert_A$; nonceA ; B; KA_{pr} (nonceA ; nonceB ; B) (2)

Bob verifica que el identificador en texto plano sea él, y usando la clave pública de Alice (extraída por ejemplo del $cert_A$) verifica que sea válida la firma sobre el nonceA sin cifrar, el nonceB enviado en el paso (1) y su identificador. El nonceA dentro de la firma previene contra ataques de texto elegido.

3_ Autenticación mutua con nonces

A ← B : nonceB (1)

A → B : $cert_A$; nonceA ; B; KA_{pr} (nonceA ; nonceB ; B) (2)

A ← B : $cert_B$; A; KB_{pr} (nonceB ; nonceA ; A) (3)

Los pasos (1) y (2) son como en el caso anterior; el paso (3) se realiza en forma análoga al paso (2).

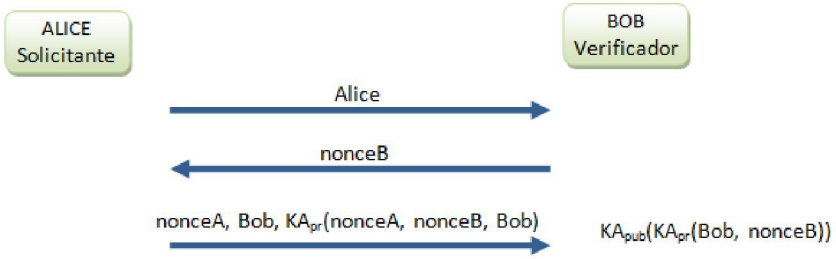


Figura 6 - Autenticación unilateral con firma digital

4.3 Protocolos

Veremos el papel que juega la criptografía de clave pública dentro de la estructura de de un protocolo de uso ampliamente difundido como es SSH.

4.3.1 SSH (Secure Shell)

SSH es un protocolo que especifica cómo conducir comunicaciones seguras en una red de datos y proporciona las siguientes características de seguridad:

Cifrado: SSH cifra todas las comunicaciones con variedad de algoritmos para elegir.
Autenticación de dos factores: SSH puede requerir un nombre de usuario/contraseña o clave pública para la autenticación. Además, estas dos opciones pueden utilizarse juntas para conformar una autenticación de dos factores.
Integridad: SSH puede crear una huella digital de los datos transferidos desde una entidad a otra, lo que garantiza que los datos no han sido modificados o manipulados de ningún modo.

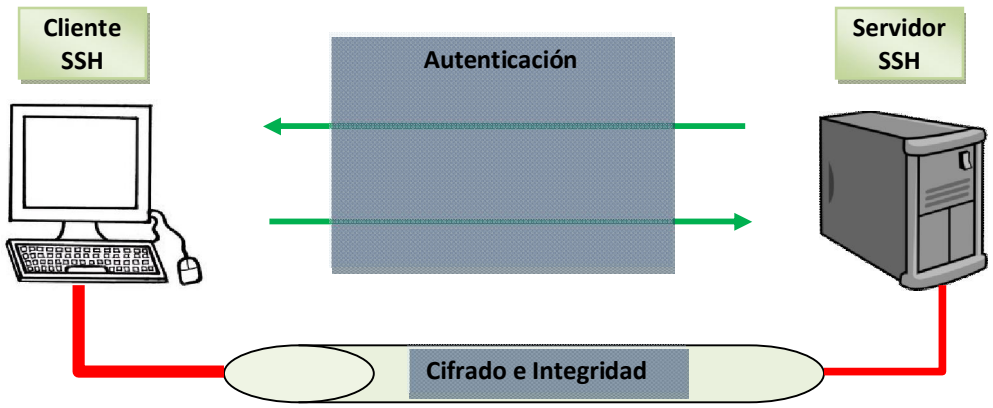


Figura 7 - Servicios de seguridad de SSH

Es decir, SSH permite conexiones de red entre equipos, con fuertes garantías de que las partes en ambos extremos de la conexión son genuinas. También asegura que los datos que pasan por estas conexiones llegan sin modificar y no son leídos por terceros no deseados.
Como pude verse en el esquema de abajo, SSH se organiza en un conjunto de tres protocolos por encima de la capa TCP.
Por medio de un análisis del proceso de asociación entre el cliente y el servidor veremos cómo los algoritmos de clave pública juegan su papel dentro de este protocolo. Por un lado se utiliza el algoritmo DH para el intercambio seguro de clave de sesión, y por otro lado nos valemos de un algoritmo asimétrico como RSA o DSA para la autenticación del servidor.

SSH User Authentication Protocol Autentica al cliente con el server usando distintos esquemas como pueden ser certificados digitales o nombres de usuario y contraseña	SSH Connection Protocol Permite varios canales lógicos sobre el túnel cifrado.
SSH Trasnport Layer Protocol Brinda los servicios de: integridad, confidencialidad, compresión de datos (opcional) y autenticación de server	
TCP Brinda servicio confiable de extremo a extremo y multiplexación de conexiones.	
IP Proporciona servicio de entrega no confiable de paquetes a través de distintas redes de datos.	

El intercambio de paquetes de la capa de transporte SSH se compone de los pasos que se detallan en la figura 9.
En la siguiente captura de Wireshark se pude observa la secuencia de conexión entre un servidor y un cliente OpenSSH con SSH v2.0.

```
 1 TCP    55672 > ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=26417 TSecr
 2 TCP    ssh > 55672 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=1
 3 TCP    55672 > ssh [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=26417 TSecr=120407
 4 SSHv2  Server Protocol: SSH-2.0-OpenSSH_5.1p1 Debian-3ubuntu1\r
 5 TCP    55672 > ssh [ACK] Seq=1 Ack=40 Win=14656 Len=0 TSval=26420 TSecr=120410
 6 SSHv2  Client Protocol: SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu6\r
 7 TCP    ssh > 55672 [ACK] Seq=40 Ack=40 Win=5824 Len=0 TSval=120411 TSecr=26420
 8 SSHv2  Client: Key Exchange Init
 9 TCP    ssh > 55672 [ACK] Seq=40 Ack=832 Win=7424 Len=0 TSval=120411 TSecr=26420
10 SSHv2  Server: Key Exchange Init
11 SSHv2  Client: Diffie-Hellman GEX Request
12 SSHv2  Server: Diffie-Hellman Key Exchange Reply
13 SSHv2  Client: Diffie-Hellman GEX Init
14 SSHv2  Server: Diffie-Hellman GEX Reply
15 TCP    55672 > ssh [ACK] Seq=1000 Ack=1696 Win=19328 Len=0 TSval=26437 TSecr=120419
16 SSHv2  Client: New Keys
17 TCP    ssh > 55672 [ACK] Seq=1696 Ack=1016 Win=8960 Len=0 TSval=121545 TSecr=27354
18 SSHv2  Encrypted request packet len=48
```

Figura 8 - Captura del handshake SSH en Wireshark

De acuerdo a la figura 9, podemos dividir el handshake de SSH en cuatro partes:

Primera Parte	Conexión TCP	Paquetes 1 - 3
	Anuncio de versiones	Paquetes 4 y 5
	Negociación de algoritmos	Paquetes 6 y 7
Segunda Parte	Intercambio de claves	Paquetes 8 -11
Tercera Parte	Comienzo del cifrado	Paquete 12
Cuarta Parte	Pedido de autenticación del cliente	---------

Primera parte

Después de esta parte las dos entidades derivan la siguiente información:

- Versión SSH
- Métodos de intercambio de claves soportados y preferidos
- Tipos de clave soportadas y preferidas por el host
- Algoritmo de cifrado de datos soportados y preferidos
- Algoritmos de integridad soportados y preferidos
- Algoritmos de compresión soportados y preferidos
- Un número aleatorio único generado por cada parte (llamado cookie)

Los paquetes 1 a 3 estableces el acuerdo de tres vías y se abre una sesión TCP entre cliente y servidor.

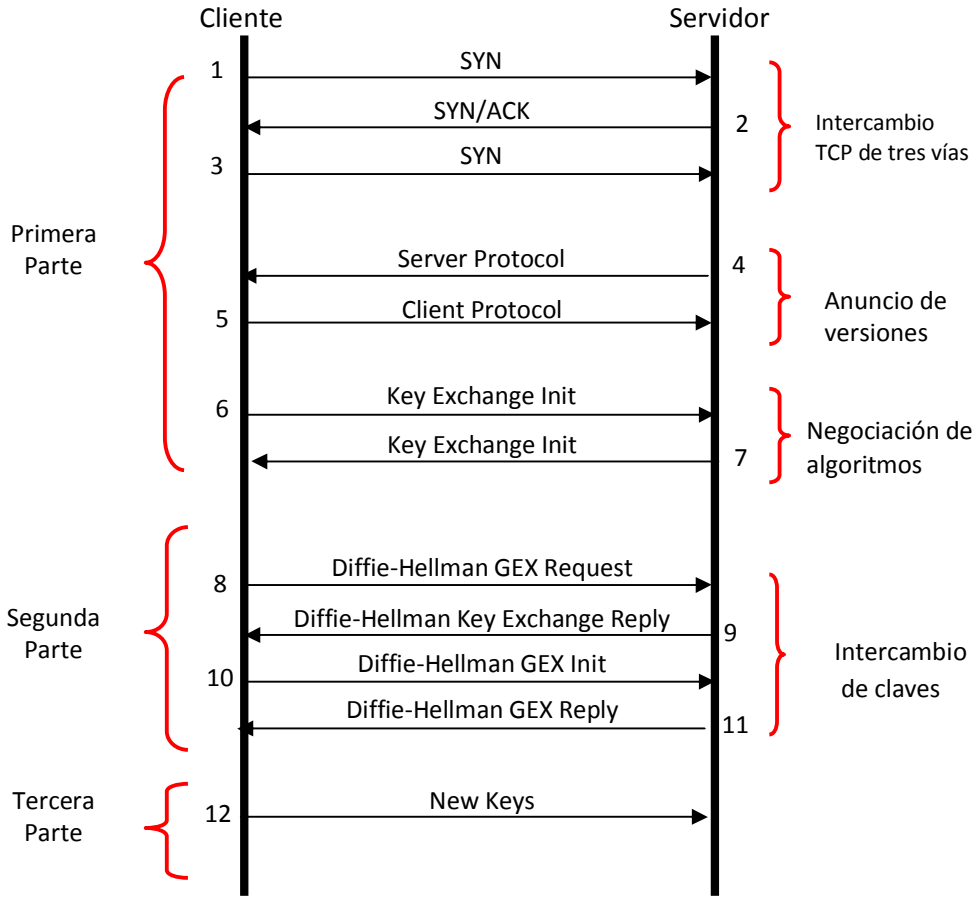


Figura 9 - Handshake SSH

Los paquetes 4 y 5 (version string announcement) anuncian la versión de SSH. Existen dos versiones del protocolo, incompatibles entre sí, SSH1 y SSH2. La versión SSH1 es menos segura dado que usa CRC32 en vez de un MAC para la integridad del mensaje (semejante a lo que pasó con WEP en redes wireless). El CRC no es una función resistente a colisiones como los son las funciones de hash MD5 o SHA. Si las versiones

de protocolo son compatibles, la conexión continúa con el siguiente paso. Si no lo son, cualquier lado puede forzar el cierre.
Los paquetes 6 y 7 (Key Exchange Init) tienen como objetivo negociar los algoritmos de intercambio de clave, de cifrado, de integridad de mensaje y de compresión. También se comunican los tipos de clave de host que se aceptan. Los métodos preferidos y soportados se presentan en una lista separada con comas. Los valores preferidos se ponen primeros en la lista.
Para la integridad de datos se usa un algoritmo de MAC que calcula un hash del mensaje sin cifrar junto con un secreto compartido y un número de secuencia de sesión, que luego se envía al receptor. El secreto compartido se establece con el intercambio de claves DH y se calcula en forma independiente por cada parte sin necesidad de transmitirlo por la red. El MAC no se incluye en los paquetes hasta tanto no se haya enviado el paquete número 10.
Con SSH existe la opción de comprimir la trasmisión. De acuerdo con la RFC 4253 Zlib es el único método soportado. En el caso que se negocie la compresión, esta tiene efecto solo sobre la carga útil del mensaje y se realiza antes de las rutinas de MAC y de cifrado.
El flujo de datos es independiente en cada dirección. El servidor podría cifrar sus mensajes usando AES128 y comprimir con zlib, y el cliente podría cifrar con 3DES y no usar compresión.
El cliente y el servidor generan una secuencia aleatoria de 16 bytes conocida como cookie. Este valor es usado en el proceso de generación de clave, para asegurar que ninguna parte pueda determinar las claves completamente por su cuenta.

Segunda parte

En esta parte se produce el intercambio de claves usando el protocolo DH elegido. Con este método ambas partes pueden derivar la misma clave secreta a través de comunicaciones en texto plano, sin la posibilidad de que un tercero que escucha las comunicaciones pueda deducir el mismo valor. Después de este intercambio ambos actores derivan la siguiente información:

- El grupo Diffie-Hellamn (y por lo tanto el tamaño del número primo).
- El número primo y un generador.
- El hash de intercambio (session ID) y el secreto compartido “k”.
- La identidad del servidor es autenticada a través de criptografía de clave pública.
- Los valores de las claves de cifrado e integridad y los vectores de inicialización IV.

En el paquete 8 (DH GEX Request) el cliente le notifica al servidor el tamaño del número primo mínimo, preferido y máximo para el grupo. En este caso tuvimos, mínimo: 0x400 = 1024, preferido: 0x400 = 1024 y máximo: 0x2000 = 8192.

En el paquete 9 (DH Key Exchange Reply) el servidor elije entre las opciones disponibles, un valor para el tamaño de número primo del grupo y se lo informa al cliente. El paquete contiene dos enteros multiprecisión que son el número primo a utilizar "p" y el generador correspondiente "g". Después de esto, ambas partes saben que grupo DH van a utilizar. Faltan solamente los paquetes 10 y 11 en el intercambio de claves antes de que se negocie suficiente información como para comenzar el cifrado de datos.
En el paquete 10 (DH GEX Init) el cliente calcula un valor basándose en p y g y se lo envía al servidor. En detalle, el cliente genera un número aleatorio x, tal que $1 < x < (p-1)/2$ y calcula $e = g^x \bmod p$, este valor e es el que se envía al servidor.

En el paquete 11(DH GEX Reply) el servidor genera su propio número aleatorio, luego calcula $f = g^y \bmod p$ y le envía f al cliente. También calcula $k = e^y \bmod p$, que es el valor del secreto compartido. El cliente hace lo mismo aplicando la fórmula $k = f^x \bmod p$. Ahora ambos comparten el mismo valor k. Esto es importante dado que k es uno de los elementos usados para crear la firma de hash, que es lo que autentica al servidor.
El hash de intercambio se crea concatenando varias partes, una de las cuales es el valor de k. También incorpora: e, f, los paquetes 6 y 7, y las cadenas de identificación de versión del cliente y servidor. Una vez juntos se los pasa por el algoritmo de hash negociado y el valor resultante se firma con la clave privada del servidor (RSA o DSA). En el paquete 11 se incluye también la clave pública del servidor. A partir de este momento el server se encuentra autenticado.

$$\mathbf{H = hash\ (C_{string} \parallel S_{string} \parallel P_6 \parallel P_7 \parallel KS_{pub} \parallel e \parallel f \parallel K)}$$
$$\mathbf{s = KS_{pr}\ (H)}$$
$$\mathbf{P_{11} = (KS_{pub} \parallel f \parallel s)}$$

C_{string} = cadena de identificación del servidor
P_6 = paquete 6 – Key Exchange Init
KS_{pub}
f = valor calculado por el servidor
s = firma digital del hash
P_{11} = paquete 11 DH GEX Reply

S_{string} = cadena de identificación del cliente
P_7 = paquete 7 – Key Exchange Init
KS_{pr}
e = valor calculado por el cliente
K = secreto compartido

En este punto se tiene toda la información para la generación de las claves. Usando el secreto compartido, el hash intercambiado H, y el identificador de sesión (que es el hash H intercambiado inicialmente, si no hubo intercambio de claves subsecuentes al inicial) se calculan las claves para los siguientes propósitos:

- Vector de inicialización, IV, del cliente al servidor: hash (K || H || "A" || session_id)
- Vector de inicialización, IV, del servidor al cliente: hash (K || H || "B" || session_id)
- Clave de cifrado, del cliente al servidor: hash (K || H || "C" || session_id)
- Clave de cifrado, del servidor al cliente: hash (K || H || "D" || session_id)

- Clave de integridad, del cliente al servidor: hash (K || H || “E” || session_id)
- Clave de integridad, del servidor al cliente: hash (K || H || “F” || session_id)

Tercera parte

En el paquete 12 (New Keys), el cliente le avisa al servidor que las claves y algoritmos deberían comenzar a trabajar a partir de este momento. Durante la sesión SSH cualquiera de las partes puede invocar un proceso de renovación de claves. La RFC 4253 recomienda que se haga una vez cada GB de datos transmitidos o una hora de actividad, lo que suceda primero. Esto hace más difícil para un atacante obtener suficiente texto cifrado para llevar adelante un criptoanálisis.

Cuarta parte

Una vez que se estableció la capa de transporte cifrada y el servidor fue autenticado, ahora es tiempo de autenticar al cliente. Para lograr la autenticación del usuario, el cliente invoca SSH-AUTH enviando primero un mensaje SSH2_MSG_USERAUTH_REQUEST especificando el método “none”. El servidor responde con la lista de métodos soportados, que generalmente son: contraseña, clave pública y basada en host. Esto ya no es visible en la captura dado que la comunicación se encuentra cifrada.

Capitulo 5

Comparación de Desempeño

"No existe arte más difícil que el de vivir. Porque para los demás artes y ciencias en todas partes se encuentran numerosos maestros. Hasta personas jóvenes creen que las han aprendido de tal manera que se las pueden enseñar a otros. Y durante toda la vida tiene uno que seguir aprendiendo a vivir, y, cosa que os sorprenderá más aún, durante toda la vida tiene uno que aprender a morir"

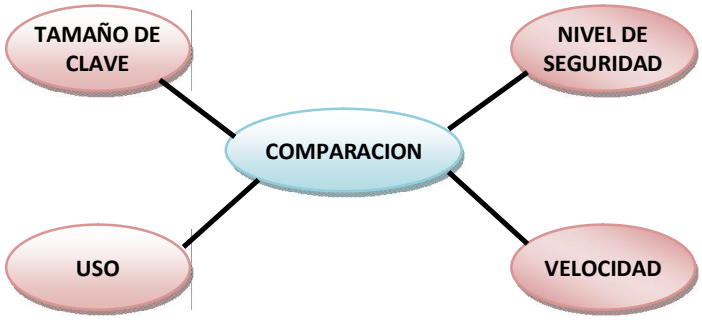
Séneca

5 Comparación de desempeño

En este capítulo se destacan algunos puntos de comparación entre los distintos algoritmos de clave pública. Es de ayuda ver la clara y concisa introducción al tema de "complejidad" que se hace en el anexo L de [FOROUZAN 2007].

5.1 Comparación de algoritmos de clave pública

A través de lo expuesto en capítulos precedentes ha quedado demostrada la necesidad que tiene el mundo moderno de los algoritmos de clave pública tanto para simplificar el manejo de claves como para hacer posible la firma digital y de esta manera facilitar una comunicación electrónica segura. Lo que resta plantear es que algoritmo de clave pública es el mejor. Esto no es sencillo, dado que hay que tener en cuenta el uso que se le dará al algoritmo, ya que el desempeño del mismo depende de su implementación en particular. Además al ser las soluciones a los problemas subyacentes en que se basan de distinta índole, se presumen posibles niveles de seguridad que pueden no ser del todo correctos. Se tienen en cuenta los siguientes factores a la hora de la comparación:



De todos ellos indudablemente el más importante es el nivel de seguridad. Pero como ya se mencionó la elección es un tanto subjetiva, dependiendo de la presunción que se haga de las distintas tecnologías de clave publica; ¿es más difícil la factorización, o el cálculo del logaritmo discreto con módulo un número primo, o el cálculo de logaritmos sobre una curva elíptica?

Vamos a analizar tres tecnologías representativas en particular (RSA, DH/DSA, EC). A todas ellas se les puede incrementar el tamaño de clave de manera de hacerlas más seguras, prescindiendo en contrapartida de la velocidad de cómputo. Los ataques más eficientes contra cada tipo de algoritmo de clave pública requieren de una cierta carga computacional que queda determinada por un parámetro de seguridad que se relaciona al tamaño de la clave.

RSA

El algoritmo de clave pública RSA implica la exponenciación módulo un número *n*, que es el producto de dos números primos grandes. El largo de la clave se refiere el largo de *n* en bits (por ejemplo 1024 bits en RSA). El ataque más eficiente contra RSA es la Criba General del Campo de Números (GNFS por sus siglas en inglés) que descompone el módulo en sus números primos originales. Si logramos averiguar los números p y q podemos calcular:

$$\Phi(n) = (p-1).(q-1)$$
$$d^{-1} = e \bmod \Phi(n)$$
$$P \equiv C^d \bmod n$$

- e y n son públicos
- p y q se calculan con GNFS

En RSA el cifrado y el descifrado se basan en la exponenciación modular como se muestra a continuación. Una forma directa de hacer la exponenciación es:

$$x \xrightarrow{CU} x^2 \xrightarrow{MUL} x^3 \xrightarrow{MUL} x^4$$

donde CU significa elevación al cuadrado y MUL multiplicación. Desafortunadamente, los exponentes e y d son número muy largos en el rango de 1024 a 3072 bits. Una exponenciación directa como la de arriba requeriría 2^{1024} multiplicaciones. Dado que la cantidad de átomos en el universo es de 2^{300}, 2^{1024} es un número imposible de computar con la forma vista. Para solucionar este problema existe el algoritmo de elevar al cuadrado y multiplicar (square-and-multiply algorithm) que se vio en el capítulo 1. Se puede determinar la complejidad de este algoritmo con un exponente E de largo de bit t+1 ($\log_2 E = t+1$). El número de elevaciones al cuadrado es independiente del valor de E, pero el número de multiplicaciones es igual al peso de Hamming (el número de unos en su representación binaria). Por lo tanto la cantidad de operaciones se representan como:

$$\#CU = t$$
$$\#\overline{MUL} = 0{,}5t \text{ (se asume igual probabilidad de unos que de ceros)}$$

Se puede ver cuántas operaciones son necesarias en promedio para una exponenciación con 1024 bits. La exponenciación directa requiere $2^{1024} \approx 10^{300}$ multiplicaciones lo que es prácticamente imposible. El algoritmo de elevar al cuadrado y multiplicar requiere en promedio solamente: 1,5 . 1024 = 1536 elevaciones al cuadrado y multiplicaciones. Esto es un ejemplo de la diferencia de un algoritmo con complejidad lineal y otro con complejidad logarítmica. En el capítulo 9 de [STALLING 2011] se puede ver en más detalle este tema.

DH y DSA

El algoritmo Diffie-Hellman y el algoritmo DSA se basan en la exponenciación módulo un número primo grande *p* cuya longitud es el largo de la clave (generalmente 1024 bits).

El problema de DH (DHP) se define como: "Dado un grupo cíclico finito G de orden n, un elemento primitivo $\alpha \in G$ y dos elementos $A = \alpha^a$ y $B = \alpha^b$ en G. El problema de DH es encontrar el elemento α^{ab} del grupo."

Supongamos que el atacante conoce un método eficiente de calcular el logaritmo discreto en Z_p^*. Entonces también puede resolver el problema de DH y obtener la clave k_{AB} con los siguientes pasos:

1. Calcula la clave privada de Alice a = k_{prA} resolviendo el problema del logaritmo discreto $a \equiv \log_\alpha A \bmod p$.
2. Calcula la clave de sesión $k_{AB} \equiv B^a \bmod p$.

Este cálculo no es factible si se elige *p* lo suficientemente grande. La forma más sencilla de intentar hacerlo es con la búsqueda por fuerza bruta para calcular el logaritmo discreto: $\log_\alpha A$. Simplemente se calculan sucesivamente las potencias del generador α hasta que el resultado es igual a A:

$$\alpha^1 = A?$$
$$\alpha^2 = A?$$
$$.$$
$$.$$
$$\alpha^a = A?$$

Para un logaritmo discreto "*a"* aleatorio, se espera encontrar la solución correcta después de verificar la mitad de los posibles valores de *a*. Esto nos da una complejidad de O(|G|) pasos, donde |G| es la cardinalidad del grupo. Para evitar este tipo de ataques en sistemas basados en logaritmos discretos, esta cardinalidad debe ser lo suficientemente grande. En Z_p^* tenemos |G| = (p-1), siendo necesario entonces (p-1)/2 pasos para calcular el logaritmo discreto. Existen otros algoritmos mucho más eficientes para atacar el problema del logaritmo: Método "Baby-Step Giant-Step" de Shanks, Método "Pollard´s Rho" y Método del cálculo indexado entre otros.

EC

Los algoritmos de curvas elípticas están basados en cálculos con puntos sobre una curva elíptica. Hay que tener en cuenta que la criptografía de curva elíptica (ECC) no es un tipo particular de sistema criptográfico, sino que ofrece a la criptografía de clave pública otra manera de hacer la matemática. Estos sistemas criptográficos son una variación de DH y DSA (ECDH y ECDSA). Cuando hablamos del tamaño de la clave nos referimos al tamaño del número primo. Los valores de las claves están en el rango de 160 a 200 bits.

De acuerdo a lo visto, si un atacante quiere quebrar el algoritmo ECDHP cuenta con los siguientes datos: E, p, P, A, y B. Lo que precisa es calcular el secreto compartido entre las partes T_{AB} = a.b.P. Esto se llama el problema de DH sobre curva elíptica (ECDHP). La manera de resolver el ECDHP es resolver alguno de estos logaritmos discretos:

$$a = \log_P A \quad o \quad b = \log_P B$$

Generalmente los mejores ataques contra el ECDHP son más débiles que los mejores algoritmos para resolver el problema de logaritmos discretos módulo p y que los algoritmos de factorización que se usan en los ataques a RSA. En particular, el algoritmo de cálculo indexado, que es muy eficiente contra el DLP módulo p, no es de utilidad contra las curvas elípticas. Los únicos ataques posibles son los algoritmos de logaritmo discreto genéricos:

"Baby-Step Giant-Step" y "Pollard´s Rho". Dado que la cantidad de pasos requeridos para estos ataques es aproximadamente igual a la raíz cuadrada de la cardinalidad del grupo, entonces se debe usar un orden de grupo de al menos 2^{160}. Esto implica que el primo p que se usa para la curva elíptica debe tener una longitud de 160 bits. Si se ataca a ese grupo con un algoritmo genérico, se precisan cerca de $\sqrt{2^{160}} = 2^{80}$ pasos. Un nivel de seguridad de 80 bits brinda una seguridad de mediano plazo. Se pueden usar curvas elípticas con largos de hasta 256 bits, lo que brinda un nivel de seguridad de hasta 128 bits.
En la siguiente tabla se muestran las aplicaciones más importantes de la criptografía de clave pública:

FIRMA DIGITAL	Brindan autenticación del remitente e integridad del mensaje. Se la puede implementar utilizando RSA, DSA o ECDSA.
CIFRADO/ DESCIFRADO	Para comunicaciones del tipo almacenamiento y reenvío como el correo electrónico, es más eficiente realizar el cifrado de datos con algoritmos simétricos. La clave simétrica se cifra con la clave pública del receptor. Este manejo de claves se puede implementar con RSA o con DH en modo "half-static" donde se usa una clave de largo plazo y otra de corto plazo.
INTERCAMBIO DE CLAVE ONLINE	Para comunicaciones online como la navegación web se pude aplicar el método anterior para establecer la clave de sesión. Sin embargo es posible obtener una propiedad llamada "perfect forward secrecy" (PFS) lo que significa que si se compromete la cave de largo plazo que se está usando, igualmente las sesiones pasadas permaneces seguras. DH y ECDH brindan esta propiedad cuando se usan en modo efímero donde ambas claves son de corto plazo.

Hay dos modos de DH, dependiendo de si las claves generadas para una combinación particular de g y p se reutilizan. DH estático reutiliza los valores de las claves y es por lo tanto más rápido. DH efímero crea un nuevo conjunto de claves para cada transacción. Las ventajas de las claves efímeras es que después de la transacción pueden ser desechadas, brindando PFS (Perfect Forward Secrecy), e incluso una irrupción en la máquina no ayuda a recuperar la información que fue intercambiada. El emisor y el receptor pueden usar modos diferentes por lo que una combinación DH efímero-estático es un ejemplo con un emisor efímero y un receptor estático.

Comparación de los niveles de seguridad

Para poder realizar esta comparación debemos establecer que largos de clave en los distintos algoritmos ofrecen similar nivel de seguridad contra los mejores ataques conocidos hasta el momento. Como nivel de referencia se toma el módulo de 1024 en RSA y se determinan los largos de clave en DH, DSA y EC que ofrecen el mismo nivel de seguridad.
La comparación con DH y DSA es directa. El algoritmo GNFS requiere el mismo tiempo para factorizar un módulo RSA de 1024 bits que para calcular el logaritmo discreto con un módulo de 1024 bits en DH o DSA.
La comparación con las curvas elípticas es más complicada. Se estima que factorizar un módulo RSA de 1024 bits con GNFS requiere $3x10^{11}$ MIPS-years[1]. Tomando para una PC

[1] Un MIPS-years es un tiempo de un año con una computadora que realiza un millón de instrucciones por segundo.

un valor de 300 MIPS, da un valor de 2^{30} años de PC para factorizar un módulo de 1024 en RSA (2^{30}/x años si usamos x PCs).
Un ataque de búsqueda de colisión en paralelo sobre curvas elípticas requiere alrededor de $2^{k/2}$ sumas en curvas elípticas (k = tamaño de la clave en bits). En un ataque basado en software con una PC, cada suma en curvas elípticas precisa alrededor de 2^{-14} segundos (2^{-39} años), dando un tiempo total para el ataque de $2^{k/2 - 39}$ años de PC. Esto sería igual a los 2^{30} años de PC para factorizar un módulo de 1024 bits en RSA cuando k = 138, lo que representa una curva elíptica pequeña. Una buena comparación práctica de niveles de seguridad entre RSA y ECC se puede encontrar en [LENSTRA 1999].

Comparación de velocidad

Ahora podemos comparar las velocidades de cálculo que nos dan los distintos algoritmos para longitudes de clave equivalentes (en términos de nivel de seguridad). La comparación depende del uso de la criptografía de clave pública que se evalúa: firma digital, cifrado/descifrado, e intercambio de claves online.
La tabla extraída de [GUDURIC] muestra los tiempos de cómputo para las operaciones asociadas con la firma digital.

	RSA con 1024 bits (e = 3)	**DSA con 1024 bits**	**ECDSA con 168 bits sobre GF(p)**
Firma	43	7	5
Verificación	0,6	27	19
Generación de clave	1100	7	7
Generación de parámetros	-----	6500	muy grande

Comparación de tiempos en firma digital [mseg] con un µP a 200 MHz

A los fines de tener un punto de comparación con algún algoritmo simétrico se tomó de [NARASIMHAM] la siguiente tabla:

Tamaño del Texto	**Cifrado DES**	**Descifrado DES**	**Cifrado RSA**	**Descifrado RSA**
128 bits	0.054945	0.00001	0.0549	0.0549
256 bits	0.054946	0.00001	0.1098	0.0549
512 bits	0.070976	0.00052	0.2197	0.1648
1 k	0.1418	0.0010	0.3846	0.3296
2 k	0.2835	0.0020	0.7142	0.6593
5 k	0.6816	0.0084	1.7032	1.7032
10 k	1.3601	0.0142	3.402	3.402

Tiempos de ejecución para cifrado y descifrado en RSA y DES (en segundos)

Para las operaciones de firma y verificación de firma, DSA y ECDSA sobre GF(p) son comparables. Sin embargo, RSA es más lento para firmar, pero mucho más rápido para verificar la firma (prestar atención que el exponente público e tiene un peso de Hamming bajo).
Para las operaciones de cifrado/descifrado la comparación es similar a la de la firma digital. DH es comparable con ECDH sobre GF(p) y RSA es más lento para descifrar y mucho más

rápido para cifrar. Para el intercambio de claves online con PFS (Perfect Forward Secrecy), DH y ECDH son comparables pero RSA es mucho más lento dado que tiene que crear el par de claves RSA de un solo uso.
Las curvas elípticas son comparables a DSA/DH. Las diferencias aparecen con RSA, que tiene un proceso de verificación de firma muy rápido, lo cual es muy útil en sistemas basados en certificados, donde la operación más común es la verificación de una firma.

Diferentes escenarios
No hay un algoritmo que sea absolutamente mejor que otro, cada uno ofrece sus ventajas relativas dependiendo de la implementación en la que se lo aplica.
En los *sistemas basados en certificados* el par de claves de la Autoridad de Certificación (CA) se usa para firmar y verificar las firmas en certificados y Listas de Revocación de Certificados (CRL). Cada certificado y CRL se firma una sola vez pero se verifica miles de veces. Por lo tanto, la operación más crítica es la verificación de la firma. Evidentemente RSA con su propiedad de rápida verificación de firma juega un papel muy importante.
Para *operaciones online* si se usa RSA implica resignar la propiedad de PFS o tolerar la lenta generación de los pares de claves RSA para cada sesión. En este caso es más ventajoso utilizar DH o ECDH.
En las *comunicaciones inalámbricas* los costos de transmisión son altos, por lo cual se tornan atractivas las curvas elípticas debido a su pequeño tamaño de clave. Con las curvas elípticas, las firmas digitales son pequeñas y también las claves simétricas cifradas, lo que se traduce en menor cantidad de bits a transmitir.
Para aplicaciones como *correo electrónico seguro* donde el correo se cifra y se firma, el mensaje se firma una vez y la firma se verifica en cada receptor. La clave simétrica que se usa para cifrar el mensaje se cifra en el emisor con la clave pública de cada receptor, y cada receptor tiene que descifrar su clave simétrica correspondiente. El número total de cifrados es igual que el de descifrados pero todos los cifrados se hacen en un solo lugar (el remitente). Por lo tanto las operaciones de tiempo más críticas son la verificación de firma y el cifrado de la clave pública. Esto le da una pequeña ventaja a RSA pero no tanto como en el caso sistemas basados en certificados.

"Every little action of the common day makes or unmakes character"
De Profundis – Oscar Wilde

6 Conclusiones y trabajo futuro

A raíz de la exponencial evolución de Internet a nivel global, y de las comunicaciones y redes de datos en general; asegurar las mismas ha pasado a ser un aspecto de capital importancia, siendo un requisito insoslayable en el diseño e implementación de redes nuevas.

La única manera que contamos en el presente (y en el futuro inmediato) de salvaguardar la información es por medio del cifrado de datos. La criptografía nos brinda la confidencialidad, autenticidad y no repudio de la información dondequiera que esta se encuentre. Por otro lado, debemos asumir el costo de la sobrecarga de cómputo, que puede llegar a comprometer la viabilidad de muchos sistemas; como así también la falta de uniformidad a nivel global en lo que respecta al método de cifrado.

Los algoritmos de cifrado confían en la tremenda dificultad de resolver ciertos problemas matemáticos, no obstante, estos algoritmos que alguna vez fueron efectivos van cayendo ante el aumento de la potencia de cálculo y de los avances en el criptoanálisis. Para contrarrestar esta circunstancia, es necesario aumentar el largo de la clave a fin de mantener el mismo nivel de seguridad. La otra solución es encontrar algoritmos de mayor complejidad algebraica, área esta de incesante y creciente actividad. En [BERNSTEIN 2009] se plantea esta situación en el marco del advenimiento de las computadoras cuánticas: "*Imagine that it's fifteen years from now and someone announces the successful construction of a large quantum computer. The New York Times runs a front-page article reporting that all of the public-key algorithms used to protect the Internet have been broken. Users panic. What exactly will happen to cryptography*?"

Cabe hacer una digresión acerca de la complementariedad de los algoritmos de clave privada y de clave pública. Si bien estos últimos son más nuevos y más versátiles que los primeros, esto no implica que los algoritmos simétricos vayan a ser reemplazados, ya que cada uno tiene sus ventajas relativas. Los algoritmos de clave privada establecen su predominio a la hora de cifrar grandes cantidades de datos, dado que el volumen de cómputo asociado al empleo de clave pública podría hacer inviable la aplicación. Por otro lado los algoritmos de clave asimétrica prevalecen al momento de implementar el intercambio de claves, las firmas digitales y garantizar el no repudio.

En cuanto a la conveniencia de utilización de un algoritmo de clave pública sobre otro, vimos que no es fácil hacer una comparativa en forma directa dada la disimilitud de los problemas subyacentes en los que se basan. Para una primera aproximación es fundamental evaluar el tipo de aplicación en la que va a utilizarse el algoritmo. Por otro lado, es de gran importancia el nivel de seguridad, que se evaluará en función del tiempo necesario para quebrar el algoritmo con el actual estado del arte del hardware y del software. Finalmente no hay que dejar de lado la cuestión de la velocidad para el

cifrado/descifrado que, dependiendo de la aplicación, puede ocupar un lugar más relevante.

Como línea de investigación para el trabajo de tesis evaluaré la posibilidad de proponer una alternativa, basada en hardware reconfigurable (FPGA), al sistema criptográfico de Mensajes Navales de la Armada Argentina, que actualmente se basa en un equipo suizo discontinuado llamado Gretacoder 505. Uno de los temas a analizar es el reemplazo del sistema utilizado para la distribución de las claves simétricas, que se lleva a cabo en forma personal con una valija con custodia donde se transportan los módulos de cifrado.

7 Referencias

- [FOROUZAN 2007], FOROUZAN Behrouz, "*Cryptograpy & Network Security*", McGraw-Hill, 2007
- [STALLINGS 2011], STALLINGS William, "*Cryptography and Network Security Principles and Practice*", Pearson, 2011.
- [WASHINGTON 2003], WASHINGTON Lawrence. *"Elliptic Curves : Number Theory and Cryptography"*, CRC Press, 2003.
- [BUCHMANN 2001], BUCHMANN Johannes. "*Introduction to Cryptography*", Springer, 2001.
- [RHEE 2003], RHEE Man Young. *"Internet Security"*, Wiley, 2003.
- [MENEZES 2001], MENEZES Alfred "Handbook of Applied Cryptography", CRC Press, 2001. Disponible en Internet: http://cacr.uwaterloo.ca/hac/
- [GOLDREICH 2004], GOLDREICH Oded. *"Foundations of Cryptography"*, Cambridge, 2004.
- [KÜSTERS 2011], KÜSTERS Ralf y WILKE Thomas, "Moderne Kryptographie", Vieweg+Teubner, 2011.
- "*Einführung in die Kryptographie*", Network Associates Inc. und Tochtergesellschaften, 1990.
- [SELKE 2000], SELKE Gisbert, "*Kryptographie*", O´REILLY, 2000
- [BERNSTEIN 2009], BERNSTEIN Daniel, "*Post -Quantum Cryptography*", Springer, 2009
- [GROßSCHÄDL 2000] GROßSCHÄDL Johann "*The Chinese Remainder Theorem and its Application in a High-Speed RSA Crypto Chip*", Annual Computer Security Applications Conference (ACSAC), 2000.
- [RAYMOND], RAYMOND Jean-Francois, "*Security Issues in the Diffie-Hellman Key Agreement Protocol*".
- [WASHINGTON 2008], WASHINGTON Lawrence, "*Elliptic Curves: Number Theory and Cryptography*", Chapman & Hall/CRC, 2008.
- [GAVALA], GAVALA, Javier Cobos, "*Introducción a la Matemática Discreta*".
- HOLOWCZAK Richard. *"RSA Demo Applet"*. [citado 12 de marzo 2012]. Disponible en Internet: http://cisnet.baruch.cuny.edu/holowczak/classes/9444/rsademo/#overview
- [WIENER 1989],WIENER Michael, "*Cryptanalysis of Short RSA Secret Exponents*", 1989.
- RFC 3161 "*Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*"
- [GUDURIC], GUDURIC Jana, *"Performance of public – key cryptography algorithms"*.
- [NARASIMHAM], NARASIMHAM Challa, *"Evaluation of Performance Characteristics of Cryptosystem Using Text Files"*.
- [KARU], KARU Priit, *"Practical Comparison of Fast Public-key Cryptosystem"*.
- [LENSTRA 1999], LENSTRA Arjen, "*Selecting Cryptographic Key Sizes*
- *in Commercial Applications*", 1999.
- [PKCS#1], RSA Laboratories, *"RSA Security Inc. Public-Key Cryptography Standards (PKCS)"*, [citado 12 de marzo 2012]. Disponible en Internet:

ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf

- *"RSA"* [citado 12 de marzo 2012]. Disponible en Internet: http://es.wikipedia.org/wiki/RSA.
- ROLLAND Robert. "Public Key Cryptography", [citado 12 de marzo 2012]. Disponible en Internet: http://tlapixqui.izt.uam.mx/cimpa/files/rolland/pub_key.pdf
- *"Miller-Rabin Primality Test",* [citado 12 de marzo 2012]. Disponible en Internet: http://www.cryptomathic.com/labs/rabinprimalitytest.html
- *"Primality Test"* [citado 12 de marzo 2012]. Disponible en Internet: http://en.wikipedia.org/wiki/Primality_test
- "*Diffie-Hellman*". [citado 12 de marzo 2012]. Disponible en Internet: http://es.wikipedia.org/wiki/Diffie-Hellman
- "Digital Signature Algotithm". [citado 12 de marzo 2012]. Disponible en Internet: http://en.wikipedia.org/wiki/Digital_Signature_Algorithm
- "Modular Arithmetic Calcualtor". Disponible en Internet: http://ptrow.com/perl/calculator.pl
- "Euler´s Totien Calculator". Disponible en Internet: http://www.javascripter.net/math/calculators/eulertotientfunction.htm
- RSA Calculator. Disponible en Internet: http://people.eku.edu/styere/Encrypt/RSAdemo.html