ECDH

Este documento no pretende ser un curso exhaustivo, en el mejor de los casos, únicamente puede considerarse como un conjunto de notas complementarias en alguna asignatura. Por supuesto, todo documento es susceptible de mejora, cualquier sugerencia o comunicación de error u omisión será bienvenida.

Uno de los algoritmos más utilizados en protocolos de comunicación es, curiosamente, el primero que se propuso y en el que, de una forma u otra se basa el resto, el algoritmo de Diffie y Hellman de intercambio público de clave.

El algoritmo DH fue propuesto considerando el grupo de residuos módulo un cierto número (primo) p y considera el problema del logaritmo discreto como base para garantizar la seguridad computacional. Actualmente, el algoritmo ha sido modificado para utilizar el grupo definido por una curva elíptica. La adaptación es más bien directa, permitiendo además el uso de claves de menor tamaño sin afectar a la seguridad.

Algoritmo de intercambio público de clave

El algoritmo considera los siguientes parámetros:

- Una determinada curva elíptica $y^2 \equiv x^3 + ax + b \pmod{n}$
- Un punto G de la curva. El subgrupo generado por G determina el conjunto de puntos que se considerará.

Todos estos parámetros son de acceso público. El algoritmo se basa en que, dado un entero k cualquiera, el cálculo del punto $H = kG$ de la curva es sencillo, pero, sin embargo, dado H, el cálculo del entero k (número de veces que es necesario componer G para obtener H) es complejo. Este problema es la versión del cálculo del logaritmo discreto cuando se consideran grupos definidos por curvas elípticas.

El método ECDH se resume en el Algoritmo 1, en esencia idéntico al algoritmo DH original.

Elección de parámetros

Un tema esencial para garantizar la seguridad del algoritmo ECDH es que el subgrupo generado por el punto escogido tenga un orden que impida el uso de algoritmos eficientes para el cálculo del logaritmo discreto.

Algoritmo 1 Producto en una curva elíptica de un punto por un escalar

Entrada: Una curva elíptica $y^2 \equiv x^3 + ax + b \pmod{n}$
Entrada: Un punto G de la curva
Entrada: Dos interlocutores *Alice* y *Bob*
Salida: Un punto P de la curva comunicado de forma segura
Método
$\langle Alice \rangle$ Generar aleatoriamente el secreto k_A // entero
$\langle Bob \rangle$ Generar aleatoriamente el secreto k_B // entero
$\langle Alice \rightarrow Bob \rangle$ Enviar $S_A = k_A G$
$\langle Alice \leftarrow Bob \rangle$ Enviar $S_B = k_B G$
$\langle Alice \rangle$ Calcular $H = k_A S_B$
$\langle Bob \rangle$ Calcular $H = k_B S_A$
FinMétodo.

Como hemos comentado, por una parte, existe un algoritmo para calcular el número de puntos que pertenecen a la curva, por otra parte, no se conoce un método polinómico para calcular el orden de un subgrupo. Una implementación efectiva del algoritmo ECHD necesita escoger un punto que garantice que el subgrupo generado sea de tamaño suficiente.

Un procedimiento para la elección del punto generador se muestra en el Algoritmo 2 y tiene en cuenta el Teorema de Lagrange:

> El orden de todo subgrupo de un grupo es un divisor del orden del grupo que lo contiene.

y el Teorema de Euler, que enunciado para adaptarse al contexto de las curvas elípticas puede enunciarse como:

> Todo elemento P de un grupo cumple que $mP = 0$ siendo m el orden del grupo y 0 el elemento neutro.

El algoritmo tiene en cuenta que, siendo N el número de puntos en la curva elíptica, cualquier punto P cumple que:

$$NP = 0$$

considerando un entero p_k (valor primo de tamaño suficiente divisor de N):

$$p_k(hP) = 0$$

por lo que, si el punto P es tal que $hP \neq 0$, podemos asegurar que su orden es p_k.

Algoritmo 2 Elección de un punto P de una curva elíptica cuyo subgrupo generado tiene un orden elevado.

Entrada: Una curva elíptica $y^2 \equiv x^3 + ax + b \pmod{n}$
Entrada:
Salida: Un punto P de la curva tal que el orden del subgrupo generado es alto
Método
Calcular N, el orden de la curva elíptica
Descomponer $N = p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$ // p_i factor primo
Escoger p_k primo de tamaño suficiente // tamaño deseado del subgrupo
Calcular $h = N/p_k$ // cofactor
$G = 0$
Mientras $G == 0$ **hacer**
Escoger aleatoriamente un punto P de la curva
Calcular $G = hP$
FinMientras
Devolver G
FinMétodo.

Habitualmente, los parámetros de la curva (valores a, b y n que la definen, y el orden de esta), el punto G a utilizar, el orden del subgrupo generado por G y el cofactor h no se calculan sino que se escogen de un catálogo publicado por la NSA.

Un ejemplo de curva utilizada es $Curve25519$[1], que puede definirse como los puntos que cumplen:

$$y^2 \equiv x^3 + 486662x^2 + x \pmod{2^{255} - 19}$$

considerando el punto con coordenada $x = 9$, el subgrupo generado tiene orden $2^{252} + 27742317777372353535851937790883648493$.

Existe un catálogo publicado por la NIST (FIPS-186-3 apéndice D) que contiene un listado de *curvas recomendadas*.

Seguridad

Teniendo en cuenta que el algoritmo ECDH devuelve las coordenadas de un punto de una curva elíptica conocida, sólo una de las coordenadas debe utilizarse como secreto compartido. Habitualmente, se considera un resumen de la componente x, truncando (si es necesario) de algún modo el hash cuando el tamaño excede el de la clave a utilizar (SHA devuelve un mínimo de 160 bits mientras que AES utiliza una clave mínima de 128 bits).

[1] Definida originalmente por Diffie y Hellman.

Dejando de lado cualquier ataque basado en explotar un canal lateral, atacar un sistema basado en curvas elípticas supone implementar un método eficiente para el cálculo del logaritmo discreto.

El algoritmo *baby-step giant-step* (BSGS) es uno de los métodos existentes y basa su ataque en el precomputo y almacenamiento de un cierto número de productos. Para una correcta ejecución este número es $\mathcal{O}(\sqrt{n})$, donde n es la talla en bits del orden del grupo utilizado.

Considerando la curva $Curve25519$, implementar el algoritmo BSGS supondría alamacenar $8{,}50706 * 10^{37}$ puntos de la curva en una tabla hash, si estos se pudieran codificar utilizando un byte (obviamente no es posible), se necesitarían aproximadamente 10^{14} yottabytes para almacenar la tabla hash (absolutamente inabordable en 2018).

Otras aproximaciones para el cálculo del logaritmo discreto no consideran ese compromiso entre espacio y tiempo de ejecución, planteando soluciones que resuelven el problema con complejidad temporal $\mathcal{O}(\sqrt{n})$. Considerando la misma curva, realizando 10^{10} comprobaciones por segundo, se necesitaría la vida de 10^9 universos para resolver el problema (se estima que, aproximadamente, han transcurrido 10^{18} segundos desde el Big Bang).

Dejamos de lado en este documento consideraciones acerca de posibles debilidades de las curvas catalogadas y de uso recomendado.