The Static Diffie-Hellman Problem

or

A $\sqrt[3]{n}$ algorithm for finding some discrete logs

Robert Gallant

Certicom Corp.

Overview

- an algorithm for finding discrete logarithms
- applicability of algorithm to (attacking) existing protocols/systems
- an interpretation of the algorithm as a reduction

What is the discrete log problem

Given a group $<G>$ of prime order n generated by the group element G, the **discrete log problem** (DLP) (in $<G>$) is:

Given a random $A \in <G>$, find the integer a such that $A = aG$.

An algorithm that finds and outputs the required integer a is said to solve the discrete logarithm problem.

Many cryptographic protocols are based on the (assumed) difficulty of solving the discrete log problem in certain groups.

Shanks $\sqrt{n}$ algorithm for solving DLP

Shanks's baby-step giant-step algorithm (BSGS) finds the integer a such that $A = aG$ with about $2\sqrt{n}$ group operations.

Basic idea of algorithm: Let $q = \lceil \sqrt{n} \rceil$.
Imagine the group elements arranged as follows:

Shanks $\sqrt{n}$ algorithm for solving DLP

$$
\begin{array}{cccc}
0G & 1G & 2G & \ldots \quad (q-1)G \\
qG & (q+1)G & (q+2)G & \ldots \quad (2q-1)G \\
2qG & (2q+1)G & (2q+2)G & \ldots \quad (3q-1)G \\
\ldots & \ldots & \ldots & \\
((q-1)q)G & ((q-1)q+1)G & ((q-1)q+2)G & \ldots
\end{array}
$$

All n elements of the group here. The matrix has q columns and q rows.

Finding the integer a such that $A = aG$ is akin to finding where the given element A lives in this matrix.

Inutition for Shanks BSGS algorithm

We don't want to compute every element of the matrix to find A.

Geometrically we see there must be an intersection of the column containing G and the row containing A.

Shanks' Baby-Step-Giant-Step algorithm (BSGS): Compute the elements in the column containing G (giant steps) and the elements in the row containing A (baby steps), along with the the corresponding 'offsets'. (About $2q$ elements to compute and store.)

Find the intersection: at this point the offsets gives us a relation between G and A and allows us to find the integer a such that $A = aG$.

Algorithm inputs and assumptions

The inputs to our algorithm are as follows:

- a generator G of the group $<G>$
- the group order n, which must be prime
- integers $u, v > 0$ such that $n = uv + 1$
- a group element A (the element whose logarithm we seek)
- a function SDHP$_A()$

Our algorithm assumes that

- each group element is represented by a unique binary string
- we are given an efficient way to add/subtract arbitrary group elements (via an oracle, for example)

The input function SDHP$_A()$

The algorithm requires as input a function

$$\begin{aligned} \text{SDHP}_A() : <G> &\rightarrow <G> \\ T &\rightarrow aT \end{aligned}$$

We do not need details about the function implementation. We simply evaluate the function at certain inputs.

In terms of public key cryptography, this function performs the basic private key operation corresponding to the public key A: It scalar-multiplies an arbitrary input point T by the private key a.

The input function SDHP$_A()$

Recall the Diffie-Hellman function in $<G>$: Given arbitrary group elements $B = bG, C = cG$,

$$\begin{aligned} \mathrm{DH} : <G> \times <G> &\rightarrow <G> \\ (B, C) &\rightarrow bcG \end{aligned}$$

The SDHP$_A$ function solves the Diffie-Hellman problem when one of the inputs is fixed to be the point A. i.e.

$$\mathrm{SDHP}_A(T) = \mathrm{DH}(A, T)$$

i.e., It is the Diffie-Hellman function on a (very) restricted set of inputs.

Outline of algorithm

Recall the order of G is a prime $n = uv + 1$.
Suppose w generates F_n^*.

Imagine the group elements arranged as follows:

$$
\begin{array}{cccccc}
0G & & & & & \\
 & w^0G & w^1G & w^2G & \ldots & w^{u-1}G \\
 & w^uG & w^{u+1}G & & \ldots & w^{2u-1}G \\
 & w^{2u}G & w^{2u+1}G & & \ldots & w^{3u-1}G \\
 & \vdots & \ddots & \vdots & \ddots & \\
 & w^{(v-1)u}G & w^{(v-1)u+1}G & \ldots & & w^{vu-1}G
\end{array}
$$

This matrix is u by v and NOT necessarily $\sqrt{n} \times \sqrt{n}$ (like the earlier Shanks BSGS matrix.)

Basic idea of our algorithm

The function SDHP$_A()$ mapping $T \to aT$ allows us to compute a *known* multiple A' of the element A that is (somewhere) in the same column as G. (The first column.)

We will then apply a baby-step giant-step algorithm to the elements of *the first column* which will lead to the logarithm.

Jump to the first column...

The group element $A = aG$ is somewhere in this matrix. Assume $a = w^d$ for some d. We focus on finding d.

Evaluate the SDHP$_A$ function at $A = w^d G$. We get back the group element $aA = w^d w^d G = w^{2d} G$.

Evaluate the function at the element $w^{2d} G$ to obtain the element $w^{3d} G$. Continue in this manner.

After calling the function u times, we get the element $A' = w^{ud} G$.

This element is in the first column of our matrix.

Do BSGS variant in first column

We have $A' = w^{ud}G$ in the first column of the matrix. The element $G = w^0 G$ is also in the first column of the matrix.

Because the first column is closed under multiplication by powers of w^u, we can do a baby-step-giant-step algorithm *in the first column*:

Set of baby steps $\{w^{iu}G, i\}$. ($i = 0 \ldots \sqrt{v}$.)
Set of giant steps $\{w^{-j\sqrt{v}u}A', j\}$. ($j = 0 \ldots \sqrt{v}$.)

This finds the integer α such that $A' = w^{u\alpha}G$.

Thus: $A' = w^{ud}G = w^{u\alpha}G$, and $w^{uv} = 1$
Thus: $d \equiv \alpha \bmod v$.

This takes about $2\sqrt{v}$ steps so far.

Using $\alpha = d \bmod v$

Because $d \equiv \alpha \bmod v$, the group element $w^{-\alpha}A = w^{-\alpha}w^{d}G$ is in the set

$$\{w^{iv}G | i = 0 \ldots u-1\}$$

This set is closed under multiplication by powers of w^v, so again we can do a baby-step-giant-step algorithm in this set to find an integer β such that $w^{\beta v}G = w^{-\alpha}A$.

Thus: $w^{-\alpha}A = w^{\beta v}G$ so $A = w^{\alpha+\beta v}G$.
Thus: As $A = w^{d}G$ we have $d = \alpha + \beta v$ and so the desired logarithm $a = w^{\alpha+\beta v} \bmod n$.

This second BSGS step takes about $2\sqrt{u}$ steps.

Result

Suppose that G generates a group of prime order $n = uv + 1$, and that in addition to the element A we are given a SDHP$_A(\cdot)$ function.

Then there is an algorithm for finding an integer a such that $A = aG$ that requires u calls to the SDHP$_A$ function and at most $2(\sqrt{u} + \sqrt{v})$ group scalar multiplies and table lookups.

Least work case with this algorithm

The algorithm works for any choice of positive integers u, v such that $n = uv + 1$. The algorithm requires u calls to the SDHP_A function and about $2(\sqrt{u} + \sqrt{v})$ scalar multiplications in the group $<G>$.

If $u \approx \sqrt[3]{n}$, then the logarithm a can be found with u calls to the SDHP function and off-line work of about $2\log_2(n)\sqrt[3]{n}$ operations in $<G>$.

Disregarding work factors that are logarithmic in the group order n, this algorithm requires $\sqrt[3]{n}$ steps to find a logarithm.

Notes on algorithm...

Notes:

- algorithm is basically a generic algorithm
- Pohlig-Hellman
- For $\sqrt[3]{n}$ steps, you need a factor of $n-1$ of size about $\sqrt[3]{n}$.
- For $\sqrt[3]{n}$ steps you must query the SDHP$_A$ oracle $\sqrt[3]{n}$ times, which may be very unrealistic in practice.
- However, the algorithm will be faster than a square root algorithm even with a logarithmic (in n) number of oracle queries.
- there is Pollard rho variant (removing need for large amounts of storage)

Algorithm relevance (as an attack)

If this algorithm is to be relevant as an attack on a protocol:

- need access to SDHP$_A()$ function from somewhere
- protocol must be discrete-log based with a group where finding logs takes longer than $\sqrt[3]{n}$ work (otherwhise this algorithm is no faster than existing methods)

Taking logarithms in (most) elliptic curve groups often assumed to take $\sqrt{n}$ work.

Protocols providing a SDHP$(\cdot)$

- Ford-Kaliski Key Retrieval
- Basic Elgamal encryption
- Chaum and van Antwerpen Undeniable Signatures

These protocols were originally described using the group F_p^*.
In this case index calculus algorithms will find logarithms faster that the algorithm presented here.

However no special features of F_p^* were used, and in one case it was stated that 'the elliptic curve variant is obvious'.

Recent protocol providing a SDHP$(\cdot)$

Ford-Kaliski Key Retrieval: is currently undergoing standardization.
This protocol is based on the discrete logarithm problem and draft versions of the standard allow for elliptic curve variants.

In this protocol, a user enters password π into a trusted client, which contacts a trusted server. A long term secret a is stored on the server on behalf of the client.

Ford-Kaliski protocol flow outline

Client		Server
$b \in_R [1, n-1]$		
$c = g_\pi^b$	$\xrightarrow{c}$	$s = c^a$
$z = s^{b^{-1} \bmod n}$	$\xleftarrow{s}$	

Value $z = g_\pi^a$ is called a hardened password. It is a function of the (weakly protected) secret π and a (strongly protected) secret a.

The protocol helps guard against dictionary attacks on the password.

Attacking this protocol

The server provides an SDHP$_A$ oracle to the client.

An adversary could use the server as an SDHP oracle to recover the (long term) private key a using the algorithm we describe.

For example if doing this protocol with elliptic curve groups this might lead to a $\sqrt[3]{n}$ attack to find a, which might be less work than you would otherwise expect.

The draft standards were specifically modified to account for this attack presented here.

Elgamal provides a SDHP() oracle

Basic Elgamal encryption is already known to have problems.

Nonetheless...
Basic Elgamal encryption also provides an SDHP oracle.

Elgamal Encryption

Basic ElGamal Encryption: Alice chooses a private key a and computes her public key as $A = aG$.

Bob picks random r and encrypts message $m \in <G>$ to Alice:

$$(B, C) = (rG, rA + m)$$

Alice decrypts ciphertext (B, C) to plaintext

$$m = C - aB$$

Chosen ciphertext as SDHP oracle

Chosen Ciphertext Attack: Send any (B, C) to Alice, and she reveals the decryption m.

This provides an SDHP oracle: To get Alice to compute aT for arbitrary $T \in <G>$, send Alice:

$$(T, R),$$

where R can be chosen arbitrarily.

Alice returns the decrypted plaintext:

$$m = R - aT.$$

Eve easily calculates $aT = R - m$.

Chosen ciphertext on Elgamal

Thus if (repeated) chosen ciphertext attacks are possible against Alice, then she provides a SDHP$_A$ oracle so the earlier algorithm can be applied.

Basic ElGamal encryption is already known to be vulnerable to chosen ciphertext attacks in the sense that Eve can use a chosen ciphertext query to decrypt any other past ciphertext.

This attack uses Alice to find a, with which all ciphertexts (past and future) can be deciphered off-line.

For example if doing basic Elgamal with elliptic curve groups this might lead to a $\sqrt[3]{n}$ attack to find a, which might be less work than you would otherwise expect.

Another interpretation of the algorithm

For some groups of size n, the algorithm finds the discrete logarithm of A in considerably less than $\sqrt{n}$ time.

This is a reduction from finding the discrete logarithm of A to computing the SDHP$_A()$ function. (NOT a polynomial reduction)

This reduction can be interpreted as: If you assume finding the discrete logarithm of A requires at least $\sqrt{n}$ work then computing the SDHP$_A()$ function requires almost $\sqrt{n}$ work.

Applying this to Elliptic Curves

Because each SDHP$_A$ function is the Diffie-Hellman function with one input restricted to A, this is a statement that if the discrete logarithm problem 'is hard' then the Diffie-Hellman problem 'is hard'.

For example if $<G>$ is an elliptic curve group of prime order $n = uv + 1$ with $u \approx \sqrt[3]{n}$, and we assume no algorithm can solve the discrete logarithm problem in less than $\sqrt{n}$ steps, then this is saying...

In the elliptic curve group G, solving the Diffie-Hellman problem takes about $\sqrt{n}$ steps.

DLP and DHP connections

Several works reduce the discrete logarithm problem to the Diffie-Hellman problem.

- den Boer ('88)
- Mauer-Wolf ('96)
- Boneh-Lipton ('96)
- Muzereau, Smart, Vercauteren ('04)

These are all polynomial or subexponential reductions and are generally much stronger statements than that given by the reduction outlined here.

DLP and DHP connections

Although our reduction is weaker, it is a little different.

Other works reduce the problem of finding an arbitrary discrete log to the problem of computing the (two-input) Diffie-Hellman function.

Our reduction is from computing the logarithm of a single element A to computing the Diffie-Hellman function on a restricted domain dependent on A, namely SDHP_A.

Bounds

The reduction can be used to get explicit bounds. For example

If finding the logarithm a of A costs at least $\sqrt{n}$ operations in $<G>$, and a scalar multiplication in $<G>$ costs C, and $n = uv + 1$ with $u \approx 9C^2$, then computing the SDHP$_A()$ costs at least

$$\frac{\sqrt{n}}{27C^2}.$$

operations in $<G>$.

This is a reasonably tight reduction for elliptic curves.

Incidentally, the bound is similar to those calculated in the Muzereau, Smart and Vercauteran paper.

Possible to do better?

We have shown that if in addition to the standard inputs to a generic algorithm for solving discrete logarithms, one is given an additional input $\mathrm{SDHP}_A()$, then one can sometimes compute the logarithm of A in about $\sqrt[3]{n}$ work. Perhaps it is possible to do even better than this?

Possible to do better?

The paper 'Short Signatures Without Random Oracles' (Boneh, Boyen) defines the q-*Strong Diffie-Hellman Problem* .

Given an isomorphism ψ from $<G_1>$ to $<G_2>$, with $\psi(G_2) = G_1$. Groups have prime order n.
The problem is (additive notation):
Given a $(q+2)$-tuple $(G_1, G_2, aG_2, a^2G_2, a^3G_2, \ldots a^qG_2)$ as input, output a pair $(c, \frac{1}{a+c}G_1)$ for some $c \in \mathbb{Z}_n^*$.

Possible to do better?

In that paper they prove that a generic algorithm for solving this problem must take at least $\sqrt[3]{n}$ steps.

A generic algorithm given $aG_2, a^2G_2, a^3G_2, \ldots a^qG_2$ gets basically the same inputs as our algorithm.

Their proof shows that no generic algorithm, additionally given a SDHP$_A$ function, can find logarithms much faster than the one we describe.

Summary

- A generic-ish algorithm for finding discrete logarithms
- applicability of algorithm to attacking existing protocols/systems
- interpretations of the algorithm as a reduction