# CS259C, Final Paper: Discrete Log, CDH, and DDH

Deyan Simeonov

12/10/11

## 1 Introduction and Motivation

In this paper we will present an overview of the relations between the Discrete Logarithm (DL), Computational Diffie-Hellman (CDH) and Decision Diffie-Hellman (DDH) problems. It will be organized as follows. In section one, by looking at some applications, we will provide some insight on why we might be interested in looking at such relationships. In sections two and three we will make precise definitions and look at some immediate consequences. Section four is devoted to self-reducibility and self-correctness. Section five looks at current results in specific families of problem instances.

Loosely speaking, the DL problem asks, given a description of a group $G$ (unless otherwise noted, all groups will be abelian cyclic groups under addition and of prime order and by 'description' we mean identity, an (efficient) algorithm for performing subtraction and a generator $g$) and elements $h_1, h_2 \in G$ to find the unique integer $a \in \mathbb{Z}_p$ s.t. $h_2 = ah_1$. Except in some trivial cases - for instance if $G = \mathbb{Z}_p$, where we can use simple division using Bezout's Lemma - this problem is believed to be hard (in fact, in generic groups it is known to require at least $\Omega(\sqrt{p})$ group operations). Using the hardness of DL as a motivation, one can come up with various simple security protocols. A first example is the Diffie-Hellman key exchange: $A_1$ and $A_2$, both of which have the description of a group $G$, want to share a secret. This is done by $A_i$ picking a random $r_i \in \mathbb{Z}_p$ and sending $P_i = r_i g$ to the other party. Now $A_i$ can compute $r_i P_{1-i} = r_0 r_1 g$. Since an outside adversary sees $g, r_0 g, r_1 g$, it is reasonable to try to show that given this information only, it is hard to deduce $r_0 r_1 g$. Another example is ElGamal encryption: $A$ wants to be able to receive encrypted messages, so he picks a $G$, a random $s \in \mathbb{Z}_p$ and publishes $(G, h = sg)$. Now to send a message $m \in G$ to $A$, $B$ picks a random $r \in \mathbb{Z}_p$ and outputs $(c_1, c_2) = (rg, rh + m)$. $A$ can clearly compute $m = c_2 - sc_1$ and we want to believe that no one else can compute it (or, as a stronger assumption, distinguish it from any random message).

Note that while DL is obviously sufficient to solve the above, it is not obvious at all (and mostly unknown) whether it is necessary. For instance, in the DH key exchange (and similarly in ElGamal), it seems reasonable to ask the question of whether one can find $r_0 r_1 g$ without having to find $r_0, r_1$ first. This is called the CDH assumption. Similarly, one could ask the (intuitively) stronger question of whether one can distinguish $r_0 r_1 g$ from a random element of the group $G$ - which is a very reasonable thing to ask, since it implies indistinguishability of an encryption of $m$ and any other encryption. This gives rise to the DDH assumption. It is obvious that if one can solve DL, then one can solve CDH and that if one can solve CDH then he can solve DDH. Thus, in the sense that in order to (intuitively) reduce the space of possible ways for breaking a protocol we would want to have a nice hierarchical structure of 'hard' problems, it is tempting to try to prove (or disprove) the converses of these.

## 2 Definitions

This section is mainly based on [1] and [2]. Now let's make precise the above problems and notion of 'hardness'. Since we'll be looking at solvers, it makes sense to define what does it mean to solve a problem.

**Definition 2.1.** Consider a family of problems $\cup_k \mathcal{F}_k$ and an instance generator $B$ with security parameter $k$. An algorithm $A$ is said to solve a problem in $\mathcal{F}_k$ with noticeable probability if the probability that $A$ returns the correct answer is at least $\epsilon(k)$, where $\epsilon$ is s.t. $\exists m \in \mathbb{Z} : x^m > 1/\epsilon(x)$ for large $x$. The probability is taken over the randomness in $B$ and in $A$.

**Definition 2.2.** Consider a family of problems $\cup_k \mathcal{F}_k$ and an instance generator $B$ with security parameter $k$. An algorithm $A$ is said to solve a problem in $\mathcal{F}_k$ with overwhelming probability if the probability that $A$ returns the correct answer is at least $1 - \epsilon(k)$, where $\epsilon$ is s.t. $\forall m \in \mathbb{Z} : x^m < 1/\epsilon(x)$ for large $x$. The probability is taken over the randomness in $B$ and in $A$.

Given this, we can define what it means for an algorithm to solve $DL$ and $CDH$.

**Definition 2.3.** An algorithm $A$ is said to solve the discrete logarithm problem (DL) in the group $G$ with noticeable (overwhelming) probability if, given elements $h_1, h_2 \in G$, $A$ outputs $a \in \mathbb{Z}_p$ s.t. $h_2 = ah_1$ with noticeable (overwhelming) probability. Here the probability is taken over the randomness of the choice of $h_1, h_2 \in G$ and the randomness in $A$.

**Definition 2.4.** An algorithm $A$ is said to solve the computational Diffie-Hellman problem (CDH) in the group $G$ with noticeable (overwhelming) probability if, given elements $h_0, h_1 = a_1 h_0, h_2 = a_2 h_0 \in G$, $A$ outputs $h \in G$ s.t. $h = a_1 a_2 h_0$ with noticeable (overwhelming) probability. Here the probability is taken over the randomness of the choice of $h_0, h_1, h_2 \in G$ $(h_0 \neq 0)$ and the randomness in $A$.

For $DDH$ we need to be a little bit more careful:

**Definition 2.5.** A pair of probability distributions $\mathcal{P}_1, \mathcal{P}_2$ over some space $S$ is said to be computationally indistinguishable if, for any efficient (randomized) algorithm $A$ that has black-box access to these distributions we have $|P(A = true|\mathcal{P}_1) - P(A = true|\mathcal{P}_2)|$ is negligible.

In other words, no efficient algorithm can behave 'much' differently under the two distributions. With this in mind, we can define:

**Definition 2.6.** An algorithm $A$ is said to solve the decision Diffie-Hellman problem (DDH) in the group $G$ with noticeable probability if $A$ can distinguish $\{(h, a_1 h, a_2 h, a_1 a_2 h); h \leftarrow^R G, a_1, a_2 \leftarrow^R \mathbb{Z}_p\}$ and $\{(h_0, h_1, h_2, h_3); h_0, h_1, h_2, h_3 \leftarrow^R G\}$. Here both distributions are uniform within their support.

## 3 Trivial reductions

A natural question is to what extent does the exact definition of our probability space affect the performance of our algorithms. More precisely, one might wonder what happens if we restrict the set of problem instances to some (much smaller) set. For instance, one could define 'fixed' versions of the above problems. The fixed-DL problem asks, given $h$ to output $a$, s.t. $h = ag$. The fixed-CDH and fixed-DDH problems just fix $h = g$. This, of course, restricts the dimension of our input space. Clearly, one could use a solver for a non-fixed version to solve a fixed version of the same problem (here we are talking about exact solutions; there is an issue with the probabilistic analysis, which will be resolved by self-reducibility, shown in the next section). We can also ask the reverse questions.

**Lemma 3.1.** *Suppose we have an algorithm $A$ that can solve fixed-DL, i.e. $A(h) = a$, s.t. $h = ag$. Can we construct an algorithm $B$ s.t. $B(h_1, h_2) = b$ with $h_2 = bh_1$?*

*Proof.* This is simple. Call $A(h_1) = a_1, A(h_2) = a_2$. Then we know that $h_1 = a_1 g, h_2 = a_2 g \Rightarrow h_2 = a_2/a_1 h_1$, so $B$ can just output $a_2/a_1$. Computing inverses in $\mathbb{Z}_p$ is easy (it is a finite cyclic group of known order), so we are done. □

A more careful analysis can also show the equivalence of CDH and fixed-CDH.

**Lemma 3.2.** *Suppose we have an algorithm $A$ that can solve fixed-CDH, i.e. on input $h_1 = a_1 g, h_2 = a_2 g$ we have $A(h_1, h_2) = h$, s.t. $h = a_1 a_2 g$. Can we construct an algorithm $B$ s.t. on input $h_0, h_1 = a_1 h_0, h_2 = a_2 h_0$ we have $B(h_0, h_1, h_2) = a_1 a_2 h_0$?*

*Proof.* This is proved in [1] in a very detailed way. We will sketch the proof from there. Suppose we can solve the square-CDH problem, which given $h, ah$ outputs $a^2h$ for $h \in G, a \in \mathbb{Z}_p$. Then we can compute $h_1' = a_1^2 h_0$, $h_2' = a_2^2 h_0$ and $h_s = (a_1 + a_2)^2 h_0$. Then our answer is $\frac{1}{2}(h_s - h_1' - h_2')$. So we are left with the square-DH problem. Clearly, we can use our oracle $A$ to solve the fixed-inverse-DH problem, which given $ag$ outputs $a^{-1}g$ for any $a \in \mathbb{Z}_p$ (just remember that $a^{-1} = a^{p-2}$ and simulate the standard exponentiation algorithm using the oracle). But this means that, if $h_0 = a_0 g$, we can compute $h_a = \frac{1}{a_0 a_1 + a_0} g$ and $hb = \frac{1}{a_0 a_1 - a_0} g$ (just call the fixed-inverse-DH on $h_1 + h_0$ and $h_1 - h_0$). But then $h_b - h_a = \frac{2}{a_0(a_1^2 - 1)} g$, from which we can get $h_f = \frac{a_0(a_1^2-1)}{2} g$. Now our answer is just $2hf + h_0$, which solves the square-CDH problem for $h_0, h_1$. Since $h_0$ and $h_1$ are arbitrary, we are done. Combining the above, we get that there exists an efficient algorithm for CDH that uses an algorithm for fixed-CDH as a primitive. It is not hard to see that the number of oracle calls is $O(\log p)$, and the number of group operations is constant. $\square$

Even though it is tempting to believe a similar thing for DDH, this has not yet been resolved (see [3]).

We can think of these types of results as things that smoothen the success probability of our algorithms given a particular running time. For instance, in the above we have seen that for DL and CDH, the fixed versions are (roughly) not easier than the non-fixed versions. Naturally we want to address the other direction as well - namely, is it the case that any specific problem instance is not much harder than an average problem instance.

## 4 Self-reducibility and self-correctness

Now, following [2], we make the notion of self-reducibility precise:

**Definition 4.1.** Let $P$ be a computational problem which takes some input $x \in S$, where $S$ is the space of inputs. Then $P$ is random self-reducible if there is a polynomial-time algorithm that transforms an instance of $P$ with input $x$ to an instance with input $r \xleftarrow{R} S$ s.t. the answer to the original instance can be derived in polynomial time from the answer of our newly generated instance.

A simple example of the above is showing that DL is random self-reducible. Suppose we have some instance $(h_0, h_1 = ah_0)$. Then the reduction would be just picking $r_0, r_1 \xleftarrow{R} \mathbb{Z}_p$ (in reality we would have $r_0 \neq 0$, but this is a technicality) and generating $(h_0' = r_0 h_0, h_1' = r_0 r_1 h_1)$. Trivially, if $a'$ is the answer to the second problem, we have $a' = r_1 a$. Also, since $r_0$ and $r_1$ are chosen uniformly at random, now $h_0'$ and $h_1'$ are uniformly random element of $G$, so we have ended up with a random instance of DL.

Similarly one can, given an instance $(h, ah, bh)$ of CDH, define the instance $(r_0 h, (a + r_1) r_0 h, (b + r_2) r_0 h)$ for random $r_0, r_1, r_2 \in \mathbb{Z}_p$, which is now a uniformly random instance and whose answer can be used to find the answer to the original instance. The same holds for DDH, although the construction (and proof that the distribution becomes what we want it to be) is a bit more tedious ([1], [3]): we map $(h, ah, bh, ch)$ to $(r_0 h, (a + r_1) r_0 h, (br_2 + r_3) r_0 h, (cr_2 + ar_3 + br_1 r_2 + r_1 r_3) r_0 h)$ (note that we can indeed compute this without knowing $a, b, c$).

Now, armed with the power of reducing our problem instance to random problem instances, we can turn to the question of whether an algorithm with noticeable success probability implies one with overwhelming success probability - in other words, does there exist a **self-corrector** for that problem. Note that if we have an efficient way to check correctness of our answer, then this is obvious. For instance, given an algorithm $A$ for DL that succeeds with probability $\epsilon$, we can define an algorithm $B$ that at most $k$ times calls $A$ on a random self-reduction of its input and breaks if at any point it has the correct answer (correctness we can check just by using our group operation). This makes the success probability of $B$ equal to $1 - (1 - \epsilon)^k$. Setting e.g. $k = c/\epsilon$ this becomes $1 - 1/e^c$. For the other two, however, the proof for similar results is not that obvious. We'll start with CDH (based on [1]).

**Theorem 4.1.** *Suppose we have an algorithm $A$ that solves CDH in $G$ ($|G| = p$) with probability at least $\epsilon > 1/log(p)$. Then for $\epsilon' > 1/p$ can construct an algorithm $B$ that solves CDH in $G$ with probability at least $1 - \epsilon' - log(2r)^2/(r\epsilon^2)$ and makes at most $2log(2/\epsilon')/\epsilon$ queries to $A$.*

*Proof.* Our strategy will be to use the random self-reduction and generate a list of instances that we will pass to $A$. The problem then remains how to distinguish the correct answers from the incorrect ones. The trick to address this issue is to generate a second instance of our problem such that the correct answers to both instances are related in some way we can check and that any incorrect answers will with high probability not obey the same relation. Thus, then we would just use the self-reduction to generate two lists of instances (one based on the original problem and one on our modified) that we pass to $A$ and then check the validity of the relation for all pairs of answers.

More specifically, let the input to our problem be $(h_0, h_1 = ah_0, h_2 = bh_0)$. Generate $n$ instances of our problem using the self-reduction and call $A$ on each of them. Call the list of answers $L = (Z_1, Z_2, ..., Z_n)$. Note that since $A$ has success probability $\epsilon$, the probability that $L$ contains no copy of the right answer is at most $(1-\epsilon)^n$. Now define the triple $(h_0, xh_0 + yh_1, h_2)$ where $x, y \leftarrow^R \mathbb{Z}_p$, generate $n$ instances of the new problem similarly as above and call the list of answers $L' = (Z'_1, Z'_2, ..., Z'_n)$. Again, the probability that $L'$ contains no copy of the right answer is at most $(1-\epsilon)^n$. Note that the correct answer to this new instance is $(x+ay)bh_0 = xh_2 + y(abh_0)$. Thus, by looping through all $i, j \in \{1, ..., n\}$ and checking whether $yZ_i + xh_2 = Z'_j$ we can detect the correct answer $Z_i$. Now, we already know that with probability $1 - 2(1-\epsilon)^n$ there is at least one of all the $n^2$ pairs (namely, the 'correct' pair) satisfies the above relation, so it just remains to bound the probability of having a pair of incorrect $(Z_i, Z'_j)$ s.t. $yZ_i + xh_2 = Z'_j \Rightarrow y(Z_i - abh_0) = Z'_j - (ay + x)bh_0$. Note that if any one of $Z_i, Z'_j$ is correct, this forces the other one to be correct (since then one of the sides becomes zero), so we can assume that both $Z_i$ and $Z'_j$ are incorrect. But this means that $Z_i - abh_0$ and $Z'_j - (ay + x)bh_0$ are non-zero, so for a fixed $ay + x$ there is exactly one $y$ such that the above relation holds. Since $(x, y) \leftarrow^R \mathbb{Z}_p^2$ is equivalent to $(ay + x, y) \leftarrow^R \mathbb{Z}_p^2$, this means that the probability for an incorrect pair to obey the relation is $1/p$. Thus, the probability over all incorrect pairs that any one of them obeys the relation is at most $n^2/p$. Combining all of the above, we have constructed an algorithm that works with probability $1 - 2(1-\epsilon)^n - n^2/p$. The exact bounds in the theorem statement we get by defining $n = log(2/\epsilon')/\epsilon$. □

For the sake of completeness, let's also look at a self-corrector for $DDH$ ([3]).

**Theorem 4.2.** *Suppose we have an algorithm A that solves DDH in G ($|G| = p$) with noticeable probability. Then we can construct an algorithm B that uses A as an oracle and solves DDH with overwhelming probability.*

*Proof.* Even though this may sound involved, it's actually quite natural, since we already know $|P(A(x) = true|x \leftarrow^R DDH) - P(A = true|x \leftarrow^R G^4)| > \epsilon$. For brevity, let $P(A(x) = true|x \leftarrow^R DDH) = p + a > p = P(A = true|x \leftarrow^R G^4)$ (we can always swap the output of $A$ to ensure this property holds); then $a > \epsilon$. Now it is reasonable to believe that the way we would boost the probability of $B$ is by calling $A$ many times and using the fact that the average of $n$ Bernoulli variables with probability $p$ converges in probability to $p$. Namely, call $A$ $n$ times on self-reductions of input $x$. Let $X$ be the random variable denoting the fraction of times we have 'true' in the list of outputs. Then generate a random $y \in G^4$, call $A$ $n$ times on self-reductions of $y$ and similarly let $Y$ be the random variable denoting the fraction of times we have 'true' in the list of outputs. Using Hoeffding's inequality, we get that $P(|Y - p| > t) < 2e^{-2t^2n}$. If $x$ is a DDH tuple, we have $P(|X - p - a| > t) < 2e^{-2t^2n}$, otherwise we have $P(|X - p| > t) < 2e^{-2t^2n}$. Combining these, we get that if $x$ is a DDH tuple, we have $P(X - Y < a - 2t) < P(Y - p > t) + P(X - p - a < -t) < 2e^{-2t^2n}$ and otherwise $P(X - Y > 2t) < P(Y - p < -t) + P(X - p > t) < 2e^{-2t^2n}$. Thus, if we make $B$ return 'true' iff $|X - Y| > a/2$ then $P(B = true|x \leftarrow^R G^4) < 2e^{-a^2n/2}$ and $P(B = false|x \leftarrow^R DDH) < 2e^{-a^2n/2}$. The number of calls to the oracle is $2n$. Our failure probability decreases exponentially as $n$ increases; in particular we can get an overwhelming probability by picking $n = p^\alpha$ for any $\alpha > 0$. □

## 5 Solution methods

Now, having a better grasp on the average case-worst case complexities of the problems, we can with more confidence look at some solution methods. The first obvious question is whether there is an algorithm that is independent of the structure of the group. Precisely, under Shoup's model for generic algorithms ([1]):

**Definition 5.1.** Suppose we have some injection $\sigma$ from a group $G$ to some large set $S$. Then a generic algorithm $A$ takes as input a tuple $(\sigma(g_1), ..., \sigma(g_n))$ and outputs a tuple $(r_1, ..., r_m, \sigma(h_1), ..., \sigma(h_k))$ with $r_1, ..., r_m \in \mathbb{Z}_p, g_1, ..., g_n, h_1, ..., h_k \in G$. In addition, $A$ has access for an oracle for the group operation: given $\sigma(x), \sigma(y)$ the oracle returns $\sigma(x - y)$.

Unfortunately, it can be shown (e.g. see [1]) that for both DL and CDH the success probability using $m$ queries to the oracle is at most $O(m^2/p)$, which means that in order to get a good success probability we must do at least $\Omega(\sqrt{p})$ queries, which is usually too big. In addition, the baby-step-giant-step method or the Pollard-rho method for computing discrete logarithms have expected running time $O(\sqrt{p})$.

Our goal now is to show that hardness of DL implies hardness of CDH, so we will try given an oracle for CDH to solve DL. From now on, we will assume that we are working in a group of prime order (if we are in a cyclic group of non-prime order, then we can mostly lift our group to a prime-order subgroup, perform the algorithms there and then use the Chinese Remainder Theorem to get the result; this is the Pohlig-Hellman method, details are in []).

The main idea is that we can set things up so that by performing group operations and oracle queries on elements of $G$, we can implicitly perform operations in a different group, which we can control better. For instance, if $g_1 = a_1 g, g_2 = a_2 g$, we can compute $(a_1 - a_2)g$ using the group operation and $(a_1 a_2)g$ using an oracle query. Extending this we can compute $r(a)g$ for any rational function of $a \in \mathbb{Z}_p$, so we are implicitly performing operations in $\mathbb{F}_p$. With this in mind, let's look at the den Boer reduction ([4], [1]):

**Theorem 5.1.** *Suppose $l$ is the largest prime factor of $p-1$ and let $A$ be an oracle for CDH in $G$. Then one can solve the DL in $G$ with $O(log(p)log(log(p)))$ oracle queries, $O(log(p)(\sqrt{l} + lop(p)))$ multiplications in $\mathbb{F}_p$ and $O(\sqrt{l}log^2(p))$ group operations in $G$.*

*Proof.* Suppose we have the $DLP$ with input $(h, h' = ah)$ in $G$. Let $p - 1 = \prod_{i=1}^{k} l_i^{\beta_i}$ be the factorization of $p-1$. We can find a primitive root mod $p$ in time $O(log(p)log(log(p)))$, using the method described in [2]: basically the idea is to generate elements $x$ from $\mathbb{Z}_p^*$ uniformly at random, lift $x$ to the subgroups with maximal prime-power order (i.e. $x_i = x^{(p-1)/l_i^{\beta_i}}$), check if $x_i$ generates this subgroup (i.e. if $x_i^{l_i^{\beta_i - 1}} \neq 1$) and multiply our current result by $x_i$ if the answer is 'yes'. That way at each step we potentially make our result include a generator of another subgroup with order $l_i^{\beta_i}$. Since randomness of $x$ in $G$ implies randomness of $x_i$ in the subgroup of order $l_i^{\beta_i}$, we get that for each $x$, and each $i$, the probability that we make our result include a generator of the subgroup of order $l_i^{\beta_i}$ is $\phi(l_i^{\beta_i})/l_i^{\beta_i} = (l_i - 1)/l_i > 1/2$, so the expected number of $x$ we have to choose is constant. Each set of $x_i$'s can be computed with $O(log(k)log(p)) = O(log(p)log(log(p)))$ group operations using a standard divide-and-conquer approach ([2]), so our running time for now is $O(log(p)log(log(p)))$.

Now that we have $\alpha$ - a primitive root mod $p$, we know that for some unique $c \in \mathbb{Z}_{p-1}$ we have $a = \alpha^c$. Our goal will be to find $c$. We will use the Pohlig-Hellman method. Suppose $c = \sum_{j=0}^{\beta_i} r_j l_i^j$, where $0 \leq r_j \leq l_i - 1$ for $j \neq \beta_i$. We will show how to compute $r_0$, the computation for the rest is essentially the same. We use our oracle to compute $a^{(p-1)/l_i} g = \alpha^{c(p-1)/l_i} g = \alpha^{r_0(p-1)/l_i} g$ and use group operations to compute $\alpha^{(p-1)/l_i} g$. Now in order to find $r_0$ we can just use BSGS: for $w = \sqrt{l_i}$ compute $\alpha^{(p-1)/l_i jw} g$ for all $0 \leq j \leq w$ and $\alpha^{(r_0 - j')(p-1)/l_i} g$ for all $0 \leq j' \leq w$ and intersect the lists. Computing the lists requires $O(wlog(p))$ group operations and $O(w + log(p))$ operations in $\mathbb{Z}_p^*$. Having found $r_0$, we can continue in a similar fashion to find the other $r_j$'s. Now we can easily compute $c \mod l_i^{\alpha_i}$ for all $i$ and do a CRT followed by $a = \alpha^c$ to get the final result.

As for the running time, summing over all primes we get that all BSGS steps together take $O(\sum_{i=0}^{k} \beta_i \sqrt{l_i} log(p))$ group operations which is bounded by $O(\sqrt{l}log^2(p))$ and $O(\sum_{i=0}^{k} \beta_i(\sqrt{l_i} + log(p)))$ operations in $\mathbb{F}_p$ which is bounded by $O(\sqrt{l}log(p) + log^2(p))$. We also need all elements of the form $a^{(p-1)/l_i^t} g$ for $0 \leq 1 \leq \beta_i$, which we can get in $O(log(p)log(log(p)))$ oracle queries and all elements of the form $\alpha^{(p-1)/l_i^t}$ which we get in $O(log(p)log(log(p)))$ operations in $\mathbb{F}_p$. Computing $a$ from the $r_j's$ can be done with $O(log(p))$ operations in $\mathbb{F}_p$. Summing the complexities of all steps we get the desired bounds. □

The above algorithm's trick is to translate the problem instance into one that lies in $\mathbb{Z}_p^*$ and make use of the smoothness (i.e. the largest prime factor) of $p-1$ to improve its running time. A natural question is whether there are other groups to which we can translate the problem to, preferably some that will give us algorithms that do not rely on such strict assumptions about $p$. Maurer [5] suggested the use of the group of points on an elliptic curve $E$: $y^2 = x^3 + Ax + B$ over $\mathbb{F}_p$. The reduction, which we will briefly sketch below, is essentially the same as den Boer's with some minor differences related to the specifics of the group.

Suppose we have an elliptic curve $E$ $y^2 = x^3 + Ax + B$ over $\mathbb{F}_p$ of smooth order. Let the input for our $DL$ problem be $h, h' = ah$. Pick a random integer $r \in \mathbb{Z}_p$ and let $a' = a + r$. Then with high (roughly 1/2) probability it is true that $a'^3 + Aa' + B$ is a quadratic residue $\mod p$, which means that there is some $b$ s.t. $Q = (a', b)$ is a point on the elliptic curve. We can find $bg$ by using a standard square-root method in $\mathbb{Z}_p$ and using the oracle for $CDH$ to do multiplications in $\mathbb{Z}_p^*$. Let $P = (c, d)$ be a generator of the elliptic curve group. Then clearly if we can find $k$ s.t. $Q = [k]P$ we will find $a'$ and thus $a$. The problem, however, is that we don't know $Q$ explicitly, we just know $(a'g, bg)$ - we will call this the implicit representation of $(a', b)$. But note that, as long as we can perform the group operation on $E$ just based on the implicit representations, we will be in the exact same situation as in the den Boer reduction - before the only thing we were using is the fact that we can multiply implicit representations in $\mathbb{Z}_p^*$. But, remembering that the elliptic curve group operation is a rational function of the points' coordinates and that using the $CDH$ oracle we can evaluate arbitrary rational functions implicitly, we get that indeed we can perform the group operation on $E$ using only implicit representations. The running time is similar as in the den Boer case, with the difference that we have an additional $\sqrt{l}$ factor in the number of oracle queries caused by the fact that this time we need to use the oracle every time when we are dealing with implicit representations (in particular, we cannot avoid calling the oracle during $BSGS$). A careful analysis of the running time is given in [1].

This time, the performance relies on the smoothness of $|E(\mathbb{F}_p)|$. Basically, the reduction shows that given an elliptic curve of smooth order and a $CDH$ oracle, one can solve $DL$ efficiently. It is believed that the Hasse interval contains at least one integer that is $log(p)^{O(1)}$-smooth, so indeed if one is given this particular curve he can solve $DL$ efficiently. However, the existence of such an integer does not give us much insight on what happens if we don't know a curve with that particular order, so in order to exploit this we might want to be able to generate such curves. One approach is to try random curves, therefore requiring estimates on the fraction of smooth numbers in the Hasse interval $[p - 2\sqrt{p}, p + 2\sqrt{p}]$. It is conjectured that Hasse intervals contain at least $4\sqrt{p}/L_r(1/2, c)$ integers that are $n^c$-smooth. If that is the case one can (by randomly picking a curve) and then using the Maurer reduction get a $L_r(1/2, c)$ reduction from $CDH$ to $DL$.

## References

[1] Steven Galbraith, *The Diffie-Hellman Problem.*
http://math.auckland.ac.nz/ sgal018/crypto-book/ch22.pdf

[2] Steven Galbraith, *Basic Algorithmic Number Theory.*
http://math.auckland.ac.nz/ sgal018/crypto-book/ch3.pdf

[3] Dan Boneh, *The Decision Diffie-Hellman Problem.*
http://crypto.stanford.edu/ dabo/pubs/papers/DDH.pdf

[4] Bert den Boer, *Diffie-Hellman is as Strong as Discrete Log for Certain Primes.*
Crypto 1988

[5] Ueli M. Maurer, *Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms.*
Crypto 1994