Chapter 21

The Diffie-Hellman Problem

This chapter gives a thorough discussion of the computational Diffie-Hellman problem (CDH) and related computational problems. We give a number of reductions between computational problems, most significantly reductions from DLP to CDH. We explain self-correction of CDH oracles, study the static Diffie-Hellman problem, and study hard bits of the DLP and CDH. We always use multiplicative notation for groups in this chapter (except for in the Maurer reduction where some operations are specific to elliptic curves).

21.1 Variants of the Diffie-Hellman Problem

We present some computational problems related to CDH, and prove reductions among them. The main result is to prove that CDH and Fixed-CDH are equivalent. Most of the results in this section apply to both algebraic groups (AG) and algebraic group quotients (AGQ) of prime order r (some exceptions are Lemma 21.1.9, Lemma 21.1.16 and, later, Lemma 21.3.1). For the algebraic group quotients G considered in this book then one can obtain all the results by lifting from the quotient to the covering group G' and applying the results there.

A subtle distinction is whether the base element $g \in G$ is considered fixed or variable in a CDH instance. To a cryptographer it is most natural to assume the generator is fixed, since that corresponds to the usage of cryptosystems in the real world (the group G and element $g \in G$ are fixed for all users). Hence, an adversary against a cryptosystem leads to an oracle for a fixed generator problem. To a computational number theorist it is most natural to assume the generator is variable, since algorithms in computational number theory usually apply to all problem instances. Hence both problems are studied in the literature and when an author writes CDH it is sometimes not explicit which of the variants is meant. Definition 20.2.1 was for the case when g varies. Definition 21.1.1 below is the case when g is fixed. This issue is discussed in Section 5 of Shoup [554] and in Sadeghi and Steiner [508] (where it is called "granularity").

Definition 21.1.1. Let G be an algebraic group (AG) or algebraic group quotient (AGQ) and let $g \in G$. The **Fixed-base computational Diffie-Hellman problem (Fixed-CDH)** with respect to g is: Given (g^a, g^b) to compute g^{ab}.

In this book the acronym CDH will always refer to the case where g is allowed to vary. Hence, an algorithm for CDH will always take three inputs (formally we should also include a description of the underlying group G, but we assume this is implicit in the specification of g) while an algorithm for Fixed-CDH will always take two inputs.

It is trivial that Fixed-CDH $\leq_R$ CDH, but the reverse implication is less obvious; see Corollary 21.1.18 below.

Analogously, given $g \in G$ one can define Fixed-DLP (namely, given h to find a such that $h = g^a$) and Fixed-DDH (given (g^a, g^b, g^c) determine whether $g^c = g^{ab}$). Though Fixed-DLP is equivalent to DLP (see Exercise 21.1.2) it is not expected that DDH is equivalent to Fixed-DDH (see Section 5.3.4 of [554]).

Exercise 21.1.2. Prove that Fixed-DLP is equivalent to DLP.

Exercise 21.1.3. Let G be a cyclic group of prime order r. Let $h_1, h_2, h_3 \in G$ such that $h_j \neq 1$ for $j = 1, 2, 3$. Show there exists some $g \in G$ such that (g, h_1, h_2, h_3) is a Diffie-Hellman tuple.

We now introduce some other variants of CDH. These are interesting in their own right, but are also discussed as they play a role in the proof of equivalence between CDH and Fixed-CDH.

Definition 21.1.4. Let G be a group or algebraic group quotient of prime order r. The computational problem **Inverse-DH** is: given a pair $g, g^a \in G - \{1\}$ of elements of prime order r in G to compute $g^{a^{-1} \pmod r}$. (Clearly, we must exclude the case $a = 0$ from the set of instances.)

Lemma 21.1.5. *Inverse-DH $\leq_R$ CDH.*

Proof: Suppose O is a perfect oracle for solving CDH. Let $(g, g_1 = g^a)$ be the given Inverse-DH instance. Then

$$g = g_1^{a^{-1}}.$$

Calling $O(g_1, g, g) = O(g_1, g_1^{a^{-1}}, g_1^{a^{-1}})$ gives $g_1^{a^{-2}}$. Finally,

$$g_1^{a^{-2}} = (g^a)^{a^{-2}} = g^{a^{-1}}$$

as required. □

Definition 21.1.6. Let G be an AG or AGQ. The computational problem **Square-DH** is: given (g, g^a) where $g \in G$ has prime order r to compute g^{a^2}.

Exercise 21.1.7. Show that Square-DH $\leq_R$ CDH.

Lemma 21.1.8. *Square-DH $\leq_R$ Inverse-DH.*

Proof: Let O be a perfect oracle that solves Inverse-DH and let $(g, g_1 = g^a)$ be given. If $g_1 = 1$ then return 1. Otherwise, we have

$$O(g_1, g) = O(g_1, g_1^{a^{-1}}) = g_1^a = (g^a)^a = g^{a^2}.$$

□

Hence Square-DH $\leq_R$ Inverse-DH $\leq_R$ CDH. Finally we show CDH $\leq_R$ Square-DH and so all these problems are equivalent.

Lemma 21.1.9. *Let G be a group of odd order. Then CDH $\le_R$ Square-DH.*

Proof: Let (g, g^a, g^b) be a CDH instance. Let O be a perfect oracle for Square-DH. Call $O(g, g^a)$ to get $g_1 = g^{a^2}$, $O(g, g^b)$ to get $g_2 = g^{b^2}$ and $O(g, g^a g^b)$ to get $g_3 = g^{a^2+2ab+b^2}$.

Now compute

$$(g_3/(g_1 g_2))^{2^{-1} \pmod r},$$

which is g^{ab} as required. □

Exercise 21.1.10. Let G be a group of prime order r. Show that Inverse-DH and Square-DH are random self-reducible. Hence give a self-corrector for Square-DH. Finally, show that Lemma 21.1.9 holds for non-perfect oracles. (Note that it seems to be hard to give a self-corrector for Inverse-DH directly, though one can do this via Lemma 21.1.8.)

Note that the proofs of Lemmas 21.1.5 and 21.1.8 require oracle queries where the first group element in the input is not g. Hence, these proofs do not apply to variants of these problems where g is fixed. We now define the analogous problems for fixed g and give reductions between them.

Definition 21.1.11. Let g have prime order r and let $G = \langle g \rangle$. The computational problem **Fixed-Inverse-DH** is: given $g^a \neq 1$ to compute $g^{a^{-1} \pmod r}$. Similarly, the computational problem **Fixed-Square-DH** is: given g^a to compute g^{a^2}.

Exercise 21.1.12. Show that Fixed-Inverse-DH and Fixed-Square-DH are random self-reducible.

Lemma 21.1.13. *Let $g \in G$. Let A be a perfect Fixed-CDH oracle. Let $h = g^a$ and let $n \in \mathbb{N}$. Then one can compute $g^{a^n \pmod r}$ using $\le 2\log_2(n)$ queries to A.*

Proof: Assume A is a perfect Fixed-CDH oracle. Define $h_i = g^{a^i \pmod r}$ so that $h_1 = h$. One has $h_{2i} = A(h_i, h_i)$ and $h_{i+1} = A(h_i, h)$. Hence one can compute h_n by performing the standard square-and-multiply algorithm for efficient exponentiation. □

Note that the number of oracle queries in Lemma 21.1.13 can be reduced by using window methods or addition chains.

Exercise 21.1.14. Show that if the conjecture of Stolarsky (see Section 2.8) is true then one can compute g^{a^n} in $\log_2(n) + \log_2(\log_2(n))$ Fixed-CDH oracle queries.

Lemma 21.1.15. *Fixed-Inverse-DH $\le_R$ Fixed-CDH.*

Proof: Fix $g \in G$. Let O be a perfect Fixed-CDH oracle. Let g^a be the given Fixed-Inverse-DH instance. Our task is to compute $g^{a^{-1}}$. The trick is to note that $a^{-1} = a^{r-2} \pmod r$. Hence, one computes $g^{a^{r-2}}$ using Lemma 21.1.13. The case of non-perfect oracles requires some care, although at least one can check the result using O since one should have $O(g^a, g^{a^{-1}}) = g$. □

Lemma 21.1.16. *Fixed-Square-DH $\le_R$ Fixed-Inverse-DH.*

Proof: Let $h = g^a$ be the input Fixed-Square-DH instance and let A be a perfect oracle for the Fixed-Inverse-DH problem. Call $A(gh)$ to get $g^{(1+a)^{-1}}$ and call $A(gh^{-1})$ to get $g^{(1-a)^{-1}}$.

Multiplying these outputs gives

$$w = g^{(1+a)^{-1}} g^{(1-a)^{-1}} = g^{2(1-a^2)^{-1}}.$$

Calling $A(w^{2^{-1} \pmod r})$ gives g^{1-a^2} from which we compute g^{a^2} as required. □

We can now solve a non-fixed problem using an oracle for a fixed problem.

Lemma 21.1.17. *Square-DH $\leq_R$ Fixed-CDH.*

Proof: Let $g \in G$ be fixed of prime order r and let A be a perfect Fixed-CDH oracle. Let g_1, g_1^b be the input Square-DH problem. Write $g_1 = g^a$. We are required to compute $g_1^{b^2} = g^{ab^2}$.

Call $A(g_1^b, g_1^b)$ to compute $g^{a^2b^2}$. Use the perfect Fixed-CDH oracle as in Lemma 21.1.15 to compute $g^{a^{-1}}$. Then compute $A(g^{a^2b^2}, g^{a^{-1}})$ to get g^{ab^2}. $\square$

Since CDH $\leq_R$ Square-DH we finally obtain the main result of this section.

Corollary 21.1.18. *Fixed-CDH and CDH are equivalent.*

Proof: We already showed Fixed-CDH $\leq_R$ CDH. Now, let A be a perfect Fixed-CDH oracle. Lemma 21.1.17 together with Lemma 21.1.9 gives CDH $\leq_R$ Square-DH $\leq_R$ Fixed-CDH as required.

Now suppose A only succeeds with noticeable probability $\epsilon > 1/\log(r)^c$ for some fixed c. The reductions CDH $\leq_R$ Square-DH $\leq_R$ Fixed-CDH require $O(\log(r))$ oracle queries. We perform self-correction (see Section 21.3) to obtain an oracle A for Fixed-CDH that is correct with probability $1 - 1/(\log(r)^{c'})$ for some constant c'; by Theorem 21.3.8 this requires $O(\log(r)^c \log\log(r))$ oracle queries. $\square$

Exercise 21.1.19. It was assumed throughout this section that G has prime order r. Suppose instead that G has order $r_1 r_2$ where r_1 and r_2 are distinct odd primes and that g is a generator for G.

Prove that if one has a perfect CDH oracle O_1 that applies in the subgroup of order r_1, and a perfect CDH oracle O_2 that applies in the subgroup of order r_2, then one can solve CDH in G.

More generally in this context, which of the results in this section no longer necessarily hold? Is Fixed-CDH in $\langle g \rangle$ equivalent to Fixed-CDH in $\langle g^{r_1} \rangle$?

We end with a variant of the DDH problem.

Exercise 21.1.20. Let g have prime order r and let $\{x_1, \dots, x_n\} \subset \mathbb{Z}/r\mathbb{Z}$. For a subset $A \subset \{1, \dots, n\}$ define

$$g^A = g^{\prod_{i \in A} x_i}.$$

The **group decision Diffie-Hellman problem** (GDDH) is: Given g, g^A for all proper subsets $A \subsetneq \{1, \dots, n\}$, and h, to distinguish $h = g^c$ (where $c \in \mathbb{Z}/r\mathbb{Z}$ is chosen uniformly at random) from $g^{x_1 x_2 \cdots x_n}$. Show that GDDH $\equiv$ DDH.

21.2 Lower Bound on the Complexity of CDH for Generic Algorithms

We have seen (Theorem 13.4.5) that a generic algorithm requires $\Omega(\sqrt{r})$ group operations to solve the DLP in a group of order r. Shoup proved an analogue of this result for CDH. As before, fix $t \in \mathbb{R}_{>0}$ and assume that all group elements are represented by bitstrings of length at most $t\log(r)$.

Theorem 21.2.1. *Let G be a cyclic group of prime order r. Let A be a generic algorithm for CDH in G that makes at most m oracle queries. Then the probability that $A(\sigma(g), \sigma(g^a), \sigma(g^b)) = \sigma(g^{ab})$ over $a, b \in \mathbb{Z}/r\mathbb{Z}$ and an encoding function $\sigma : G \to S \subseteq \{0,1\}^{\lceil t\log(r) \rceil}$ chosen uniformly at random is $O(m^2/r)$.*

Proof: The proof is almost identical to the proof of Theorem 13.4.5. Let $S = \{0,1\}^{\lceil t\log(r)\rceil}$. The simulator begins by uniformly choosing three distinct $\sigma_1, \sigma_2, \sigma_3$ in S and running $A(\sigma_1, \sigma_2, \sigma_3)$. The encoding function is then specifed at the two points $\sigma_1 = \sigma(g)$ and $\sigma_2 = \sigma(h)$. From the point of view of A, g and h are independent distinct elements of G.

It is necessary to ensure that the encodings are consistent with the group operations. This cannot be done perfectly without knowledge of a and b, but using polynomials as previously ensures there are no "trivial" inconsistencies. The simulator maintains a list of pairs (σ_i, F_i) where $\sigma_i \in S$ and $F_i \in \mathbb{F}_r[x,y]$ (indeed, the $F_i(x,y)$ will always be linear). The initial values are $(\sigma_1, 1)$, (σ_2, x) and (σ_3, y). Whenever A makes an oracle query on (σ_i, σ_j) the simulator computes $F = F_i - F_j$. If F appears as F_k in the list of pairs then the simulator replies with σ_k and does not change the list. Otherwise, an element $\sigma \in S$, distinct from the previously used values, is chosen uniformly at random, (σ, F) is added to the simulator's list, and σ is returned to A.

After making at most m oracle queries, A outputs $\sigma_4 \in \mathbb{Z}/r\mathbb{Z}$. The simulator now chooses a and b uniformly at random in $\mathbb{Z}/r\mathbb{Z}$. Algorithm A wins if $\sigma_4 = \sigma(g^{ab})$. Note that if σ_4 is not σ_1, σ_2 or one of the strings output by the oracle then the probability of success is at most $1/(2^{\lceil t\log(r)\rceil} - m - 2)$. Hence we assume that σ_4 is on the simulator's list.

Let the simulator's list contain precisely k polynomials $\{F_1(x,y), \dots, F_k(x,y)\}$ for some $k \le m+3$. Let E be the event that $F_i(a,b) = F_j(a,b)$ for some pair $1 \le i < j \le k$ or $F_i(a,b) = ab$. The probability that A wins is

$$\Pr(A \text{ wins } | E)\Pr(E) + \Pr(A \text{ wins } | \neg E)\Pr(\neg E). \tag{21.1}$$

For each pair $1 \le i < j \le k$ the probability that $(F_i - F_j)(a,b) = 0$ is $1/r$ by Lemma 13.4.4. Similarly, the probability that $F_i(a,b) - ab = 0$ is $2/r$. Hence, the probability of event E is at most $k(k+1)/2r + 2k/r = O(m^2/r)$. On the other hand, if event E does not occur then all A "knows" about (a,b) is that it lies in the set

$$\mathcal{X} = \{(a,b) \in (\mathbb{Z}/r\mathbb{Z})^2 : F_i(a,b) \ne F_j(a,b) \text{ for all } 1 \le i < j \le k \text{ and } F_i(a,b) \ne ab \text{ for all } 1 \le i \le k\}.$$

Let $N = \#\mathcal{X} \approx r^2 - m^2/2$ Then $\Pr(\neg E) = N/r^2$ and $\Pr(A \text{ wins } | \neg E) = 1/N$.

Hence, the probability that A wins is $O(m^2/r)$. □

21.3 Random Self-Reducibility and Self-Correction of CDH

We defined random self-reducibility in Section 2.1.4. Lemma 2.1.19 showed that the DLP in a group G of prime order r is random self-reducible. Lemma 2.1.20 showed how to obtain an algorithm with arbitrarily high success probability for the DLP from an algorithm with noticeable success probability.

Lemma 21.3.1. *Let g have order r and let $G = \langle g \rangle$. Then CDH in G is random self-reducible.*

Proof: Let $\mathcal{X} = (G - \{1\}) \times G^2$ Let $(g, h_1, h_2) = (g, g^a, g^b) \in \mathcal{X}$ be the CDH instance. Choose uniformly at random $1 \le u < r$ and $0 \le v, w < r$ and consider the triple $(g^u, h_1^u g^{uv}, h_2^u g^{uw}) = (g^u, (g^u)^{a+v}, (g^u)^{b+w}) \in \mathcal{X}$. Then every triple in $\mathcal{X}$ arises from exactly one triple (u,v,w). Hence, the new triples are uniformly distributed in $\mathcal{X}$. If $Z = (g^u)^{(a+v)(b+w)}$ is the solution to the new CDH instance then the solution to the original CDH instance is

$$Z^{u^{-1} \pmod r} h_1^{-w} h_2^{-v} g^{-vw}.$$

$\square$

Exercise 21.3.2. Show that Fixed-CDH is random self-reducible in a group of prime order r.

The following problem[1] is another cousin of the computational Diffie-Hellman problem. It arises in some cryptographic protocols.

Definition 21.3.3. Fix g of prime order r and $h = g^a$ for some $1 \le a < r$. The **static Diffie-Hellman problem** (**Static-DH**) is: Given $h_1 \in \langle g \rangle$ to compute h_1^a.

Exercise 21.3.4. Show that the static Diffie-Hellman problem is random self-reducible.

One can also consider the decision version of static Diffie-Hellman.

Definition 21.3.5. Fix g of prime order r and $h = g^a$ for some $1 \le a < r$. The **decision static Diffie-Hellman problem** (**DStatic-DH**) is: Given $h_1, h_2 \in \langle g \rangle$ to determine whether $h_2 = h_1^a$.

We now show that DStatic-DH is random-self-reducible. This is a useful preliminary to showing how to deal with DDH.

Lemma 21.3.6. *Fix g of prime order r and $h = g^a$ for some $1 \le a < r$. Then the decision static Diffie-Hellman problem is random self-reducible.*

Proof: Write $G = \langle g \rangle$. Choose $1 \le w < r$ and $0 \le x < r$ uniformly at random. Given (h_1, h_2) compute $(Z_1, Z_2) = (h_1^w g^x, h_2^w h^x)$. We must show that if (h_1, h_2) is (respectively, is not) a valid Static-DH pair then (Z_1, Z_2) is uniformly distributed over the set of all valid (resp. invalid) Static-DH pairs.

First we deal with the case of valid Static-DH pairs. It is easy to check that if $h_2 = h_1^a$ then $Z_2 = Z_1^a$. Furthermore, for any pair $Z_1, Z_2 \in G$ such that $Z_2 = Z_1^a$ then one can find exactly $(r-1)$ pairs (w, x) such that $Z_1 = h_1^w g^x$.

On the other hand, if $h_2 \ne h_1^a$ then write $h_1 = g^b$ and $h_2 = g^c$ with $c \not\equiv ab \pmod{r}$. For any pair $(Z_1, Z_2) = (g^y, g^z) \in G^2$ such that $z \not\equiv ay \pmod{r}$ we must show that (Z_1, Z_2) can arise from precisely one choice (w, x) above. Indeed,

$$\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} b & 1 \\ c & a \end{pmatrix} \begin{pmatrix} w \\ x \end{pmatrix}$$

and, since the matrix has determinant $ab - c \not\equiv 0 \pmod{r}$ one can show that there is a unique solution for (w, x) and that $w \not\equiv 0 \pmod{r}$. $\square$

We now tackle the general case of decision Diffie-Hellman.

Lemma 21.3.7. *Let g have prime order r and let $G = \langle g \rangle$. Then DDH in G is random self-reducible.*

Proof: Choose $1 \le u, w < r$ and $0 \le v, x < r$ uniformly at random. Given $(g, h_1, h_2, h_3) = (g, g^a, g^b, g^c)$ define the new tuple $(g^u, h_1^u g^{uv}, h_2^{uw} g^{ux}, h_3^{uw} h_1^{ux} h_2^{vw} g^{uvx})$. One can verify that the new tuple is a valid Diffie-Hellman tuple if and only if the original input is a valid Diffie-Hellman tuple (i.e., $c = ab$). If the original tuple is a valid Diffie-Hellman tuple then the new tuple is uniformly distributed among all Diffie-Hellman tuples. Finally, we show that if the original tuple is not a valid Diffie-Hellman tuple then the new tuple is uniformly distributed among the set of all invalid Diffie-Hellman tuples. To see this think

[1]The Static-DH problem seems to have been first studied by Brown and Gallant [111].

of (h_2, h_3) as a DStatic-DH instance with respect to the pair (g, h_1). Since $(g^u, h_1^u g^{uv})$ is chosen uniformly at random from $(G - \{1\}) \times G$ we have a uniformly random DStatic-DH instance with respect to a uniformly random static pair. The result then follows from Lemma 21.3.6. □

It is easy to turn a DLP oracle that succeeds with noticeable probability ϵ into one that succeeds with probability arbitrarily close to 1, since one can check whether a solution to the DLP is correct. It is less easy to amplify the success probability for a non-perfect CDH oracle.

A natural (but flawed) approach is just to run the CDH oracle on random self-reduced instances of CDH until the same value appears twice. We now explain why this approach will not work in general. Consider a Fixed-CDH oracle that, on input (g^a, g^b), returns $g^{ab+\xi}$ where $\xi \in \mathbb{Z}$ is uniformly chosen between $-1/\log(r)$ and $1/\log(r)$. Calling the oracle on instances arising from the random self-reduction of Exercise 21.3.2 one gets a sequence of values $g^{ab+\xi}$. Eventually the correct value g^{ab} will occur twice, but it is quite likely that some other value will occur twice before that time.

We present Shoup's self-corrector for CDH or Fixed-CDH from [553].[2] Also see Cash, Kiltz and Shoup [120].

Theorem 21.3.8. *Fix $l \in \mathbb{N}$. Let g have prime order r. Let A be a CDH (resp. Fixed-CDH) oracle with success probability at least $\epsilon > \log(r)^{-l}$. Let (g, g^a, g^b) be a CDH instance. Let $1 > \epsilon' > 1/r$. Then one can obtain an oracle that solves the CDH (resp. Fixed-CDH) with probability at least $1 - \epsilon' - \log(2r)^2/(r\epsilon^2)$ and that makes at most $2\lceil \log(2/\epsilon')/\epsilon \rceil$ queries to A (where* log *is the natural logarithm).*

Proof: Define $c = \log(2/\epsilon') \in \mathbb{R}$ so that $e^{-c} = \epsilon'/2$. First call the oracle $n = \lceil c/\epsilon \rceil$ times on random-self-reduced instances (if the oracle is a CDH oracle then use Lemma 21.3.1 and if the oracle is a Fixed-CDH oracle then use Exercise 21.3.2) of the input problem (g, g^a, g^b) and store the resulting guesses $Z_1, \dots, Z_n$ for g^{ab} in a list L_1. Note that $n = O(\log(r)^{l+1})$. The probability that L_1 contains at least one copy of g^{ab} is $\geq 1 - (1-\epsilon)^{c/\epsilon} \geq 1 - e^{-c} = 1 - \epsilon'/2$.

Now choose uniformly at random integers $1 \leq s_1, s_2 < r$ and define $X_2 = g^{s_1}/(g^a)^{s_2}$. One can show that X_2 is uniformly distributed in $G = \langle g \rangle$ and is independent of $X_1 = g^a$.

Call the oracle another n times on random-self-reduced versions of the CDH instance (g, X_2, g^b) and store the results $Z'_1, \dots, Z'_n$ in a list L_2.

Hence, with probability $\geq (1 - \epsilon'/2)^2 \geq 1 - \epsilon'$ there is some $Z_i \in L_1$ and some $Z'_j \in L_2$ such that $Z_i = g^{ab}$ and $Z'_j = g^{b(s_1 - as_2)}$. For each $1 \leq i, j \leq n$ test whether

$$Z_i^{s_2} = (g^b)^{s_1}/Z'_j. \tag{21.2}$$

If there is a unique solution (Z_i, Z'_j) then output Z_i, otherwise output $\perp$. Finding Z_i can be done efficiently by sorting L_1 and then, for each $Z'_j \in L_2$, checking whether the value of the right hand side of equation (21.2) lies in L_1.

We now analyse the probability that the algorithm fails. The probability there is no pair (Z_i, Z'_j) satisfying equation (21.2), or that there are such pairs but none of them have $Z_i = g^{ab}$, is at most ϵ'. Hence, we now assume that a good pair (Z_i, Z'_j) exists and we want to bound the probability that there is a bad pair (i.e., a solution to equation (21.2) for which $Z_i \neq g^{ab}$). Write $X_1 = g^a$, $X_2 = g^{a'}$ (where $a' = s_1 - as_2$) and $Y = g^b$. Suppose (Z, Z') is a pair such that

$$Z^{s_2} Z' = Y^{s_1}. \tag{21.3}$$

[2]Maurer and Wolf [405] were the first to give a self-corrector for CDH, but Shoup's method is more efficient.

We claim that $Z = Y^a$ and $Z' = Y^{a'}$ with probability at least $1 - 1/q$. Note that if equation (21.3) holds then

$$(Z/Y^a)^{s_1} = Y^{a'}/Z'. \tag{21.4}$$

If precisely one of $Z = Y^a$ or $Z' = Y^{a'}$ holds then this equation does not hold. Hence, $Z \neq Y^a$ and $Z' \neq Y^{a'}$, in which case there is precisely one value for s_1 for which equation (21.4) holds. Considering all n^2 pairs $(Z, Z') \in L_1 \times L_2$ it follows there are at most n^2 values for s_1, which would lead to an incorrect output for the self-corrector. Since s_1 is chosen uniformly at random the probability of an incorrect output is at most n^2/r. Since $n \leq \log(2r)/\epsilon$ one gets the result. Note that $\log(2r)^2/(r\epsilon^2) = O(\log(r)^{2+2l}/r)$. $\square$

Exercise 21.3.9. Extend Lemma 21.1.13 to the case of a non-perfect Fixed-CDH oracle. What is the number of oracle queries required?

21.4 The den Boer and Maurer Reductions

The goal of this section is to discuss reductions from DLP to CDH or Fixed-CDH in groups of prime order r. Despite having proved that Fixed-CDH and CDH are equivalent, we prefer to treat them separately in this section. The first such reduction (assuming a perfect Fixed-CDH oracle) was given by den Boer [169] in 1988. Essentially den Boer's method involves solving a DLP in $\mathbb{F}_r^*$, and so it requires $r - 1$ to be sufficiently smooth. Hence there is no hope of this approach giving an equivalence between Fixed-CDH and DLP for all groups of prime order.

The idea was generalised by Maurer [402] in 1994, by replacing the multiplicative group $\mathbb{F}_r^*$ by an elliptic curve group $E(\mathbb{F}_r)$. Maurer and Wolf [405, 406, 408] extended the result to non-perfect oracles. If $\#E(\mathbb{F}_r)$ is sufficiently smooth then the reduction is efficient. Unfortunately, there is no known algorithm to efficiently generate such smooth elliptic curves. Hence Maurer's result also does not prove equivalence between Fixed-CDH and DLP for all groups. A subexponential-time reduction that conjecturally applies to all groups was given by Boneh and Lipton [83]. An exponential-time reduction (but still faster than known algorithms to solve DLP) that applies to all groups was given by Muzereau, Smart and Vercauteren [448], and Bentahar [42, 43].

21.4.1 Implicit Representations

Definition 21.4.1. Let G be a group and let $g \in G$ have prime order r. For $a \in \mathbb{Z}/r\mathbb{Z}$ we call $h = g^a$ an **implicit representation** of a.

In this section we call the usual representation of $a \in \mathbb{Z}/r\mathbb{Z}$ the **explicit representation** of a.

Lemma 21.4.2. *There is an efficient (i.e., computable in polynomial-time) mapping from $\mathbb{Z}/r\mathbb{Z}$ to the implicit representations of $\mathbb{Z}/r\mathbb{Z}$. One can test equality of elements in $\mathbb{Z}/r\mathbb{Z}$ given in implicit representation. If h_1 is an implicit representation of a and h_2 is an implicit representation of b then $h_1 h_2$ is an implicit representation of $a + b$ and h_1^{-1} is an implicit representation of $-a$.*

In other words, we can compute in the additive group $\mathbb{Z}/r\mathbb{Z}$ using implicit representations.

Lemma 21.4.3. *If h is an implicit representation of a and $b \in \mathbb{Z}/r\mathbb{Z}$ is known explicitly, then h^b is an implicit representation of ab.*

Let O be a perfect Fixed-CDH oracle with respect to g. Suppose h_1 is an implicit representation of a and h_2 is an implicit representation of b. Then $h = O(h_1, h_2)$ is an implicit representation of ab.

In other words, if one can solve Fixed-CDH then one can compute multiplication modulo r using implicit representatives.

Exercise 21.4.4. Prove Lemmas 21.4.2 and 21.4.3.

Lemma 21.4.5. *Let g have order r. Let h_1 be an implicit representation of a such that $h_1 \neq 1$ (in other words, $a \not\equiv 0 \pmod{r}$).*

1. *Given a perfect CDH oracle one can compute an implicit representation for $a^{-1} \pmod{r}$ using one oracle query.*

2. *Given a perfect Fixed-CDH oracle with respect to g one can compute an implicit representation for $a^{-1} \pmod{r}$ using $\leq 2\log_2(r)$ oracle queries.*

Proof: Given a perfect CDH oracle A one calls $A(g^a, g, g) = g^{a^{-1} \pmod{r}}$. Given a perfect Fixed-CDH oracle one computes $g^{a^{r-2} \pmod{r}}$ as was done in Lemma 21.1.15. □

To summarise, since $\mathbb{Z}/r\mathbb{Z} \cong \mathbb{F}_r$, given a perfect CDH or Fixed-CDH oracle then one can perform all field operations in $\mathbb{F}_r$ using implicit representations. Boneh and Lipton [83] call the set of implicit representations for $\mathbb{Z}/r\mathbb{Z}$ a **black box field**.

21.4.2 The den Boer Reduction

We now present the **den Boer reduction** [169], which applies when $r-1$ is smooth. The crucial idea is that the Pohlig-Hellman and baby-step-giant-step methods only require the ability to add, multiply and compare group elements. Hence, if a perfect CDH oracle is given then these algorithms can be performed using implicit representations.

Theorem 21.4.6. *Let $g \in G$ have prime order r. Suppose l is the largest prime factor of $r-1$. Let A be a perfect oracle for the Fixed-CDH problem with respect to g. Then one can solve the DLP in $\langle g \rangle$ using $O(\log(r)\log(\log(r)))$ oracle queries, $O(\log(r)(\sqrt{l}/\log(l) + \log(r))$ multiplications in $\mathbb{F}_r$ and $O(\sqrt{l}\log(r)^2/\log(l))$ operations in G (where the constant implicit in the $O(\cdot)$ does not depend on l).*

Proof: Let the challenge DLP instance be $g, h = g^a$. If $h = 1$ then return $a = 0$. Hence, we now assume $1 \leq a < r$. We can compute a primitive root $\gamma \in \mathbb{F}_r^*$ in $O(\log(r)\log(\log(r)))$ operations in $\mathbb{F}_r$ (see Section 2.15). The (unknown) logarithm of h satisfies

$$a \equiv \gamma^u \pmod{r} \tag{21.5}$$

for some integer u. To compute a it is sufficient to compute u.[3] The idea is to solve the DLP in equation (21.5) using the implicit representation of a. Since $r-1$ is assumed to be smooth then we can use the Pohlig-Hellman (PH) method, followed by the baby-step-giant-step (BSGS) method in each subgroup. We briefly sketch the details.

Write $r - 1 = \prod_{i=1}^{n} l_i^{e_i}$ where the l_i are prime. The PH method involves projecting a and γ into the subgroup of $\mathbb{F}_r^*$ of order $l_i^{e_i}$. In other words, we must compute

$$h_i = g^{a^{(r-1)/l_i^{e_i}}}$$

[3]It may seem crazy to try to work out u without knowing a, but it works!

for $1 \le i \le n$. Using the Fixed-CDH oracle to perform computations in implicit representation, Algorithm 4 computes all the h_i together in $O(\log(r)\log\log(r))$ oracle queries.[4] A further $O(\log(r))$ oracle queries are required to compute all $g^{a^{(r-1)/l_i^f}}$ where $0 \le f < e_i$. Similarly one computes all $x_i = \gamma^{(r-1)/l_i^{e_i}}$ in $O(\log(r)\log\log(r))$ multiplications in $\mathbb{F}_r$. We then have

$$h_i = g^{x_i^{u \pmod{l_i^{e_i}}}}.$$

Following Section 13.2 one reduces these problems to $\sum_{i=1}^n e_i$ instances of the DLP in groups of prime order l_i. This requires $O(\log(r)^2)$ group operations and field operations overall (corresponding to the computations in line 6 of Algorithm 13).

For the baby-step-giant-step algorithm, suppose we wish to solve $g^a = g^{\gamma^u}$ (where, for simplicity, we redefine a and γ so that they now have order l modulo r). Set $m = \lceil \sqrt{l} \rceil$ and write $u = u_0 + m u_1$ where $0 \le u_0, u_1 < m$. From

$$g^a = g^{\gamma^u} = g^{\gamma^{u_0 + m u_1}} = g^{\gamma^{u_0}(\gamma^m)^{u_1}} \tag{21.6}$$

one has

$$(g^a)^{(\gamma^{-m})^{u_1}} = g^{\gamma^{u_0}}. \tag{21.7}$$

We compute and store (in a sorted structure) the baby steps g^{γ^i} for $i = 0, 1, 2, \dots, m-1$ (this involves computing one exponentiation in G at each step, as $g^{\gamma^{i+1}} = (g^{\gamma^i})^{\gamma}$, which is at most $2\log_2(r)$ operations in G).

We then compute the giant steps $(g^a)^{\gamma^{-mj}}$. This involves computing $w_0 = \gamma^{-m} \pmod r$ and then the sequence $w_j = \gamma^{-mj} \pmod r$ as $w_{j+1} = w_j w_0 \pmod r$; this requires $O(\log(m) + m)$ multiplications in $\mathbb{F}_r$. We also must compute $(g^a)^{w_j}$, each of which requires $\le 2\log_2(r)$ operations in G.

When we find a match then we have solved the DLP in the subgroup of order l. The BSGS algorithm for each prime l requires $O(\sqrt{l}\log(r))$ group operations and $O(\sqrt{l} + \log(r))$ operations in $\mathbb{F}_r$. There are $O(\log(r))$ primes l for which the BSGS must be run, but a careful analysis of the cost (using the result of Exercise 13.2.7) gives an overall running time of $O(\log(r)^2\sqrt{l}/\log(l))$ group operations and $O(\log(r)^2 + \log(r)\sqrt{l}/\log(l))$ multiplications in $\mathbb{F}_r$. Note that the CDH oracle is not required for the BSGS algorithm.

Once u is determined modulo all prime powers $l^e \mid (r-1)$ one uses the Chinese remainder theorem to compute $u \in \mathbb{Z}/(r-1)\mathbb{Z}$. Finally, one computes $a = \gamma^u \pmod r$. These final steps require $O(\log(r))$ operations in $\mathbb{F}_r$. □

Corollary 21.4.7. *Let $A(\kappa)$ be an algorithm that outputs triples (g, h, r) such that r is a κ-bit prime, g has order r, $r-1$ is $O(\log(r)^2)$-smooth, and $h \in \langle g \rangle$. Then DLP $\le_R$ Fixed-CDH for the problem instances output by A.*

Proof: Suppose one has a perfect Fixed-CDH oracle. Putting $l = O(\log(r)^2)$ into Theorem 21.4.6 gives a reduction with $O(\log(r)\log\log(r))$ oracle queries and $O(\log(r)^3)$ group and field operations. □

The same results trivially hold if one has a perfect CDH oracle.

Exercise 21.4.8.★ Determine the complexity in Theorem 21.4.6 if one has a Fixed-CDH oracle that only succeeds with probability ϵ.

[4]Remark 2.15.9 does not lead to a better bound, since the value n (which is m in the notation of that remark) is not necessarily large.

Cherepnev [135] iterates the den Boer reduction to show that if one has an efficient CDH algorithm for arbitrary groups then one can solve DLP in a given group in subexponential time. This result is of a very different flavour to the other reductions in this chapter (which all use an oracle for a group G to solve a computational problem in the same group G) so we do not discuss it further.

21.4.3 The Maurer Reduction

The den Boer reduction can be seen as solving the DLP in the algebraic group $G_m(\mathbb{F}_r)$, performing all computations using implicit representation. Maurer's idea was to replace $G_m(\mathbb{F}_r)$ by any algebraic group $G(\mathbb{F}_r)$, in particular the group of points on an elliptic curve $E(\mathbb{F}_r)$. As with Lenstra's elliptic curve factoring method, even when $r-1$ is not smooth then there might be an elliptic curve E such that $E(\mathbb{F}_r)$ is smooth.

When one uses a general algebraic group G there are two significant issues that did not arise in the den Boer reduction.

- The computation of the group operation in G may require inversions. This is true for elliptic curve arithmetic using affine coordinates.
- Given $h = g^a$ one must be able to compute an element $P \in G(\mathbb{F}_r)$, in implicit representation, such that once P has been determined in explicit representation one can compute a. For an elliptic curve E one could hope that $P = (a, b) \in E(\mathbb{F}_r)$ for some $b \in \mathbb{F}_r$.

Before giving the main result we address the second of these issues. In other words, we show how to embed a DLP instance into an elliptic curve point.

Lemma 21.4.9. *Let g have prime order r and let $h = g^a$. Let $E : y^2 = x^3 + Ax + B$ be an affine elliptic curve over $\mathbb{F}_r$. Given a perfect Fixed-CDH oracle there is an algorithm that outputs an implicit representation (g^X, g^Y) of a point $(X, Y) \in E(\mathbb{F}_r)$ and some extra data, and makes an expected $O(\log(r))$ oracle queries and performs an expected $O(\log(r))$ group operations in $\langle g \rangle$. Furthermore, given the explicit value of X and the extra data one can compute a.*

Proof: The idea is to choose uniformly at random $0 \le \alpha < r$ and set $X = a + \alpha$. An implicit representation of X can be computed as $h_1 = hg^\alpha$ using $O(\log(r))$ group operations. If we store α then, given X, we can compute a. Hence, the extra data is α.

Given the implicit representation for X one determines an implicit representation for $\beta = X^3 + AX + B$ using two oracle queries. Given g^β one can compute (here $(\frac{\beta}{r}) \in \{-1, 1\}$ is the Legendre symbol)

$$h_2 = g^{(\frac{\beta}{r})} = g^{\beta^{(r-1)/2}} \tag{21.8}$$

using $O(\log(r))$ oracle queries. If $h_2 = g$ then β is a square and so X is an x-coordinate of a point of $E(\mathbb{F}_r)$.

Since there are at least $(r - 2\sqrt{r})/2$ possible x-coordinates of points in $E(\mathbb{F}_r)$ it follows that if one chooses X uniformly at random in $\mathbb{F}_r$ then the expected number of trials until X is the x-coordinate of a point in $E(\mathbb{F}_r)$ is approximately two.

Once β is a square modulo r then one can compute an implicit representation for $Y = \sqrt{\beta} \pmod{r}$ using the Tonelli-Shanks algorithm with implicit representations. We use the notation of Algorithm 3. The computation of the non-residue n is expected to require $O(\log(r))$ operations in $\mathbb{F}_r$ and can be done explicitly. The computation of the terms w and b requires $O(\log(r))$ oracle queries, some of which can be avoided by storing intermediate values from the computation in equation (21.8). The computation of i using a Pohlig-Hellman-style algorithm is done as follows. First compute the sequence $b, b^2, \dots, b^{2^{e-1}}$ using $O(\log(r))$ oracle queries and the sequence $y, y^2, \dots, y^{2^{e-1}}$ using $O(\log(r))$ group operations. With a further $O(\log(r))$ group operations one can determine the bits of i. $\square$

Theorem 21.4.10. *Let $B \in \mathbb{N}$. Let $g \in G$ have order r. Let E be an elliptic curve over $\mathbb{F}_r$ such that $E(\mathbb{F}_r)$ is a cyclic group. Suppose that the order of $E(\mathbb{F}_r)$ is known and is B-smooth. Given a perfect Fixed-CDH oracle with respect to g one can solve the DLP in $\langle g \rangle$ using an expected $O(\log(r)^2 \log(\log(r)))$ oracle queries.*[5]

Indeed, there are two variants of the reduction, one using exhaustive search and one using the baby-step-giant-step algorithm. One can also consider the case of a perfect CDH oracle. The following table gives the full expected complexities (where the constant implicit in the $O(\cdot)$ is independent of B). We use the abbreviation $l(x) = \log(x)$, so that $l(l(r)) = \log(\log(r))$.

Oracle	*Reduction*	*Oracle queries*	*Group operations*	$\mathbb{F}_r$ *operations*
Fixed-CDH	*PH only*	$O(l(r)^2 l(l(r)))$	$O(Bl(r)^2/l(B))$	$O(Bl(r)^2/l(B))$
Fixed-CDH	*PH+BSGS*	$O(\sqrt{B}l(r)^2/l(B) + l(r)^2 l(l(r)))$	$O(\sqrt{B}l(r)^2/l(B))$	$O(\sqrt{B}l(r)^2/l(B))$
CDH	*PH only*	$O(l(r)l(l(r)))$	$O(Bl(r)^2/l(B))$	$O(Bl(r)^2/l(B))$
CDH	*PH+BSGS*	$O(\sqrt{B}l(r)/l(B) + l(r)l(l(r)))$	$O(\sqrt{B}l(r)^2/l(B))$	$O(\sqrt{B}l(r)^2/l(B))$

Proof: Let the discrete logarithm instance be $(g, h = g^a)$. Write $N = \#E(\mathbb{F}_r) = \prod_{i=1}^{k} l_i^{e_i}$. We assume that affine coordinates are used for arithmetic in $E(\mathbb{F}_r)$. Let P be a generator of $E(\mathbb{F}_r)$.

The reduction is conceptually the same as the den Boer reduction. One difference is that elliptic curve arithmetic requires inversions (which are performed using the method of Lemma 21.1.13 and Lemma 21.1.15), hence the number of Fixed-CDH oracle queries must increase. A sketch of the reduction in the case of exhaustive search is given in Algorithm 27.

The first step is to use Lemma 21.4.9 to associate with h the implicit representations of a point $Q \in E(\mathbb{F}_r)$. This requires an expected $O(\log(r))$ oracle queries and $O(\log(r))$ group operations for all four variants. Then $Q \in \langle P \rangle$ where P is the generator of the cyclic group $E(\mathbb{F}_r)$.

The idea is again to use Pohlig-Hellman (PH) and baby-step-giant-step (BSGS) to solve the discrete logarithm of Q with respect to P in $E(\mathbb{F}_r)$. If we can compute an integer u such that $Q = [u]P$ (with computations done in implicit representation) then computing $[u]P$ and using Lemma 21.4.9 gives the value a explicitly.

First we consider the PH algorithm. As with the den Boer reduction, one needs to compute explicit representations (i.e., standard affine coordinates) for $[N/l_i^{e_i}]P$ and implicit representations for $[N/l_i^{e_i}]Q$. It is possible that $[N/l_i^{e_i}]Q = \mathcal{O}_E$ so this case must be handled. As in Section 2.15.1, computing these points requires $O(\log(r)\log\log(r))$ elliptic curve operations. Hence, for the multiples of P we need $O(\log(r)\log\log(r))$ operations in $\mathbb{F}_r$ while for the multiples of Q we need $O(\log(r)^2\log\log(r))$ Fixed-CDH oracle queries and $O(\log(r)\log\log(r))$ group operations. (If a CDH oracle is available then this stage only requires $O(\log(r)\log\log(r))$ oracle queries, as an inversion in implicit representation can be done with a single CDH oracle query.) Computing the points $[N/l_i^f]P$ for $1 \le f < e_i$ and all i requires at most a further $2\sum_{i=1}^{k} e_i \log_2(l_i) = 2\log_2(N) = O(\log(r))$ group operations. Similarly, computing the implicit representations of the remaining $[N/l_i^f]Q$ requires $O(\log(r)^2)$ Fixed-CDH oracle queries and $O(\log(r))$ group operations.

[5]This is improved to $O(\log(r)\log\log(r))$ in Remark 21.4.11.

The computation of u_iP_0 in line 8 of Algorithm 27 requires $O(\log(r))$ operations in $\mathbb{F}_r$ followed by $O(1)$ operations in G and oracle queries.

The exhaustive search algorithm for the solution to the DLP in a subgroup of prime order l_i is given in lines 9 to 16 of Algorithm 27. The point P_0 in line 8 has already been computed, and computing Q_0 requries only one elliptic curve addition (i.e., $O(\log(r))$ Fixed-CDH oracle queries). The while loop in line 12 runs for $\le B$ iterations, each iteration involves a constant number of field operations to compute $T + P_0$ followed by two exponentiations in the group to compute g^{x_T} and g^{y_T} (an obvious improvement is to use g^{x_T} only). The complexity of lines 9 to 16 is therefore $O(B\log(r))$ group operations, and $O(B)$ field operations.

If one uses BSGS the results are similar. Suppose Q and P are points of order l, where P is known explicitly while we only have an implicit representation (g^{x_Q}, g^{y_Q}) for Q. Let $m = \lceil\sqrt{l}\rceil$ and $P_1 = [m]P$ so that $Q = [u_0]P + [u_1]P_1$ for $0 \le u_0, u_1 < m$. One computes a list of baby steps $[u_0]P$ in implicit representation using $O(\sqrt{B})$ field operations and $O(\sqrt{B}\log(r))$ group operations as above. For the giant steps $Q - [u_1]P_1$ one is required to perform elliptic curve arithmetic with the implicit point Q and the explicit point $[u_1]P_1$, which requires an inversion of an implicit element. Hence the giant steps require $O(\sqrt{B})$ field operations, $O(\sqrt{B}\log(r))$ group operations and $O(\sqrt{B}\log(r))$ Fixed-CDH oracle queries.

Since $\sum_{i=1}^{k} e_i \le \log_2(N)$ the exhaustive search or BSGS subroutine is performed $O(\log(r))$ times. A more careful analysis using Exercise 13.2.7 means the complexity is multiplied by $\log(r)/\log(B)$. The Chinese remainder theorem and later stages are negligible. The result follows. □

Algorithm 27 Maurer reduction

```
INPUT: $g, h = g^a$, $E(\mathbb{F}_r)$
OUTPUT: $a$
1: Associate to $h$ an implicit representation for a point $Q = (X,Y) \in E(\mathbb{F}_r)$ using
     Lemma 21.4.9
2: Compute a point $P \in E(\mathbb{F}_r)$ that generates $E(\mathbb{F}_r)$. Let $N = \#E(\mathbb{F}_r) = \prod_{i=1}^{k} l_i^{e_i}$
3: Compute explicit representations of $\{[N/l_i^j]P : 1 \le i \le k, 1 \le j \le e_i\}$
4: Compute implicit representations of $\{[N/l_i^j]Q : 1 \le i \le k, 1 \le j \le e_i\}$
5: for $i = 1$ to $k$ do
6:     $u_i = 0$
7:     for $j = 1$ to $e_i$ do          ▷ Reducing DLP of order $l_i^{e_i}$ to cyclic groups
8:         Let $P_0 = [N/l_i^j]P$ and $Q_0 = [N/l_i^j]Q - u_iP_0$
9:         if $Q_0 \ne \mathcal{O}_E$ then
10:            Let $(h_{0,x}, h_{0,y})$ be the implicit representation of $Q_0$
11:            $P_0 = [N/l_i]P_0$, $n = 1$, $T = P_0 = (x_T, y_T)$
12:            while $h_{0,x} \ne g^{x_T}$ or $h_{0,y} \ne g^{y_T}$ do          ▷ Exhaustive search
13:                $n = n + 1$, $T = T + P_0$
14:            end while
15:            $u_i = u_i + nl^{j-1}$
16:        end if
17:    end for
18: end for
19: Use Chinese remainder theorem to compute $u \equiv u_i \pmod{l_i^{e_i}}$ for $1 \le i \le k$
20: Compute $(X,Y) = [u]P$ and hence compute $a$
21: return $a$
```

Remark 21.4.11. We have seen that reductions involving a Fixed-CDH oracle are less efficient (i.e., require more oracle queries) than reductions using a CDH oracle. A solution[6] to this is to work with projective coordinates for elliptic curves. Line 12 of Algorithm 27 tests whether the point Q_0 given in implicit representation is equal to the point (x_T, y_T) given in affine representation. When $Q_0 = (x_0 : y_0 : z_0)$ then the test $h_{0,x} = g^{x_T}$ in line 12 is replaced with the comparison

$$g^{x_0} = (g^{z_0})^{x_T} .$$

Hence the number of oracle queries in the first line of the table in Theorem 21.4.10 can be reduced to $O(\log(r) \log\log(r))$. As mentioned in Remark 13.3.2, one cannot use the BSGS algorithm with projective coordinates, as the non-uniqueness of the representation means one can't efficiently detect a match between two lists.

Exercise 21.4.12.★ Generalise the Maurer algorithm to the case where the group of points on the elliptic curve is not necessarily cyclic. Determine the complexity if l_1 is the largest prime for which $E(\mathbb{F}_r)[l_1]$ is not cyclic and l_2 is the largest prime dividing $\#E(\mathbb{F}_r)$ for which $E(\mathbb{F}_r)[l_2]$ is cyclic.

Exercise 21.4.13. If $r+1$ is smooth then one can use the algebraic group $G_{2,r} \cong \mathbb{T}_2(\mathbb{F}_r)$ (see Section 6.3) instead of $G_m(\mathbb{F}_r)$ or $E(\mathbb{F}_r)$. There are two approaches: the first is to use the usual representation $\{a + b\theta \in \mathbb{F}_{r^2} : \mathrm{N}_{\mathbb{F}_{r^2}/\mathbb{F}_r}(a + b\theta) = 1\}$ for $G_{2,r}$ and the second is to use the representation $\mathbb{A}^1(\mathbb{F}_r)$ for $\mathbb{T}_2(\mathbb{F}_r) - \{1\}$ corresponding to the map decomp_2 from Definition 6.3.7. Determine the number of (perfect) oracle queries in the reductions from Fixed-CDH to DLP for these two representations. Which is better? Repeat the exercise when one has a CDH oracle.

Corollary 21.4.14. *Let $c \in \mathbb{R}_{>1}$. Let (G_n, g_n, r_n) be a family of groups for $n \in \mathbb{N}$ where $g_n \in G_n$ has order r_n and r_n is an n-bit prime. Suppose we are given* ***auxiliary elliptic curves*** *(E_n, N_n) for the family, where E_n is an elliptic curve over $\mathbb{F}_{r_n}$ such that $\#E_n(\mathbb{F}_{r_n}) = N_n$ and N_n is $O(\log(r_n)^c)$-smooth. Then the DLP in $\langle g_n \rangle$ is equivalent to the Fixed-CDH problem in $\langle g_n \rangle$.*

Exercise 21.4.15. Prove Corollary 21.4.14.

We now state the conjecture of Maurer and Wolf that all Hasse intervals contain a polynomially smooth integer. Define $\nu(r)$ to be the minimum, over all integers $n \in [r + 1 - 2\sqrt{r}, r + 1 + 2\sqrt{r}]$, of the largest prime divisor of n. Conjecture 1 of [407] states that

$$\nu(r) = \log(r)^{O(1)}. \tag{21.9}$$

See Remark 15.3.5 for discussion of this. Muzereau, Smart and Vercauteren [448] note that if r is a pseudo-Mersenne prime (as is often used in elliptic curve cryptography) then the Hasse interval usually contains a power of 2. Similarly, as noted by Maurer and Wolf in [405], one can first choose a random smooth integer n and then search for a prime r close to n and work with a group G of order r.

Exercise 21.4.16.★ Show how to use the algorithm of Section 19.4.4 to construct a smooth integer in the Hasse interval. Construct a 2^{40}-smooth integer (not equal to 2^{255}) close to $p = 2^{255} - 19$ using this method.

[6]This idea is briefly mentioned in Section 3 of [402], but was explored in detail by Bentahar [42].

Remark 21.4.17. There are two possible interpretations of Corollary 21.4.14. The first interpretation is: if there exists an efficient algorithm for CDH or Fixed-CDH in a group $G = \langle g \rangle$ of prime order r and if there exists an auxiliary elliptic curve over $\mathbb{F}_r$ with sufficiently smooth order then there exists an efficient algorithm to solve the DLP in G. Maurer and Wolf [408] (also see Section 3.5 of [409]) claim this gives a non-uniform reduction from DLP to CDH, however the validity of this claim depends on the DLP instance generator.[7]

In other words, if one believes that there does not exist a non-uniform polynomial-time algorithm for DLP in G (for certain instance generators) and if one believes the conjecture that the Hasse interval around r contains a polynomially smooth integer, then one must believe there is no polynomial-time algorithm for CDH or Fixed-CDH in G. Hence, one can use the results to justify the assumption that CDH is hard. We stress that this is purely a statement of existence of algorithms; it is independent of the issue of whether or not it is feasible to write the algorithms down.

A second interpretation is that CDH might be easy and that this reduction yields the best algorithm for solving the DLP. If this were the case (or if one wants a uniform reduction) then, in order to solve a DLP instance, the issue of how to implement the DLP algorithm becomes important. The problem is that there is no known polynomial-time algorithm to construct auxiliary elliptic curves $E(\mathbb{F}_r)$ of smooth order. An algorithm to construct smooth curves (based on the CM method) is given in Section 4 of [405] but it has exponential complexity. Hence, if one can write down an efficient algorithm for CDH then the above ideas alone do not allow one to write down an efficient algorithm for DLP.

Boneh and Lipton [83] handle the issue of auxiliary elliptic curves by giving a subexponential-time reduction between Fixed-CDH and DLP. They make the natural assumption (essentially Conjecture 15.3.1; as used to show that the elliptic curve factoring method is subexponential-time) that, for sufficiently large primes, the probability that a randomly chosen integer in the Hasse interval $[r + 1 - 2\sqrt{r}, r + 1 + 2\sqrt{r}]$ is $L_r(1/2, c)$-smooth is $1/L_r(1/2, c')$ for some constants $c, c' > 0$ (see Section 15.3 for further discussion of these issues). By randomly choosing $L_r(1/2, c')$ elliptic curves over $\mathbb{F}_r$ one therefore expects to find one that has $L_r(1/2, c)$-smooth order. One can then perform Algorithm 27 to solve an instance of the DLP in subexponential-time and using polynomially many oracle queries. We refer to [83] for the details.

Maurer and Wolf extend the Boneh-Lipton idea to genus 2 curves and use results of Lenstra, Pila and Pomerance (Theorem 1.3 of [380]) to obtain a reduction with proven complexity $L_r(2/3, c)$ for some constant c (see Section 3.6 of [409]). This is the only reduction from DLP to CDH that does not rely on any conjectures or heuristics. Unfortunately it is currently impractical to construct suitable genus 2 curves in practice (despite being theoretically polynomial-time).

Muzereau, Smart and Vercauteren [448] go even further than Boneh and Lipton. They allow an exponential-time reduction, with the aim of minimising the number of CDH or Fixed-CDH oracle queries. The motivation for this approach is to give tight reductions between CDH and DLP (i.e., to give a lower bound on the running time for an algorithm

[7]An instance generator for the DLP (see Example 2.1.9) outputs a quadruple (G, r, g, h) where G is a description of a group, $g \in G$ has order r, $h \in \langle g \rangle$ and r is prime. The size of the instance depends on the representation of G and g, but is at least $2\log_2(r)$ bits since one must represent r and h. If one considers the DLP with respect to an instance generator for which r is constant over all instances of a given size n, then a single auxiliary curve is needed for all DLP instances of size n and so Corollary 21.4.14 gives a non-uniform reduction. On the other hand, if there are superpolynomially many r among the outputs of size n of the instance generator (this would be conjecturally true for the instance generator of Example 2.1.9) then the amount of auxiliary data is not polynomially bounded and hence the reduction is not non-uniform.

for CDH in terms of conjectured lower bounds for the running time of an algorithm for DLP). Their results were improved by Bentahar [42, 43]. It turns out to be desirable to have an auxiliary elliptic curve such that $\#E(\mathbb{F}_r)$ is a product of three coprime integers of roughly equal size $r^{1/3}$. The reduction then requires $O(\log(r))$ oracle queries but $O(r^{1/3}\log(r))$ field operations. Islam [306] has proved that such an elliptic curve exists for each prime r. One can construct auxiliary curves by choosing random curves, counting points and factoring; one expects only polynomially many trials, but the factoring computation is subexponential. We refer to [448, 42, 43] for further details.

Exercise 21.4.18. Write down the algorithm for the Muzereau-Smart-Vercauteren reduction using projective coordinates. Prove that the algorithm has the claimed complexity.

Exercise 21.4.19. Show how to generate in heuristic expected polynomial-time primes $r, p \equiv 2 \pmod 3$ such that $r \mid (p+1)$, $r+1$ is κ-smooth, and $2^{\kappa-1} \le r < p \le 2^{\kappa+3}$. Hence, by Exercise 9.10.4, taking $E : y^2 = x^3 + 1$ then $E(\mathbb{F}_p)$ is a group of order divisible by r and $E(\mathbb{F}_r)$ has κ-smooth order and is a suitable auxiliary elliptic curve for the Maurer reduction.

Finally, we remark that the den Boer and Maurer reductions cannot be applied to relate CDH and DLP in groups of unknown order. For example, let N be composite and $g \in (\mathbb{Z}/N\mathbb{Z})^*$ of unknown order M. Given a perfect Fixed-CDH oracle with respect to g one can still compute with the algebraic group $G_m(\mathbb{Z}/M\mathbb{Z})$ in implicit representation (or projective equations for $E(\mathbb{Z}/M\mathbb{Z})$), but if M is not known then the order of $G = G_m(\mathbb{Z}/M\mathbb{Z})$ (respectively, $G = E(\mathbb{Z}/M\mathbb{Z})$) is also not known and so one cannot perform the Pohlig-Hellman algorithm in G. Later we will mention how a CDH oracle in $(\mathbb{Z}/N\mathbb{Z})^*$ can be used to factor N (see Exercise 24.2.23) and hence avoid this problem in that group.

21.5 Algorithms for Static Diffie-Hellman

Brown and Gallant [111] studied the relationship between Static-DH and DLP. Their main result is an algorithm to solve an instance of the DLP using a perfect Static-DH oracle. Cheon [131] independently discovered this algorithm in a different context, showing that a variant of the DLP (namely, the problem of computing a given g, g^a and g^{a^d}; we call this **Cheon's variant of the DLP**) can be significantly easier than the DLP. We now present the algorithm of Brown-Gallant and Cheon, and discuss some of its applications.

Theorem 21.5.1. *Let g have prime order r and let $d \mid (r-1)$. Given $h_1 = g^a$ and $h_d = g^{a^d}$ then one can compute a in $O((\sqrt{(r-1)/d} + \sqrt{d})\log(r))$ group operations, $O(\sqrt{(r-1)/d} + \sqrt{d})$ group elements of storage and $O(\sqrt{(r-1)/d} + \sqrt{d})$ multiplications in $\mathbb{F}_r$.*[8]

Proof: First, the case $a \equiv 0 \pmod r$ is easy, so we assume $a \not\equiv 0 \pmod r$. The idea is essentially the same as the den Boer reduction. Let γ be a primitive root modulo r. Then $a = \gamma^u \pmod r$ for some $0 \le u < r-1$ and it suffices to compute u. The den Boer reduction works by projecting the unknown a into prime order subgroups of $\mathbb{F}_r^*$ using a Diffie-Hellman oracle. In our setting, we already have an implicit representation of the projection a^d into the subgroup of $\mathbb{F}_r^*$ of order $(r-1)/d$.

[8] As usual, we are being careless with the $O(\cdot)$-notation. What we mean is that there is a constant c independent of r, d, g and a such that the algorithm requires $\le c(\sqrt{(r-1)/d}+\sqrt{d})\log(r)$ group operations.

The first step is to solve $h_d = g^{a^d} = g^{\gamma^{du}}$ for some $0 \leq u \leq (r-1)/d$. Let $m = \lceil \sqrt{(r-1)/d} \rceil$ and write $u = u_0 + mu_1$ with $0 \leq u_0, u_1 < m$. This is exactly the setting of equations (21.6) and (21.7) and hence one can compute (u_0, u_1) using a baby-step-giant-step algorithm. This requires $\leq m$ multiplications in $\mathbb{F}_r$ and $\leq 2m$ exponentiations in the group. Thus the total complexity is $O(\sqrt{(r-1)/d}\log(r))$ group operations and $O(\sqrt{(r-1)/d})$ field operations.

We now have $a^d = \gamma^{du}$ and so $a = \gamma^{u+v(r-1)/d}$ for some $0 \leq v < d$. It remains to compute v. Let

$$h = h_1^{\gamma^{-u}} = g^{a\gamma^{-u}} = g^{\gamma^{v(r-1)/d}}.$$

Set $m = \lceil \sqrt{d} \rceil$ and write $v = v_0 + mv_1$ where $0 \leq v_0, v_1 < m$. Using the same ideas as above (since γ is known explicitly the powers are computed efficiently) one can compute (v_0, v_1) using a baby-step-giant-step algorithm in $O(\sqrt{d}\log(r))$ group operations. Finally, we compute $a = \gamma^{u+v(r-1)/d} \pmod{r}$. $\square$

Kozaki, Kutsuma and Matsuo [353] show how to reduce the complexity in the above result to $O(\sqrt{(r-1)/d} + \sqrt{d})$ group operations by using precomputation to speed up the exponentiations to constant time. Note that this trick requires exponential storage and is not applicable when low-storage discrete logarithm algorithms are used (as in Exercise 21.5.5).

The first observation is that if $r-1$ has a suitable factorisation then Cheon's variant of the DLP can be much easier than the DLP.

Corollary 21.5.2. *Let g have prime order r and suppose $r-1$ has a factor d such that $d \approx r^{1/2}$. Given $h_1 = g^a$ and $h_d = g^{a^d}$ then one can compute a in $O(r^{1/4}\log(r))$ group operations.*

Corollary 21.5.3. *Let g have prime order r and suppose $r - 1 = \prod_{i=1}^n d_i$ where the d_i are coprime. Given $h_1 = g^a$ and $h_{d_i} = g^{a^{d_i}}$ for $1 \leq i \leq n$ then one can compute a in $O((\sum_{i=1}^n \sqrt{d_i})\log(r))$ group operations.*

Exercise 21.5.4. Prove Corollaries 21.5.2 and 21.5.3.

As noted in [111] and [131] one can replace the baby-step-giant-step algorithms by Pollard methods. Brown and Gallant[9] suggest a variant of the Pollard rho method, but with several non-standard features: one needs to find the precise location of the collision (i.e., steps $x_i \neq x_j$ in the walk such that $x_{i+1} = x_{j+1}$) and there is only a (heuristic) 0.5 probability that a collision leads to a solution of the DLP. Cheon [131] suggests using the Kangaroo method, which is a more natural choice for this application.

Exercise 21.5.5. Design a pseudorandom walk for the Pollard kangaroo method to solve the DLP in implicit representation arising in the proof of Theorem 21.5.1.

Brown and Gallant use Theorem 21.5.1 to obtain the following result.

Theorem 21.5.6. *Let g have prime order r and let $d \mid (r-1)$. Let $h = g^a$ and suppose A is a perfect oracle for the static Diffie-Hellman problem with respect to (g, h) (i.e., $A(h_1) = h_1^a$). Then one can compute a using d oracle queries, $O((\sqrt{(r-1)/d}+\sqrt{d})\log(r))$ group operations and $O((\sqrt{(r-1)/d} + \sqrt{d})\log(r))$ multiplications in $\mathbb{F}_r$.*

Proof: Write $h_1 = h = g^a$ and compute the sequence $h_{i+1} = O(h_i) = g^{a^i}$ until g^{a^d} is computed. Then apply Theorem 21.5.1. $\square$

[9]See Appendix B.2 of the first version of [111]. This does not appear in the June 2005 version.

Note that the reduction uses a Static-DH oracle with respect to g^a to compute a. The reduction does not solve a general instance of the DLP using a specific Static-DH oracle, hence it is not a reduction from DLP to Static-DH. Also recall that Exercise 20.4.6 showed how one can potentially compute a efficiently given access to a Static-DH oracle (with respect to a) that does not check that the inputs are group elements of the correct order. Hence, the Brown-Gallant result is primarily interesting in the case where the Static-DH oracle does perform these checks.

Corollary 21.5.7. *Let g have prime order r and suppose $r-1$ has a factor d such that $d \approx r^{1/3}$. Given $h = g^a$ and a perfect Static-DH oracle with respect to (g,h) then one can compute a in $O(r^{1/3})$ oracle queries and $O(r^{1/3}\log(r))$ group operations.*

Exercise 21.5.8. Prove Corollary 21.5.7.

Brown and Gallant use Theorem 21.5.6 to give a lower bound on the difficulty of Static-DH under the assumption that the DLP is hard.

Exercise 21.5.9. Let g have order r. Assume that the best algorithm to compute a, given $h = g^a$, requires $\sqrt{r}$ group operations. Suppose that $r-1$ has a factor $d = c_1\log(r)^2$ for some constant c_1. Prove that the best algorithm to solve Static-DH with respect to (g,h) requires at least $c_2\sqrt{r}/\log(r)^2$ group operations for some constant c_2.

All the above results are predicated on the existence of a suitable factor d of $r-1$. Of course, $r-1$ may not have a factor of the correct size; for example if $r-1 = 2l$ where l is prime then we have shown that given (g, g^a, g^{a^2}) one can compute a in $O(\sqrt{r/2}\log(r))$ group operations, which is no better than general methods for the DLP. To increase the applicability of these ideas, Cheon also gives a method for when there is a suitable factor d of $r+1$. The method in this case is not as efficient as the $r-1$ case, and requires more auxiliary data.

Theorem 21.5.10. *Let g have prime order r and let $d \mid (r+1)$. Given $h_i = g^{a^i}$ for $1 \le i \le 2d$ then one can compute a in $O((\sqrt{(r+1)/d} + d)\log(r))$ group operations, $O(\sqrt{(r+1)/d} + \sqrt{d})$ group elements storage and $O((\sqrt{(r+1)/d} + \sqrt{d})\log(r))$ multiplications in $\mathbb{F}_r$.*

Proof: As in Exercise 21.4.13 the idea is to work in the algebraic group $G_{2,r}$, which has order $r+1$. Write $\mathbb{F}_{r^2} = \mathbb{F}_r(\theta)$ where $\theta^2 = t \in \mathbb{F}_r$. By Lemma 6.3.10 each element $\alpha \in G_{2,r} - \{1\} \subseteq \mathbb{F}_{r^2}^*$ is of the form $\alpha_0 + \alpha_1\theta$ where

$$\alpha_0 = \frac{a^2 - t}{a^2 + t}, \qquad \alpha_1 = \frac{2a}{a^2 + t}$$

for some $a \in \mathbb{F}_r$. For each $d \in \mathbb{N}$ there exist polynomials $f_{d,0}(x), f_{d,1}(x) \in \mathbb{F}_r[x]$ of degree $2d$ such that, for α as above, one has

$$\alpha^d = \frac{f_{d,0}(a) + \theta f_{d,1}(a)}{(a^2 + t)^d}.$$

The idea is to encode the DLP instance g^a into the element $\beta \in G_{2,r}$ as

$$\beta = \frac{a^2 - t}{a^2 + t} + \theta\frac{2a}{a^2 + t}.$$

We do not know β, but we can compute $(a^2 - t)$, $(a^2 + t)$ and $2a$ in implicit representation.

Let γ be a generator for $G_{2,r}$, known explicitly. Then $\beta = \gamma^u$ for some $0 \le u < r+1$. It suffices to compute u.

The first step is to project into the subgroup of order $(r+1)/d$. We have $\beta^d = \gamma^{du}$ for some $0 \le u < (r+1)/d$. Let $m = \lceil \sqrt{(r+1)/d} \rceil$ so that $u = u_0 + mu_1$ for $0 \le u_0, u_1 < m$. Write $\gamma^i = \gamma_{i,0} + \theta\gamma_{i,1}$. Then $\beta^d \gamma^{-u_0} = \gamma^{du_1}$ and so $(f_{d,0}(a) + \theta f_{d,1}(a))(\gamma_{-u_0,0} + \theta\gamma_{-u_0,1}) = (a^2+t)^d(\gamma_{du_1,0} + \theta\gamma_{du_1,1})$. Hence

$$\left(g^{f_{d,0}(a)}\right)^{\gamma_{-u_0,0}} \left(g^{f_{d,1}(a)}\right)^{\gamma_{-u_0,1}} = \left(g^{(a^2+t)^d}\right)^{\gamma_{du_1,0}}$$

and similarly for the implicit representation of the coefficient of θ. It follows that one can perform the baby-step-giant-step algorithm in this setting to compute (u_0, u_1) and hence $u \pmod{(r+1)/d}$. Note that computing $g^{f_{d,0}(a)}, g^{f_{d,1}(a)}$ and $g^{(a^2+t)^d}$ requires $6d$ exponentiations. The stated complexity follows.

For the second stage, we have $\beta = \gamma^{u+v(r+1)/d}$ where $0 \le v < d$. Giving a baby-step-giant-step algorithm here is straightforward and we leave the details as an exercise. $\square$

One derives the following result. Note that it is not usually practical to consider a computational problem whose input is a $O(r^{1/3})$-tuple of group elements, hence this result is mainly of theoretical interest.

Corollary 21.5.11. *Let g have prime order r and suppose $r+1$ has a factor d such that $d \approx r^{1/3}$. Given $h_i = g^{a^i}$ for $1 \le i \le 2d$ then one can compute a in $O(r^{1/3}\log(r))$ group operations.*

Corollary 21.5.12. *Let g have prime order r and suppose $r+1$ has a factor d such that $d \approx r^{1/3}$. Given $h = g^a$ and a perfect Static-DH oracle with respect to (g,h) then one can compute a in $O(r^{1/3})$ oracle queries and $O(r^{1/3}\log(r))$ group operations.*

Exercise 21.5.13. Fill in the missing details in the proof of Theorem 21.5.10 and prove Corollaries 21.5.11 and 21.5.12.

Satoh [511] extends Cheon's algorithm to algebraic groups of order $\varphi_n(r)$ (essentially, to the groups $G_{n,r}$). He also improves Theorem 21.5.10 in the case of $d \mid (r+1)$ to only require $h_i = g^{a^i}$ for $1 \le i \le d$.

A natural problem is to generalise Theorem 21.5.10 to other algebraic groups, such as elliptic curves. The obvious approach does not seem to work (see Remark 1 of [131]), so it seems a new idea is needed to achieve this. Finally, Section 5.2 of [132] shows that, at least asymptotically, most primes r are such that $r-1$ or $r+1$ has a useful divisor.

Both [111] and [131] remark that a decryption oracle for classic textbook Elgamal leads to an Static-DH oracle: Given an Elgamal public key (g, g^a) and any $h_1 \in \langle g \rangle$ one can ask for the decryption of the ciphertext $(c_1, c_2) = (h_1, 1)$ (one can also make this less obvious using random self-reducibility of Elgamal ciphertexts) to get $c_2 c_1^{-a} = h_1^{-a}$. From this one computes h_1^a. By performing this repeatedly one can compute a sequence $h_i = g^{a^i}$ as required. The papers [111, 131] contain further examples of cryptosystems that provide Static-DH oracles, or computational assumptions that contain values of the form $h_i = g^{a^i}$.

21.6 Hard Bits of Discrete Logarithms

Saying that a computational problem is hard is the same as saying that it is hard to write down a binary representation of the answer. Some bits of a representation of the answer may be easy to compute (at least, up to a small probability of error) but if a computational problem is hard then there must be at least one bit of any representation of the answer that is hard to compute. In some cryptographic applications (such as key derivation or designing secure pseudorandom generators) it is important to be able to locate some of these "hard bits". Hence, the main challenge is to prove that a specific bit is hard. A potentially easier problem is to determine a small set of bits, at least one of which is hard. A harder problem is to prove that some set of bits are all simultaneously hard (for this concept see Definition 21.6.14).

The aim of this section is to give a rigorous definition for the concept of "hard bits" and to give some easy examples (hard bits of the solution to the DLP). In Section 21.7 we will consider related problems for the CDH problem. We first show that certain individual bits of the DLP, for any group, are as hard to compute as the whole solution.

Definition 21.6.1. Let $g \in G$ have prime order r. The computational problem **DL-LSB** is: given (g, g^a) where $0 \le a < r$ to compute the least significant bit of a.

Exercise 21.6.2. Show that DL-LSB $\le_R$ DLP.

Theorem 21.6.3. *Let G be a group of prime order r. Then DLP $\le_R$ DL-LSB.*

Proof: Let A be a perfect oracle that, on input (g, g^a) outputs the least significant bit of $0 \le a < r$. In other words, if the binary expansion of a is $\sum_{i=0}^{m} a_i 2^i$ then A outputs a_0. We will use A to compute a.

The first step is to call $A(g, h)$ to get a_0. Once this has been obtained we set $h' = hg^{-a_0}$. Then $h' = g^{2a_1+4a_2+\cdots}$. Let $u = 2^{-1} = (r+1)/2 \pmod{r}$ and define

$$h_1 = (h')^u.$$

Then $h_1 = g^{a_1+2a_2+\cdots}$ so calling $A(g, h_1)$ gives a_1. For $i = 2, 3, \ldots$ compute $h_i = (h_{i-1}g^{-a_{i-1}})^u$ and $a_i = A(g, h_i)$, which computes the binary expansion of a. This reduction runs in polynomial-time and requires polynomially many calls to the oracle A. $\square$

Exercise 21.6.4. Give an alternative proof of Theorem 21.6.3 based on bounding the unknown a in the range

$$(l-1)r/2^j \le a < lr/2^j.$$

Initially one sets $l = 1$ and $j = 0$. At step j, if one has $(l-1)r/2^j \le a < lr/2^j$ and if a is even then $(l-1)r/2^{j+1} \le a/2 < lr/2^{j+1}$ and if a is odd then $(2^j + l - 1)r/2^{j+1} \le (a+r)/2 < (2^j + l)r/2^{j+1}$. Show that when $j = \lceil \log_2(r) \rceil$ one can compute $2^{-j}a \pmod{r}$ exactly and hence deduce a.

Exercise 21.6.5. Since one can correctly guess the least significant bit of the DLP with probability 1/2, why does Theorem 21.6.3 not prove that DLP is easy?

One should also consider the case of a DL-LSB oracle that only works with some noticeable probability ϵ. It is then necessary to randomise the calls to the oracle, but the problem is to determine the LSB of a given the LSBs of some algebraically related values. The trick is to guess some $u = O(\log(1/\epsilon)) = O(\log(\log(r)))$ most significant bits of a and set them to zero (i.e., replace h by $h' = g^{a'}$ where the u most significant bits of a' are zero). One can then call the oracle on $h'g^y$ for random $0 \le y \le r - r/2^u$ and take a majority vote to get the result. For details of the argument see Blum and Micali [73].

We conclude that computing the LSB of the DLP is as hard as computing the whole DLP. Such bits are called **hardcore bits** since if DLP is hard then computing the LSB of the DLP is hard.

Definition 21.6.6. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function computable in polynomial-time (i.e., there is some polynomial $p(n)$ such that for $x \in \{0,1\}^n$ one can compute $f(x)$ in at most $p(n)$ bit operations). A function $b : \{0,1\}^* \to \{0,1\}$ is a **hardcore bit** or **hardcore predicate** for f if, for all probabilistic polynomial-time algorithms A, the **advantage**

$$\mathrm{Adv}_{x \in \{0,1\}^n}\big(A(f(x)) = b(x)\big)$$

is negligible as a function of n.

We now give some candidate hardcore predicates for the DLP. We also restate the meaning of hardcore bit for functions defined on $\{0,1,\dots,r-1\}$ rather than $\{0,1\}^*$.

Definition 21.6.7. For all $n \in \mathbb{N}$ let (G_n, g_n, r_n) be such that G_n is a group and $g_n \in G_n$ is an element of order r_n where r_n is an n-bit prime. We call this a **family of groups**. For $n \in \mathbb{N}$ define the function $f_n : \{0,1,\dots,r_n-1\} \to G_n$ by $f_n(a) = g_n^a$. For $n \in \mathbb{N}$ define $i(n) \in \{0,1,\dots,n-1\}$. The predicate $b_{i(n)} : \{0,1,\dots,r_n-1\} \to \{0,1\}$ is defined so that $b_{i(n)}(a)$ is bit $i(n)$ of a, when a is represented as an n-bit string. Then $b_{i(n)}$ is a **hardcore predicate for the DLP** (alternatively, bit $i(n)$ is a **hardcore bit for the DLP**) if, for all probabilistic polynomial-time algorithms A, the advantage

$$\mathrm{Adv}_{a \in \{0,1,\dots,r_n-1\}}\big(A(f_n(a)) = b_{i(n)}(a)\big)$$

is negligible as a function of n.

The least significant bit (LSB) is the case $i(n) = 0$ in the above definition. If the DLP is hard then Theorem 21.6.3 shows that the LSB is a hardcore bit.

Example 21.6.8. Fix $m \in \mathbb{N}$. Let g have prime order $r > 2^m$. Suppose A is a perfect oracle such that, for $x \in \{0,1,\dots,r-1\}$, $A(g^x)$ is the predicate $b_m(x)$ (i.e., bit m of x). One can use A to solve the DLP by guessing the $m-1$ LSBs of x and then using essentially the same argument as Theorem 21.6.3. Hence, if m is fixed and g varies in a family of groups as in Example 21.6.7 then $b_m(x)$ is a hardcore predicate for the DLP. A similar result holds if m is allowed to grow, but is bounded as $m = O(\log(\log(r)))$.

We now give an example of a hardcore predicate that is not just a bit of the DLP.

Exercise 21.6.9. Let g have prime order r. Let $f : \{0,1,\dots,r-1\} \to G$ be $f(x) = g^x$. Define the predicate $b : \{0,1,\dots,r-1\} \to \{0,1\}$ by $b(x) = x_1 \oplus x_0$ where x_0 and x_1 are the two least significant bits of x. Show that b is a hardcore predicate for f.

It is not true that any bit of the DLP is necessarily hardcore. For example, one can consider the most significant bit of a, which is $b_{n-1}(x)$ in Definition 21.6.7.

Example 21.6.10. Let $r = 2^l + u$ be a prime where $0 < u < 2^{l-\kappa}$. Let $0 \le a < r$ be chosen uniformly at random and interpreted as an $(l+1)$-bit string. Then the most significant bit of a is equal to 1 with probability $u/r < u/2^l < 1/2^\kappa$ and is equal to 0 with probability at least $1 - 1/2^\kappa$. Hence, when $\kappa \le 1$ then the most significant bit is not a hardcore bit for the DLP. Note that the function g^a is not used here; the result merely follows from the distribution of integers modulo r.

Exercise 21.6.11. Let $r = 2^l + 2^{l-1} + u$ where $0 < u < 2^{l/2}$. Let $0 \le a < r$ be uniformly chosen and represented as an $(l+1)$-bit string. Show that neither the most significant bit (i.e., bit l) nor bit $l-1$ of a are hardcore for the DLP.

The above examples show that for some primes the most significant bit is easy to predict. For other primes the most significant bit can be hard.

Exercise 21.6.12. Suppose $r = 2^l - 1$ is a Mersenne prime and let g have order r. Fix $0 \le i \le l$. Show that if $O(g, h)$ is a perfect oracle that returns the i-th bit of the DLP of h with respect to g then one can compute the whole DLP.

To summarise, low order bits of the DLP are always as hard as the DLP, while high order bits may or may not be hard. However, our examples of cases where the high order bits are easy are due not to any weakness of the DLP, but rather to statistical properties of residues modulo r. One way to deal with this issue is to define a bit as being "hard" if it cannot be predicted better than the natural statistical bias (see, for example, Definition 6.1 of Håstad and Näslund [279]). However this approach is less satisfactory for cryptographic applications if one wants to use the DLP as a source of unpredictable bits. Hence, it is natural to introduce a more statistically balanced predicate to use in place of high order bits. In practice, it is often more efficient to compute the least significant bit than to evaluate this predicate.

Exercise 21.6.13. Let g have order r. Let $f : \{0, 1, \dots, r-1\} \to G$ be $f(x) = g^x$. Define $b(x) = 0$ if $0 \le x < r/2$ and $b(x) = 1$ if $r/2 \le x < r$. Show, using the method of Exercise 21.6.4, that $b(x)$ is a hardcore bit for f.

We do not cover all results on hard bits for the DLP. See Section 9 of Håstad and Näslund [279] for a general result and further references.

So far we only discussed showing that single bits of the DLP are hard. There are several approaches to defining the notion of a set of k bits being simultaneously hard. One definition states that the bits are hard if, for every non-constant function $B : \{0, 1\}^k \to \{0, 1\}$, given an oracle that takes as input g^x and computes B on the k bits of x in question one can use the oracle to solve the DLP. Another definition, which seems to be more useful in practice, is in terms of distinguishing the bits from random.

Definition 21.6.14. Let $f : \{0, 1\}^n \to \{0, 1\}^m$ be a one way function and let $S \subset \{1, \dots, n\}$. We say the bits labelled by S are **simultaneously hard** if there is no polynomial-time algorithm that given $f(x)$ can distinguish the sequence $(x_i : i \in S)$ from a random $\#S$-bit string.

Peralta [480] (using next-bit-predictability instead of hardcore predicates or Definition 21.6.14) proves that $O(\log(\log(r)))$ least significant bits of the DLP are hard. Schnorr [524] (using Definition 21.6.14) proves that essentially any $O(\log(\log(r)))$ bits of the DLP are simultaneously hard (using the "bits" of Exercise 21.6.13 for the most significant bits).

Patel and Sundaram [478] showed, under a stronger assumption, that many more bits are simultaneously hard. Let g be an element of prime order r, let $l \in \mathbb{N}$ and set $k = \lceil \log_2(r) \rceil - l$. The ideas of Patel and Sundaram lead to the following result. If, given g^x, the k least significant bits of x are not simultaneously hard then there is an efficient algorithm to solve the DLP in an interval of length 2^l (see Exercise 13.3.6 for the definition of this problem). Hence, under the assumption that the DLP in an interval of length 2^l is hard, then one can output many bits. Taking $l = \log(\log(p))^{1+\epsilon}$ gives an essentially optimal asymptotic bit security result for the DLP.

21.6.1 Hard Bits for DLP in Algebraic Group Quotients

One can consider hard bits for the DLP in algebraic group quotients. In other words, let O_i be a perfect oracle that on input the equivalence class of an element $[g^a]$ outputs bit i of a. The first problem is that there is more than one value a for each class $[g^a]$ and so the bit is not necessarily well-defined.

Section 7 of Li, Näslund and Shparlinski [387] considers this problem for LUC. To make the problem well-defined they consider an element $g \in \mathbb{F}_{p^2}$ of prime order r and an oracle A such that $A(t) = a_i$ where a_i is the i-th bit of a for the unique $0 \le a < r/2$ such that $t = \mathrm{Tr}_{\mathbb{F}_{p^2}/\mathbb{F}_p}(g^a)$. The idea of their method is, given t, to compute the two roots $h_1 = g^a$ and $h_2 = g^{r-a}$ of $X^2 - tX + 1$ in $\mathbb{F}_{p^2}$ then use previous methods (e.g., Theorem 21.6.3 or Exercise 21.6.4) on each of them to compute either a or $r-a$ (whichever is smaller).

Exercise 21.6.15. Work out the details of the Li, Näslund and Shparlinski result for the case of the least significant bit of the DLP in LUC.

Exercise 21.6.16. Consider the algebraic group quotient corresponding to elliptic curve arithmetic using x-coordinates only. Fix $P \in E(\mathbb{F}_q)$ of prime order r. Let A be an oracle that on input $u \in \mathbb{F}_q$ outputs a_0 where a_0 is the 0-th bit of a such that $0 \le a < r/2$ and $x([a]P) = u$. Show that the method of Li, Näslund and Shparlinski can be applied to show that this bit is a hard bit for the DLP.

Li, Näslund and Shparlinski remark that it seems to be hard to obtain a similar result for XTR. Theorem 3 of Jiang, Xu and Wang [315] claims to be such a result, but it does not seem to be proved their paper.

21.7 Bit Security of Diffie-Hellman

We now consider which bits of the CDH problem are hard. Since the solution to a CDH instance is a group element it is natural to expect, in contrast with our discussion of the DLP, that the hardcore bits and the proof techniques will depend on which group is being studied.

We first consider the case $g \in \mathbb{F}_p^*$ where p is a large prime and g is a primitive root. Our presentation follows Boneh and Venkatesan [85]. We assume every element $x \in \mathbb{F}_p^*$ is represented as an element of the set $\{1, 2, \ldots, p-1\}$ and we interpret $x \pmod{p}$ as returning a value in this set.

Definition 21.7.1. Let p be odd. Let $x \in \{1, 2, \ldots, p-1\}$. Define

$$\mathrm{MSB}_1(x) = \begin{cases} 0 & \text{if } 1 \le x < p/2 \\ 1 & \text{otherwise.} \end{cases}$$

For $k \in \mathbb{N}$ let $0 \le t < 2^k$ be the integer such that

$$tp/2^k \le x < (t+1)p/2^k$$

and define $\mathrm{MSB}_k(x) = t$.

An alternative definition, which is commonly used in the literature and sometimes used in this book, is $\mathrm{MSB}_k(x) = u \in \mathbb{Z}$ such that $|x - u| \le p/2^{k+1}$ (e.g., $u = \lfloor tp/2^k + p/2^{k+1} \rfloor$). For this definition it is unnecessary to assume $k \in \mathbb{N}$ and so one can allow $k \in \mathbb{R}_{>0}$.

Note that these are not bits of the binary representation of x. Instead, as in Exercise 21.6.13, they correspond to membership of x in a certain partition of $\{1, 2, \ldots, p-1\}$.

Ideally we would like to show that, say, MSB_1 is a hardcore bit for CDH. This seems to be out of reach for $\mathbb{F}_p^*$. Instead, we will show that, for $k \approx \sqrt{\log_2(r)}$, if one can compute $\mathrm{MSB}_k(g^{ab} \pmod{p})$ then one can compute $g^{ab} \pmod{p}$. A consequence of this result is that there exists some predicate defined on $\mathrm{MSB}_k(g^{ab} \pmod{p})$ whose value is a hardcore bit for CDH.

The central idea of most results on the bit security of CDH is the following. Let p be an odd prime and let $g \in \mathbb{F}_p^*$ be a primitive root. Let $h_1 = g^a, h_2 = g^b$ be a CDH instance where b is coprime to $p-1$. For $k \in \mathbb{N}$ let A_k be a perfect oracle such that

$$A_k(g, g^a, g^b) = \mathrm{MSB}_k(g^{ab}).$$

Choose a random element $1 \le x < p$ and set $u = A_k(g, h_1 g^x, h_2)$. One has

$$u = \mathrm{MSB}_k(g^{(a+x)b}) = \mathrm{MSB}_k(g^{ab}t) \quad \text{where} \quad t = h_2^x.$$

In other words, the oracle A_k gives the most significant bits of multiples of the unknown g^{ab} by uniformly random elements $t \in \mathbb{F}_p^*$. The problem of using this information to compute g^{ab} is (a special case of) the hidden number problem.

21.7.1 The Hidden Number Problem

Definition 21.7.2. Let p be an odd prime and $k \in \mathbb{R}_{>1}$. Let $\alpha \in \mathbb{F}_p^*$ and let $t_1, \dots, t_n \in \mathbb{F}_p^*$ be chosen uniformly at random. The **hidden number problem** (**HNP**) is, given $(t_i, u_i = \mathrm{MSB}_k(\alpha t_i \pmod{p}))$ for $1 \le i \le n$ to compute α.

Throughout this section we will allow any $k \in \mathbb{R}_{>1}$ and define $\mathrm{MSB}_k(x)$ to be any integer u such that $|x - u| < p/2^{k+1}$.

Before giving the main results we discuss two easy variants of Definition 21.7.2 where the values t_i can be chosen adaptively.

Lemma 21.7.3. *Let p be an odd prime and $1 \le \alpha < p$. Suppose one has a perfect oracle A_1 such that $A_1(t) = \mathrm{MSB}_1(\alpha t \pmod{p})$. Then one can compute α using $O(\log(p))$ oracle queries.*

Exercise 21.7.4. Prove Lemma 21.7.3.

Lemma 21.7.5. *Let p be an odd prime and $1 \le \alpha < p$. Suppose one has a perfect oracle A such that $A(t) = \mathrm{LSB}_1(\alpha t \pmod{p})$, where $\mathrm{LSB}_1(x)$ is the least significant bit of the binary representation of $0 \le x < p$. Then one can compute α using $O(\log_2(p))$ oracle queries.*

Exercise 21.7.6. Prove Lemma 21.7.5.

Lemmas 21.7.3 and 21.7.5 show that the hidden number problem can be easy if the values t_i in Definition 21.7.2 are chosen adaptively. However, it intuitively seems harder to solve the hidden number problem when the t_i are randomly chosen. On the other hand, as k grows the HNP becomes easier; the case $k = \log_2(p)$ being trivial. Hence, one could hope to be able to solve the HNP as long as k is sufficiently large. We now explain the method of Boneh and Venkatesan [85] to solve the HNP using lattices.

Definition 21.7.7. Let $(t_i, u_i = \mathrm{MSB}_k(\alpha t_i))$ for $1 \le i \le n$. Define a lattice $L \subseteq \mathbb{R}^{n+1}$ by the rows of the basis matrix

$$B = \begin{pmatrix} p & 0 & 0 & \cdots & 0 & 0 \\ 0 & p & 0 & & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \\ 0 & 0 & 0 & \cdots & p & 0 \\ t_1 & t_2 & t_3 & \cdots & t_n & 1/2^{k+1} \end{pmatrix}.$$

Define the vector $\underline{u} = (u_1, u_2, \dots, u_n, 0) \in \mathbb{R}^{n+1}$ where $|u_i - (\alpha t_i \pmod{p})| < p/2^{k+1}$.

Lemma 21.7.8. *Let L, $\underline{u}$ and n be as in Definition 21.7.7. Then* $\det(L) = p^n/2^{k+1}$ *and there exists a vector $\underline{v} \in L$ such that $\|\underline{u} - \underline{v}\| < \sqrt{n+1}p/2^{k+1}$.*

Proof: The first statement is trivial. For the second, note that $u_i = \mathrm{MSB}_k(\alpha t_i \pmod{p})$ is the same as saying $\alpha t_i = u_i + \epsilon_i + l_i p$ for some $\epsilon_i, l_i \in \mathbb{Z}$ such that $|\epsilon_i| \le p/2^{k+1}$, for $1 \le i \le n$. Now define $\underline{v} \in L$ by

$$\begin{aligned}\underline{v} = (-l_1, -l_2, \ldots, -l_n, \alpha)B &= (\alpha t_1 - l_1 p, \ldots, \alpha t_n - l_n p, \alpha/2^{k+1}) \\ &= (u_1 + \epsilon_1, \ldots, u_n + \epsilon_n, \alpha/2^{k+1}).\end{aligned}$$

The result follows since $\alpha/2^{k+1} < p/2^{k+1}$. □

We now show that, for certain parameters, it is reasonable to expect that any vector in the lattice L that is close to $\underline{u}$ gives the solution α.

Theorem 21.7.9. *Let $p > 2^8$ be prime and let $\alpha \in \mathbb{F}_p^*$. Let $n = 2\lceil\sqrt{\log_2(p)}\rceil \in \mathbb{N}$ and let $k \in \mathbb{R}$ be such that $\log_2(p) - 1 \ge k > \mu = \frac{1}{2}\sqrt{\log_2(p)} + 3$. Suppose $t_1, \ldots, t_n$ are chosen uniformly and independently at random in $\mathbb{F}_p^*$ and set $u_i = \mathrm{MSB}_k(\alpha t_i)$ for $1 \le i \le n$. Construct the lattice L as above. Let $\underline{u} = (u_1, \ldots, u_n, 0)$. Then, with probability at least $1 - 1/2^n \ge 63/64$ over all choices for $t_1, \ldots, t_n$, any vector $\underline{v} \in L$ such that $\|\underline{v} - \underline{u}\| < p/2^{\mu+1}$ is of the form*

$$\underline{v} = (\beta t_1 \pmod{p}, \ldots, \beta t_n \pmod{p}, \beta/2^{k+1})$$

where $\beta \equiv \alpha \pmod{p}$.

Proof: In the first half of the proof we consider $t_1, \ldots, t_n$ as fixed values. Later in the proof we compute a probability over all choices for the t_i.

First, note that every vector in the lattice is of the form

$$\underline{v} = (\beta t_1 - l_1 p, \beta t_2 - l_2 p, \ldots, \beta t_n - l_n p, \beta/2^{k+1})$$

for some $\beta, l_1, \ldots, l_n \in \mathbb{Z}$. If $\beta \equiv \alpha \pmod{p}$ then we are done, so suppose now that $\beta \not\equiv \alpha \pmod{p}$. Suppose also that $\|\underline{v} - \underline{u}\| < p/2^{\mu+1}$, which implies $|(\beta t_i \pmod{p}) - u_i| < p/2^{\mu+1}$ for all $1 \le i \le n$. Note that

$$\begin{aligned}|(\beta - \alpha)t_i \pmod{p}| &= |(\beta t_i \pmod{p}) - u_i + u_i - (\alpha t_i \pmod{p})| \\ &\le |(\beta t_i \pmod{p}) - u_i| + |(\alpha t_i \pmod{p}) - u_i| \\ &< p/2^{\mu+1} + p/2^{\mu+1} = p/2^\mu.\end{aligned}$$

We now consider $\gamma = (\beta - \alpha)$ as a fixed non-zero element of $\mathbb{F}_p$ and denote by A the probability, over all $t \in \mathbb{F}_p^*$, that $\gamma t \equiv u \pmod{p}$ for some $u \in \mathbb{Z}$ such that $|u| < p/2^\mu$ and $u \ne 0$. Since γt is uniformly distributed over $\mathbb{F}_p^*$ it follows that

$$A \le \frac{2(p/2^\mu)}{p-1} \le \frac{1}{p-1}\left(\frac{2(p-1)+2}{2^\mu}\right) < \frac{2}{2^\mu} + \frac{2}{p-1} < \frac{4}{2^\mu}.$$

Since there are n uniformly and independently chosen $t_1, \ldots, t_n \in \mathbb{F}_p^*$ the probability that $|\gamma t_i \pmod{p}| < p/2^\mu$ for all $1 \le i \le n$ is A^n. Finally, there are $p - 1$ choices for $\beta \in \{0, 1, \ldots, p-1\}$ such that $\beta \not\equiv \alpha \pmod{p}$. Hence, the probability over all such β and all $t_1, \ldots, t_n$ that $\|\underline{v} - \underline{u}\| < p/2^{\mu+1}$ is at most

$$(p-1)A^n < \frac{(p-1)4^n}{2^{\mu n}} < \frac{2^{\log_2(p)+2n}}{2^{\mu n}}.$$

Now, $\mu n = (\frac{1}{2}\sqrt{\log_2(p)} + 3)2\lceil\sqrt{\log_2(p)}\rceil \ge \log_2(p) + 3n$ so $(p-1)A^n < 2^{-n}$. Since $n \ge 6$ the result follows. □

Corollary 21.7.10. *Let $p > 2^{32}$ be prime, let $n = 2\lceil\sqrt{\log_2(p)}\rceil$ and let $k = \lceil\sqrt{\log_2(p)}\rceil + \lceil\log_2(\log_2(p))\rceil$. Given $(t_i, u_i = \mathrm{MSB}_k(\alpha t_i))$ for $1 \le i \le n$ as in Definition 21.7.2 one can compute α in polynomial-time.*

Proof: One constructs the basis matrix B for the lattice L in polynomial-time. Note that $n = O(\sqrt{\log(p)})$ so that the matrix requires $O(\log(p)^2)$ bits storage.

Running the LLL algorithm with factor $\delta = 1/4 + 1/\sqrt{2}$ is a polynomial-time computation (the lattice is not a subset of $\mathbb{Z}^{n+1}$ so Remark 17.5.5 should be applied, noting that only one column has non-integer entries) which returns an LLL-reduced basis. Let $\underline{u}$ be as above. The Babai nearest plane algorithm finds $\underline{v}$ such that $\|\underline{v} - \underline{u}\| < (1.6)2^{(n+1)/4}\sqrt{n+1}p/2^{k+1}$ by Theorem 18.1.7 and Lemma 21.7.8. This computation requires $O(\log(p)^{4.5})$ bit operations by Exercise 18.1.9. To apply Theorem 21.7.9 we need the vector $\underline{v}$ output from the Babai algorithm to be within $p/2^{\mu+1}$ of $\underline{u}$ where $\mu = \frac{1}{2}\sqrt{\log_2(p)} + 3$. Hence, we need

$$\frac{(1.6)2^{(n+1)/4}\sqrt{n+1}}{2^{k+1}} < \frac{1}{2^{\mu+1}},$$

which is $\mu + \log_2(1.6) + (n+1)/4 + \log_2(\sqrt{n+1}) < k = \lceil\sqrt{\log_2(p)}\rceil + \lceil\log_2(\log_2(p))\rceil$. Since

$$\begin{aligned}\mu + \log_2(1.6) + (n+1)/4 + \log_2(\sqrt{n+1}) &= \sqrt{\log_2(p)}/2 + 3.95 + \lceil\sqrt{\log_2(p)}\rceil/2 + \tfrac{1}{2}\log_2(n+1)\\ &\le \lceil\sqrt{\log_2(p)}\rceil + 3.95 + \tfrac{1}{2}\log_2(n+1)\end{aligned}$$

the result follows whenever p is sufficiently large (the reader can check that $p > 2^{32}$ is sufficient).

It follows from Theorem 21.7.9 that, with probability at least 63/64 the vector $\underline{v} = (v_1, \dots, v_{n+1}) \in \mathbb{R}^{n+1}$ output by the Babai algorithm is such that $v_{n+1}2^{k+1} \equiv \alpha \pmod{p}$. It follows that the hidden number α can be efficiently computed. □

Note that if $p \approx 2^{160}$ then $\mu \approx 9.32$. In practice, the algorithm works well for primes of this size. For example, Howgrave-Graham and Smart [299] present results of practical experiments where 8 of the most significant bits are provided by an oracle. We stress that these results do not show that all of the $k = \lceil\sqrt{\log_2(p)}\rceil + \lceil\log_2(\log_2(p))\rceil$ most significant bits are hard. Instead, one can only deduce that there is a predicate defined on these k bits that is a hardcore predicate for CDH.

Nguyen and Shparlinski [459] also remark that one could use other methods than LLL and the Babai nearest plane algorithm. They show that if one uses the Ajtai, Kumar and Sivakumar algorithm for CVP then one only needs $k = \lfloor\log(\log(p))\rfloor$ bits to obtain an algorithm for the hidden number problem with complexity of $p^{O(1/\log(\log(p)))}$ bit operations. They further show that if one has a perfect oracle for CVP (with respect to the ℓ_∞ norm) then one can solve the hidden number problem in polynomial time given only $k = 1 + \epsilon$ bits for any $\epsilon > 0$.

One final remark, the methods in this section assume a perfect oracle that outputs $\mathrm{MSB}_1(\alpha t \pmod{p})$. Since there seems to be no way to determine whether the output of the oracle is correct, it is an open problem to get results in the presence of an oracle that sometimes makes mistakes (though, as we mention in the next section, when applying the hidden number problem to the bit security of CDH then there is a solution in the case of oracles with a relatively low probability of giving an incorrect answer). For further discussion and applications of the hidden number problem see Shparlinski [559].

21.7.2 Hard Bits for CDH Modulo a Prime

We can finally state a result about hard bits for CDH.

Theorem 21.7.11. *Let $p > 2^{32}$ be prime, let g be a primitive root modulo p and let $k = \lceil \sqrt{\log_2(p)} \rceil + \lceil \log_2(\log_2(p)) \rceil$. Suppose there is no polynomial-time algorithm to solve*[10] *CDH in $\mathbb{F}_p^*$. Then there is no polynomial-time algorithm to compute the k most significant bits of g^{ab} when given g, g^a and g^b.*

Proof: Let (g, g^a, g^b) be an instance of the CDH problem in $\langle g \rangle$ and write $\alpha = g^{ab}$ for the solution. We assume that $\gcd(b, p-1) = 1$ (this requirement is removed by González Vasco and Shparlinski [261]; other work mentioned below allows g to have prime order, in which case this restriction disappears).

Given a polynomial-time algorithm A such that $A(g, g^x, g^y) = \mathrm{MSB}_k(g^{xy} \pmod{p})$ then one can call $A(g, g^a g^r, g^b)$ polynomially many times for uniformly random $r \in \{1, 2, \dots, p-2\}$ to get $\mathrm{MSB}_k(\alpha t)$ where $t = g^{br} \pmod{p}$. Applying Corollary 21.7.10 gives a polynomial time algorithm to compute α. $\square$

A number of significant open problems remain:

1. Theorem 21.7.11 shows it is hard to compute all of $\mathrm{MSB}_k(g^{ab})$ but that does not imply that, say, $\mathrm{MSB}_1(g^{ab})$ is hard. A stronger result would be to determine specific hardcore bits for CDH, or at least to extend the results to MSB_k for smaller values of k. Boneh and Venkatesan [86] give a method that works for $k = \lceil 2\log(\log(p)) \rceil$ bits (where g is a primitive root in $\mathbb{F}_p^*$) but which needs a hint depending on p and g; they claim this is a non-uniform result but this depends on the instance generator (see the footnote of Section 21.4.3). For $k = \lfloor \log(\log(p)) \rfloor$ one can also consider the approach of Nguyen and Shparlinski [459] mentioned above.

   Akavia [8] uses a totally different approach to prove that MSB_1 is hard for CDH, but the method is again at best non-uniform (i.e., needs polynomial-sized auxiliary information depending on p and g^b).

2. We assumed perfect oracles for computing $\mathrm{MSB}_k(\alpha t)$ in the above results. For non-perfect oracles one can use the above methods to generate a list of candidate values for g^{ab} and then apply the CDH self-corrector of Section 21.3. We refer to González Vasco, Näslund and Shparlinski [260] for details.

   The method of Akavia [8] also works when the oracle for MSB_1 is unreliable.

3. The above results assumed that g is a primitive root modulo p, whereas in practice one chooses g to lie in a small subgroup of $\mathbb{F}_p^*$ of prime order. The proof of Theorem 21.7.11 generates values t that lie in $\langle g \rangle$ and so they are not uniformly at random in $\mathbb{F}_p^*$. González Vasco and Shparlinski have given results that apply when the order of g is less than $p-1$ (see Chapter 14 of [558] for details and references). Shparlinski and Winterhof [560, 561], building on work of Bourgain and Konyagin, have obtained results when the order of g is at least $\log(p)/\log(\log(p))^{1-\epsilon}$.

Exercise 21.7.12. This exercise concerns a static Diffie-Hellman key exchange protocol due to Boneh and Venkatesan [85] for which one can prove that the most significant bit is a hardcore bit. Suppose Alice chooses a prime p, an integer $1 \le a < p-1$ such that $\gcd(a, p-1) = 1$ and sets $g = 2^{a^{-1} \pmod{p-1}} \pmod{p}$. Alice makes p and g public and keeps a private. When Bob wants to communicate with Alice he sends g^x for random $1 \le x < p-1$ so that Alice and Bob share the key 2^x. Prove that $\mathrm{MSB}_1(2^x)$ is a hardcore bit.

[10] As we have seen, to make such a statement precise one needs an instance generator that outputs groups from a family.

[Hint: Suppose one has a perfect oracle A that on input g^y outputs $\mathrm{MSB}_1(2^y)$. Then one can store Bob's tranmission g^x and call $A(g^x g^y)$ to get $\alpha 2^y$, where $\alpha = 2^x$ is the desired hidden number. Then apply Lemma 21.7.3.]

Exercise 21.7.13. Let $g \in \mathbb{F}_p^*$ be a primitive root and let $\epsilon > 0$. Show that if one has a perfect oracle for $\mathrm{MSB}_{1+\epsilon}(g^{ab})$ then one can solve DDH in $\mathbb{F}_p^*$.

21.7.3 Hard Bits for CDH in Other Groups

So far we have only considered CDH in (subgroups of) $\mathbb{F}_p^*$ where p is prime. It is natural to consider CDH in subgroups of $\mathbb{F}_{p^m}^*$, in algebraic tori, in trace systems such as LUC and XTR, and in elliptic curves. The first issue is what is meant by "bits" of such a value. In practice, elements in such a group are represented as an n-tuple of elements in $\mathbb{F}_p$ and so it is natural to consider one component in $\mathbb{F}_p$ and take bits of it as done previously. When p is small one can consider a sequence of bits, each from different components. An early reference for bit security of CDH in this setting is Verheul [619].

It is possible to extend the results to traces relatively easily. The idea is that if $\{\theta_1, \ldots, \theta_m\}$ is a basis for $\mathbb{F}_{p^m}$ over $\mathbb{F}_p$, if $\alpha = \sum_{j=1}^m \alpha_j \theta_j$ is hidden and if $t_i = \sum_{j=1}^m t_{i,j}\theta_j$ are known then $\mathrm{Tr}(\alpha t_i)$ is a linear equation in the unknown α_i. Li, Näslund and Shparlinski [387] have studied the bit security of CDH in LUC and XTR. We refer to Chapters 6 and 19 of Shparlinski [558] for further details and references.

Exercise 21.7.14. Let $\mathbb{F}_{2^m}$ be represented using a normal basis and let $g \in \mathbb{F}_{2^m}^*$. Suppose one has a perfect oracle A such that $A(g, g^a, g^b)$ returns the first coefficient of the normal basis representation of g^{ab}. Show how to use A to compute g^{ab}. Hence, conclude that the first coefficient is a hardcore bit for CDH in $\mathbb{F}_{2^m}^*$.

Exercise 21.7.15. Let $\mathbb{F}_{2^m} = \mathbb{F}_2[x]/(F(x))$ and let $g \in \mathbb{F}_{2^m}^*$ have prime order $r > m$. Suppose one has a perfect oracle A such that $A(g, g^a, g^b)$ returns the constant coefficient of the polynomial basis representation of g^{ab}. Show how to use A to compute g^{ab}. Hence, conclude that the constant coefficient is a hardcore bit for CDH in $\mathbb{F}_{2^m}^*$.

Hard Bits for Elliptic Curve Diffie-Hellman

We now consider the case of elliptic curves E over $\mathbb{F}_q$. A typical way to extract bits from an elliptic curve point P is to consider the x-coordinate $x(P)$ as an element of $\mathbb{F}_q$ and then extract bits of this. It seems hard to give results for the bit security of CDH using an oracle $A(P, [a]P, [b]P) = \mathrm{MSB}_k(x([ab]P))$; the natural generalisation of the previous approach is to call $A(P, [a]P + [z]P, [b]P) = \mathrm{MSB}_k(x([ab]P + [zb]P))$ but the problem is that it is difficult to infer anything useful about $x([ab]P)$ from $x([ab]P + [zb]P)$ (similarly for least significant bits); see Jao, Jetchev and Venkatesan [309] for some results. However, Boneh and Shparlinski [84] had the insight to consider a more general oracle.

Definition 21.7.16. Let p be an odd prime and $k \in \mathbb{N}$. Let $A_{x,k}(A, B, P, [a]P, [b]P)$ be an oracle that returns $\mathrm{LSB}_k(x([ab]P))$ where $P \in E(\mathbb{F}_p)$ for the elliptic curve $E : y^2 = x^3 + Ax + B$. Similarly, let $A_{y,k}(A, B, P, [a]P, [b]P)$ be an oracle that returns $\mathrm{LSB}_k(y([ab]P))$.

The crucial idea is that, given a point $P = (x_P, y_P) \in E(\mathbb{F}_p)$ where $E : y^2 = x^3 + Ax + B$, one can consider an isomorphism $\phi(x, y) = (u^2 x, u^3 y)$ and $\phi(P) \in E'(\mathbb{F}_p)$ where $E' : Y^2 = X^3 + u^4 AX + u^6 B$. Hence, instead of randomising instances of CDH in a way analogous to that done earlier, one calls the oracle $A_{x,k}(u^4 A, u^6 B, \phi(P), \phi([a]P), \phi([b]P))$ to get $\mathrm{LSB}_k(x(\phi([ab]P))) = \mathrm{LSB}_k(u^2 x([ab]P) \pmod{p})$ where u is controlled by the attacker. This is very similar to the easy case of the hidden number problem in $\mathbb{F}_p^*$ from Lemma 21.7.5.

Lemma 21.7.17. *Suppose* $p \equiv 2 \pmod 3$. *Then* $\mathrm{LSB}_1(y([ab]P))$ *is a hardcore bit for CDH on elliptic curves over* $\mathbb{F}_p$.

Proof: We suppose $A_{y,1}$ is a perfect oracle for $\mathrm{LSB}_1(y([ab]P))$ as above. Calling

$$A_{y,1}(u^4 A, u^6 B, \phi(P), \phi([a]P), \phi([b]P))$$

gives $\mathrm{LSB}_1(u^3 y([ab]P))$. Since $\gcd(3, p-1) = 1$ it follows that cubing is a permutation of $\mathbb{F}_p^*$ and one can perform the method of Lemma 21.7.5 to compute $y([ab]P)$. Given $y([ab]P)$ there are at most 3 choices for $x([ab]P)$ and so CDH is solved with noticeable probability. $\square$

In the general case (i.e., when $p \not\equiv 2 \pmod 3$) Boneh and Shparlinski have to work harder. They use the method of Alexi, Chor, Goldreich and Schnorr [9] or the simplified version by Fischlin and Schnorr [203] to extend the idea to non-perfect oracles.[11] Once this is done, the following trick can be applied to determine $\mathrm{LSB}_1(tx([ab]P))$: when t is a square one calls the oracle for $\mathrm{LSB}_1(u^2 x([ab]P))$ on $u = \sqrt{t} \pmod p$, and when t is not a square one flips a coin. The resulting non-perfect oracle for LSB_1 therefore solves the problem. We refer to [84] for the details.

We make some remarks.

1. A nice feature of the elliptic curve results is that they are independent of the order of the point P and so work for subgroups of any size.

2. The literature does not seem to contain bit security results for CDH on elliptic curves over non-prime fields. This would be a good student project.

3. Jetchev and Venkatesan [314] use isogenies to extend the applicability of the Boneh-Shparlinski method. Their motivation is that if one has an $\mathrm{LSB}_1(x([ab]P))$ oracle that works with only small (but noticeable) probability then it is possible to have a CDH instance on an elliptic curve E for which the oracle does not work for any twist of E. By moving around the isogeny class they claim that the probability of success increases. However, it is still possible to have a CDH instance on an elliptic curve E for which the oracle does not work for any elliptic curve in the isogeny class of E.

21.8 Further Topics

There are a number of other results related to the Diffie-Hellman problem that we do not have to space to cover. For example, Coppersmith and Shparlinski considered the existence of polynomial relations between g^x, g^y and g^{xy}. Canetti, Friedlander and Shparlinski considered the distribution of Diffie-Hellman triples (g^x, g^y, g^{xy}) in G^3. We refer to [558] for a survey of these topics and references.

[11] This is why Boneh and Shparlinski consider least significant bits rather than most significant bits for their result. The technique of Alexi et al is to randomise the query $\mathrm{LSB}_1(t\alpha)$ as $\mathrm{LSB}_1(s\alpha) \oplus \mathrm{LSB}_1((t+s)\alpha)$ for suitable values s. A good student project would be to obtain an analogous result for other bits (e.g., most significant bits).