The Static Diffie-Hellman Problem

Daniel R. L. Brown
dbrown@certicom.com
Certicom Research

Robert P. Gallant
rgallant@certicom.com
Certicom Research

June 23, 2005

Abstract

In this paper we describe an algorithm for finding the discrete logarithm of an arbitrary group element Q. The algorithm requires as input a function that solves a special case of the Diffie-Hellman problem, called the static Diffie-Hellman problem for Q. Some protocols and environments provide such a function to adversaries, in which case the algorithm may be interpreted as an attack which finds a private key. The algorithm can also be interpreted as a reduction relating the hardness of computing discrete logarithms with the hardness of solving Diffie-Hellman instances.

Mathematics Subject Classification: 11Y16, 11T71, 94A60
Key words: Diffie-Hellman, Elliptic Curve Cryptography, Discrete Logarithms, Generic Algorithms, Complexity

1 Introduction

Let $\langle G \rangle$ be a cyclic group with generator G of prime order n, written additively. For any element $Q \in \langle G \rangle$ there is a unique integer $q \in [0, n-1]$ such that $Q = qG$. (The group element qG is the element obtained by adding q copies of G together. This is called scalar multiplication of G by q.) The integer q is said to be the discrete logarithm of Q to the base G. The problem of computing the discrete logarithm of X to the base G, for a randomly chosen element $X \in \langle G \rangle$, is called the discrete logarithm problem (DLP) in $\langle G \rangle$. Groups where it is computationally difficult to solve the discrete logarithm problem are important for cryptography.

Many cryptographic protocols depend more fundamentally on the computational difficulty of the Diffie-Hellman problem (DHP). In the group $\langle G \rangle$, this is the problem of computing abG, given a random pair of group elements aG and bG.

If the discrete logarithm problem is easy in a group, then the Diffie-Hellman problem is also easy in that group. Several authors have studied whether hardness of the discrete logarithm problem implies hardness of the Diffie-Hellman problem.

In Section 2 we discuss an algorithm for finding the discrete logarithm of an arbitrary element $Q \in \langle G \rangle$. The algorithm takes as input the group generator G, positive integers u, v such that $n = uv + 1$, and a function $\mathrm{SDHP}_Q()$ related to Q that we will discuss shortly. The algorithm outputs the integer $q \in [0, n-1]$ satisfying $Q = qG$.

The function SDHP_Q takes as input an arbitrary group element X and outputs the group element qX, where as above the integer q satisfies $Q = qG$. This function is said to solve the static Diffie-Hellman problem for Q (SDHP_Q). Thus the SDHP_Q function solves certain instances of the Diffie-Hellman problem, namely those instances where one of the input elements is Q. The static Diffie-Hellman problem is thus a special case of the Diffie-Hellman problem.

Some cryptographic protocols provide an SDHP_Q function to an adversary as a normal part of their operation. Such systems include the Ford-Kaliski server-assisted key generation protocol [7], basic ElGamal encryption [6], and Chaum and van Antwerpen's Undeniable Signatures [2]. Some implementations of other protocols may also provide such a function to adversaries. In such cases the adverary can use our algorithm to attack the protocol by finding the logarithm of Q, which may be a long term private key. How this vulnerability affects the security level of a system will depend on many factors, and this vulnerability is but one consideration. Such attacks are discussed in more detail in Section 3.

The algorithm of Section 2 solves discrete logarithm challenges with the aid of the function SDHP_Q associated with the logarithm challenge Q. For some groups, this algorithm finds the logarithm with a cost less than that assumed possible without the aid of the helper function. In these cases the algorithm can be interpreted as a reduction relating the difficulty of computing discrete logarithms and the difficulty of computing the SDHP_Q function. This is analagous to the work in [4, 1, 13] reducing the discrete logarithm problem to the Diffie-Hellman problem. This reduction is discussed more fully in Section 4.

We are not aware of any previous work relating the difficulty of computing the SDHP_Q function to other problems. Nor are we aware of any previous work exploiting an SDHP_Q function available in a cryptosystem to find the associated private key.

2 Using a SDHP$_Q$ function to find a logarithm

In this section, we give an algorithm that finds logarithms in the group $\langle G\rangle$. The algorithm takes as input the generator G, positive integers u, v such that the group order is $n = uv + 1$, and the function SDHP$_Q$ for an arbitrary element $Q \in \langle G\rangle$. After u evaluations of the SDHP$_Q$ function, and with further off-line computational work of about $2(\sqrt{u} + \sqrt{v})$ group scalar multiplications, the algorithm outputs the unique integer $q \in [0, n-1]$ satisfying $Q = qG$.

For $Q \in \langle G\rangle$, we define the SDHP$_Q$ function as follows.

Definition 1 (The SDHP$_Q$ function on $\langle G\rangle$). *The function SDHP$_Q$ maps any $X \in \langle G\rangle$ to the group element qX, where q is the unique integer in $[0, n-1]$ such that $Q = qG$.*

A SDHP$_Q$ function solves the static Diffie-Hellman problem for Q, which is the set of Diffie-Hellman problems where one of the parameters is fixed to be the point Q. Being able to compute the SDHP$_Q$ function for every $Q \in \langle G\rangle$ is equivalent to being able to compute the Diffie-Hellman function on $\langle G\rangle$.

The basic idea of the algorithm is to work with exponents. To find q, we instead find z, where $q = g^z$ for some primitive element (generator) g of $\mathbb{F}_n^*$. To do this, we first compute $q^u G$ by evaluating the SDHP$_Q$ function u times in an iterative fashion. We then find integers $t \in [0, u-1]$ and $y \in [0, v-1]$ such that $z = tv + y$. We accomplish this by finding discrete logarithms in subgroups of $\mathbb{F}_n^*$ of order u and v, represented as subsets of $\langle G\rangle$.

Theorem 1. *There is an algorithm that (1) takes as input the group generator G, positive integers u, v such that the group has order $n = uv + 1$, and a SDHP$_Q$ function for some $Q \in \langle G\rangle$, (2) outputs the logarithm q of Q to the base G, and (3) requires u evaluations of the SDHP$_Q$ function, at most $2(\lceil\sqrt{u}\,\rceil + \lceil\sqrt{v}\,\rceil)$ scalar multiplications in $\langle G\rangle$, and performs 10 simple arithmetic operations on numbers no larger than n.*

Proof. Let g be a generator of the multiplicative group $\mathbb{F}_n^*$. For simplicity, we omit the notation "mod n" for expressions over $\mathbb{F}_n^*$.

The case $q = 0$ is easily detected by checking that SDHP$_Q(G) = 0$ (the group identity), so henceforth we will assume that $q \in \mathbb{F}_n^*$. Because g generates $\mathbb{F}_n^*$, we have $q = g^z$ for some integer $z \in [0, n-2]$. Since

$z \in [0, n-2]$ we can write $z = tv + y$ for some $t \in [0, u-1]$ and $y \in [0, v-1]$. We now determine the values of t and y.

We find $y = z \bmod v$ as follows. Apply u iterations of SDHP_Q to G to obtain

$$E = \mathrm{SDHP}_Q^u(G) = q^u G. \tag{1}$$

Because $q^u = g^{zu} = g^{tuv+uy} = (g^u)^y$, we see q^u lies in $\langle g^u \rangle$, the order v subgroup of $\mathbb{F}_n^*$. Let $m = \lceil \sqrt{v}\, \rceil$. Because $y \in [0, v-1]$, we have $y = y_q m + y_r$ for some $0 \leqslant y_q < m$ and $0 \leqslant y_r < m$. To find y, we find the values of y_q, y_r by emulating Shanks' Baby-Step-Giant-Step (BSGS) algorithm [14] to find an element in each of the lists

$$U = \{(G, 0), (g^u G, 1), (g^{2u} G, 2), \ldots, (g^{(m-1)u} G, m-1)\} \tag{2}$$

and

$$V = \{(E, 0), (g^{-mu} E, 1), (g^{-2mu} E, 2), \ldots, (g^{-(m-1)mu} E, m-1)\}. \tag{3}$$

such that the first coordinates of the two elements are equal.[1] The second coordinate of these pairs determine i and j such that $g^{iu} G = g^{-jmu} E$. From this we have $E = g^{(i+jm)u}$. Since also $E = q^u G = g^{uy} G$, we have $y = i + jm \bmod v$.

We now find t. Compute the point

$$F = g^{-y} \mathrm{SDHP}_Q(G). \tag{4}$$

Since $\mathrm{SDHP}_Q(G) = qG = g^{tv+y} G$ we see $F = g^{tv} G = (g^v)^t G$. To find t, we once again use BSGS. Let $k = \lceil \sqrt{u}\, \rceil$ and as above find an element from each of the sets

$$Q = \{(G, 0), (g^v G, 1), (g^{2v} G, 2), \ldots, (g^{(k-1)v} G, k-1)\} \tag{5}$$

and

$$W = \{(F, 0), (g^{-kv} F, 1), (g^{-2kv} F, 2), \ldots, (g^{-(k-1)kv} F, k-1)\}, \tag{6}$$

such that the first coordinates are equal. The second coordinates provide integers h, l such that $g^{hv} G = g^{-lkv} F$. Thus $F = g^{(h+lk)v} G$. Since we also have $F = (g^v)^t G$, we have $t = h + lk \bmod u$. Finally we can now easily determine $z = tv + y$, and our desired logarithm $q = g^z$.

[1] Because y can be written as $y = y_q m + y_r$ for some $0 \leqslant y_q < m$ and $0 \leqslant y_r < m$, these elements exist.

The computations required by this algorithm are: u queries to the SDHP_Q function, at most $2\lceil \sqrt{u}\ \rceil + 2\lceil \sqrt{v}\ \rceil$ scalar multiplications and table lookups in the group $\langle G \rangle$, seven exponentiations in $\mathbb{F}_n^*$ (g^u, w^m, g^{-y}, g^v, q^k, q^t, $q = g^z$), and three further arithmetic operations ($y = i + jm$, $t = h + lk$, and $z = tv + y$). □

Other than the evaluations of the SDHP_Q function, the majority of the cost in implementing this algorithm is computing and storing the set U in (2) and the set Q in (5), followed by an iteration over each element in the set V in (3) and in the set W in (6). In other words, the cost of this algorithm is generally dominated by about $2(\sqrt{u} + \sqrt{v})$ scalar multiplications in the group $\langle G \rangle$. Henceforth, we will ignore the constant number of operations in $\mathbb{F}_n$ and the table-lookups. If we count each query to the SDHP function as a cost of one scalar multiplication, then the total cost of the algorithm is approximately $u + 2\sqrt{v}$ scalar multiplications.

In Section 4 it will be useful to know the cost of the algorithm in terms of group operations. This amounts to estimating the cost of a scalar multiplication in the $\langle G \rangle$ relative to the cost of a group operation in $\langle G \rangle$. Because so many scalar multiplications are required in the algorithm, precomputation is worthwhile. In precomputation, operations common to each scalar multiplication are performed once and stored for re-use in each scalar multiplication. For example, simply precomputing the $2 \cdot 16 \cdot \lceil \log_2(n) \rceil$ values $x2^yE$ and $x2^yG$ for each integer $y \in [0, \lfloor \log_2(n) \rfloor]$ and $x \in [0, 15]$ allows one to compute the scalar multiplication of E or G by an integer in $[0, n-1]$ in at most $\log_2(n)/4$ group operations using standard algorithms. In this case we may assume that the cost C of a scalar multiplication in $\langle G \rangle$ is approximately $\log_2(n)/4$ group operations. If extreme amounts of precomputation are possible the value of C may even be a small constant such as $C = 5$.

Like the Pollard-ρ algorithm and the BSGS algorithm, the algorithm of Theorem 1 is generic in the sense that it works in many groups: the algorithm needs only to perform group addition and inversion operations and requires that group elements have unique representations as binary strings. We use the BSGS algorithm to find q, mainly because the resulting analysis is simple. The Pollard-ρ algorithm [16] can also be used with straightforward modifications. The primary advantage of the Pollard-ρ algorithm over the BSGS algorithm is that it requires much less memory.

3 The Algorithm as an Attack

In some real world systems, an SDHP_Q oracle is available. An SDHP_Q oracle is a system entity that will compute a SDHP_Q function for an adversary. In this case, an adversary might use our algorithm to find the private key q more quickly than by using other algorithms for the discrete logarithm problem. To assess the cost of the attack, we model each SDHP oracle query as taking C group operations, representing the situation where the adversary must wait for the entity to perform a scalar multiplication when answering an SDHP oracle query.

The minimum total cost of the algorithm is achieved when $u \approx \sqrt[3]{n}$, for a total cost of about $3\sqrt[3]{n}$ scalar multiplications. For a random prime n, we expect $n-1$ to have a factorization similar to that of random integer. The largest prime factor of random integers is commonly of size $n^{\log 2} \approx n^{2/3}$, see [10, §4.5.4]. Thus taking v to be this prime factor, so $v \approx n^{2/3}$, we have $u \approx n^{1/3}$ which is roughly optimal for minimizing the total cost of the algorithm. In this case the algorithm has a cost of roughly $Cu + 2C(\sqrt{u} + \sqrt{v}) \approx 3C\sqrt[3]{n}$ group operations. If the system is based on a group where solving the discrete logarithm problem is assumed to take at least $\sqrt{n}$ group operations, then this attack may be faster than previously known attacks.

A class of groups of cryptographic interest where the discrete logarithm problem is assumed to require at least $O(\sqrt{n})$ group operations are elliptic curve groups. The fastest known algorithms for solving a general elliptic curve DLP are the generic algorithms such as BSGS and Pollard-ρ, which require $O(\sqrt{n})$ group operations. Therefore Theorem 1 may be relevant for systems using elliptic curve groups.

Elliptic curves of considerable interest are those recommended by NIST. Complete factorizations of $n-1$ for all but the largest of the NIST curves are given in Table 1. It is evident that for all of these factorizations, some prime factors can be collected to form a factor $u \approx \sqrt[3]{n}$. Furthermore, it is also evident that much smaller values of u can be found; this will be useful in Section 4.

For a concrete example consider when $n \approx 2^{160}$, and $u \approx \sqrt[3]{n} \approx 2^{53}$. We take the cost of a scalar multiplication in the group to be $C \approx 40$ group operations. Then the cost needed to find q using the algorithm is equivalent to about 2^{61} group operations.

In practice it may be impractical or impossible to query a SDHP oracle a large number ($\approx \sqrt[3]{n}$, i.e.) of times. However we stress that even for smaller values of u (i.e. values of u much less than $\approx \sqrt[3]{n}$) the algorithm applies, and in general the algorithm will still cost less than the $O(\sqrt{n})$

K163	$2 \cdot 3 \cdot 7 \cdot 89 \cdot 163 \cdot 1141450141721 \cdot 8405730267419952240402658413113$
B163	$2 \cdot 53 \cdot 383 \cdot 21179 \cdot 6799065232765820831739327781612099975633$
P192	$2^4 \cdot 5 \cdot 2389 \cdot 9564682313913860059195669 \cdot 3433859179316188682119986911$
P224	$2^2 \cdot 3^6 \cdot 5 \cdot 17 \cdot 2153 \cdot 50520606258875818707470860153287666700917696099933389351507$
K233	$2 \cdot 3^2 \cdot 11 \cdot 233 \cdot 108642473 \cdot 2207506409 \cdot 311893462098235579692316688834118334464906148909$
B233	$2 \cdot 7 \cdot 37 \cdot 18979 \cdot 38113 \cdot 202109 \cdot 3033517343 \cdot 30043507786353646304476366422610331237097041$
P256	$2^4 \cdot 3 \cdot 71 \cdot 131 \cdot 373 \cdot 3407 \cdot 17449 \cdot 38189 \cdot 187019741 \cdot 622491383 \cdot 1002328039319 \cdot 2624747550333869278416773953$
K283	$2^5 \cdot 3 \cdot 7^2 \cdot 11 \cdot 29 \cdot 71 \cdot 281 \cdot 283 \cdot 2671 \cdot 338407 \cdot 22546501513 \cdot 192823925599 \cdot 116698795142641026871500401147411093537981$
B283	$2 \cdot 3 \cdot 5 \cdot 1171 \cdot 21557 \cdot 709278129089 \cdot 77278332289855843 \cdot 270232628419903655237 \cdot 692755864106165926572290093$
P384	$2 \cdot 3^2 \cdot 7^2 \cdot 13 \cdot 1124679999981664229965379347 \cdot 3055465788140352002733946906144561090641249606160407884365391979704929268480326390471$
K409	$2 \cdot 3^3 \cdot 5 \cdot 7^3 \cdot 137 \cdot 193 \cdot 409 \cdot 2957 \cdot 218513 \cdot 884595540581 \cdot 577398720414295771068959122346240935663610269238758501582529986181411789537723272519316503$
B409	$2 \cdot 5 \cdot 19 \cdot 5197 \cdot 2967389 \cdot 373915204316167 \cdot 4452775636363539023772341 \cdot 10586924767739866570546627965598729 \cdot 12799224774627771513209574192269246077$

Table 1: Factorizations of $n-1$ for some NIST curves

group operations required by generic discrete logarithm algorithms such as the Shank's BSGS or Pollard-ρ algorithms. In particular for small values of u the algorithm cost will be close to $2C\sqrt{v}$, and so the algorithm may be faster than generic algorithms even for u as small as $4C^2$.

In some groups used for cryptography, index calculus algorithms for solving discrete logarithms have subexponential cost. These algorithms are usually significantly faster than generic algorithms. For example, in the group $\mathbb{F}_p^*$ of units in a finite field, where $p \approx 2^{1024}$, it is expected that index calculus algorithms can find discrete logarithms with a cost of about 2^{80} group operations.

The algorithm of Theorem 1 still applies for groups where index calculus methods are available, but it may not always result in a faster method to find logarithms in the group. The algorithm in Theorem 1 has a cost of at least $\sqrt[3]{n}$ scalar multiplications (including oracle queries), as noted earlier. If $n \approx 2^{1024}$, then this cost is at least 2^{341} group operations, which is far greater than the cost using index calculus methods.

Some standards, such as DSA in FIPS 186-2 and ANSI X9.30 and Diffie-Hellman in ANSI X9.42, use a group $\langle G \rangle$ that is a much smaller subgroup of the group $\mathbb{F}_p^*$. Generally, the size $\langle G \rangle$ is chosen so that generic DLP algorithms in $\langle G \rangle$ have approximately the same cost as index calculus algorithms in $\mathbb{F}_p^*$. In cases like this Theorem 1 may may provide a faster attack on the system than is currently known.

Some systems using hardware modules for protecting private keys might provide an SDHP oracle to an adversary. For example, a smart card is a highly constrained environment, and sometimes all hashing and key derivation is done outside the security boundary of the smart card, usually on the smart card reader. A smart card (holding a user's private key q) used in an encryption scheme may be presented with a recipient public key R and the card may simply return the element qR to the reader, where the subsequent encryption processing is performed. In this case, a malicious smart card reader could use the smart card as an SDHP oracle.

We now discuss certain protocols that provide a SDHP oracle to an adversary.

3.1 ElGamal Encryption

ElGamal encryption, in its original form (see [6], or [14, §8.4.2]) makes an SDHP oracle available. ElGamal encryption of message m to an entity with public key $Q = g^q$ results in a ciphertext $c = (c_1, c_2) = (g^x, mg^{qx})$. In a chosen ciphertext attack against an encryption scheme, an adversary can select any ciphertext and obtain its decryption. For ElGamal encryption, if this adversary chooses $c = (g^x, c_2)$, then it obtains $m = c_2/g^{qx}$. The adversary can compute $g^{qx} = c_2/m$, which solves SDHP_Q for instance g^x. Therefore with just a little arithmetic a chosen ciphertext adversary can use the ElGamal decryption oracle as an SDHP oracle.

ElGamal encryption is already known to be vulnerable to chosen ciphertext attacks. The known attacks mainly allow information to be learned about previously encrypted messages, but they do not leak anything about the private key. Finding the private key q is a serious break of the system, because all encryption is compromised. When using the victim as an SDHP oracle in this manner the private key q can be found with about $\sqrt[3]{n}$ cost, assuming the adversary knows a factor $u \approx \sqrt[3]{n}$ of $n-1$. The attack requires u decryption queries to the victim, which when $u \approx \sqrt[3]{n}$ may be too large to allow for a realistic attack. However as mentioned before smaller values of u can be used and may give a practical attack.

Two countermeasures to this attack on ElGamal encryption are to use a key derivation function or to use symmetric encryption rather than simple multiplication to transform the message. Such countermeasures are incorporated in more modern variants of the DH-based encryption, such as DHAES and ECIES [11].

A second, more fundamental, kind of countermeasure is to add integrity to the ciphertext. The integrity serves to ensure that a valid ciphertext can only be created by an entity that knows the ephemeral private key x associated with the ephemeral public key g^x. If the decryptor reveals g^{qx}, an entity who already knows x does not learn anything new, because g^{qx} is computable from the static public key g^q and the ephemeral private key x. In our attack the adversary chooses ephemeral public keys g^x for which it does not know the corresponding ephemeral private key x. This countermeasure is also part of DHAES and ECIES. Cramer and Shoup's public-key encryption scheme [3] also includes this countermeasure. The original rationale for including integrity in the ciphertext is to prevent chosen ciphertext attacks by making them no more effective than passive attacks.

Incidentally, the original description of the Diffie-Hellman key agreement protocol [5] did not use key derivation function. Therefore, if an adversary has some means of obtaining the DH shared secret key, our attack may be possible.

3.2 Ford-Kaliski Key Retrieval

In the Ford-Kaliski key retrieval scheme [7], which is currently being standardized in [8] and [9], an SDHP oracle is available. Part of this scheme (see Figure 1) requires the server to compute c^q for any value c sent to it by a client, where q is a long-term key. The server thus provides an SDHP oracle to an adversary, so the security of q is potentially weakened by an adversary using our attack.

$$\begin{array}{lcl}
\underline{\text{Client}} & & \underline{\text{Server}} \\
b \in_R [1, n-1] & & \\
c = g_\pi^b & & \\
 & \xrightarrow{c} & \\
 & & s = c^q \\
 & \xleftarrow{s} & \\
z = s^{b^{-1} \bmod n} & &
\end{array}$$

Figure 1: Ford-Kaliski Key Retrieval

The generator g_π in Figure 1 is derived from user password π. The value $z = g_\pi^d$ depends on a user secret and on a server secret, and is called a *hardened password*. In practice, cryptographic keys will be derived from z. The protocol prevents dictionary attacks against the user's password because z is derived from a strong secret q kept by the server. If q is revealed, then the user becomes vulnerable to dictionary searches.

In early drafts of [8] and [9], elliptic curve groups are allowed with this protocol, so our attack may be relevant to the protocol as described there. Recent versions of these draft standards have incorporated measures specifically designed to counter our attack.

3.3 Chaum and van Antwerpen's Undeniable Signatures

Another scheme in which an SDHP oracle is available is Chaum and van Antwerpen's undeniable signature scheme [2]. In this scheme, the signer has private key q and computes her signature on message m as $s = m^q$. The groups suggested for use in [2] are $\mathbb{F}_p^*$ where p is a strong prime. Therefore, this protocol is not affected by the attack, because index calculus methods solve the DLP faster than our attack. However if the protocol is implemented with other groups such as elliptic curve groups or DSA groups, which is reasonable because they make the scheme much more efficient, then our attacks may be viable. A very simple countermeasure to this attack is to hash the message before exponentiating.

4 The Algorithm as a Reduction

In Diffie-Hellman (DH) key agreement [5], Alice and Bob exchange g^x and g^y and then compute a shared secret g^{xy}, using their respective knowledge of x and y. Hardness of the DHP is necessary to ensure that an adversary who sees g^x and g^y cannot easily compute the shared secret g^{xy}.

In the static variant of DH key agreement, Alice has a static private key q, meaning she always uses $x = q$. It is unclear how this reuse affects her security. If the instances of the DHP are arranged in a square matrix indexed by $\langle G \rangle \times \langle G \rangle$, as in Figure 2(a), then Alice is more concerned about the hardness of the Diffie-Hellman problem instances in the single row defined by $x = q$ (Figure 2(b).) Even if the DHP is hard for almost all instances, it is possible that it is easy when restricted to Alice's row, and so hardness of the DHP may not be sufficient to ensure security for Alice. The *static Diffie-Hellman problem* (SDHP) is to solve this subset of the instances of the DHP that Alice would be concerned with:

Problem 1 (SDHP). *Given fixed g and g^q, and random g^y, find g^{qy}.*

It is not apparent from existing results that computing the SDHP_Q function is hard if q is unknown. In this section we discuss a version of this statement that is implied by Theorem 1. Algorithms used in previous reductions of this sort have required an input function computing the Diffie-Hellman

$$\begin{array}{cccc} g & \cdots & g^y & \cdots \\ \vdots & \ddots & \vdots & \ddots \\ g^x & \cdots & g^{xy} & \cdots \\ \vdots & \ddots & \vdots & \ddots \end{array}$$

(a) DHP

$$\begin{array}{cccc} g^q & \cdots & g^{qy} & \cdots \end{array}$$

(b) SDHP

Figure 2: The SDHP as a single row of the DHP matrix

secret corresponding to any *pair* of group elements, and subsequently finds logarithms of arbitrary elements in the group. Our algorithm works when the input function can only compute Diffie-Hellman secrets when one of the inputs is fixed to be a group element Q, but as a consequence can only compute the discrete logarithm corresponding to Q.

den Boer [4] shows that if $n-1$ is smooth, then DLP $\leqslant_P$ DHP, where $\leqslant_P$ indicates a polynomial-cost reduction. The intuition is that given a DHP-oracle, one can basically perform the Pohlig-Hellman algorithm in the exponent space, to find a discrete logarithm in the multiplicative group $\mathbb{F}_n^*$, which in turn provides a logarithm in $\langle G \rangle$.

Maurer and Wolf [12] show that if a certain auxiliary group exists, then DLP $\leqslant_P$ DHP. The auxiliary group has smooth order and its elements are represented using elements from $\langle G \rangle$. This work can be thought of as extending den Boer's work in the same way that Lenstra's elliptic curve factoring method extends Pollard's $p-1$ factoring method.

Boneh and Lipton [1] proved that there is always a subexponential-cost reduction DLP $\leqslant$ DHP. One consequence is that the DHP can be solved in subexponential time only if the DLP can be solved in subexponential time.

Muzereau, Smart and Vercauteren [15] find auxiliary elliptic curves for some NIST recommended elliptic curves, and give exact estimates of the difficulty of the DHP in comparison to the assumed difficulty of the DLP for these curves.

4.1 Hardness of the SDHP

Maurer and Wolf [13, §3] discuss how strongly one can connect the DHP to the DLP:

> As mentioned already, it is obvious that the DH problem is at most as hard as the DL problem. The strongest form of the

> converse statement would be that no more efficient way exists for solving the DH problem than to solve the DL problem first. In a strict sense, this would mean that given g^u and g^v, it is only possible to obtain g^{uv} when computing u or v first. However, it appears that such a statement can be proved only by giving an efficient algorithm that, when given g^u, g^v, and g^{uv}, computes u or v. Of course such an algorithm can only exist for groups for which it is easy to compute discrete logarithms because this algorithm itself can be used to compute the discrete logarithm of a group element a efficiently when giving as input a, g^s (in random order), and a^s.
>
> A less strict version is that for groups for which the DH problem can be solved efficiently *for all instances* (or at least a non-negligible fraction) it is possible to compute discrete logarithms efficiently. It was shown that this is true for certain classes of groups.

Instead of considering all instances of the DHP, or a non-negligible fraction thereof, the SDHP involves a *negligible fraction of instances.* So, according to the notion of strength that Maurer and Wolf discuss above, by considering fewer instances, we are establishing a *stronger form of converse statement*, namely a tighter connection between the DHP and DLP.

If we assume that finding q is hard in the sense that it cannot be found with computational cost less than $\sqrt{n}$ group operations in $\langle G \rangle$, then our reduction implies the SDHP is hard. The bound on the difficulty of the SDHP given by our result depends on the relative size of u and v. In the following, we have selected u so that the resulting lower bound on the difficulty of the SDHP is as large as possible. Weaker results are obtained for other values of u.

Corollary 2. *Assume that the conditions of Theorem 1 hold, that a scalar multiplication has cost C, that $u = 9C^2$, and that $C > 4$. If the private key q cannot be found with computational cost less than $\sqrt{n}$, then any algorithm that solves SDHP has a computational cost of at least*

$$\frac{\sqrt{n}}{27C^2} - 1. \tag{7}$$

Proof. Suppose that an algorithm can compute any SDHP query with computational cost equivalent to at most W group operations. From Theorem 1 and our assumption, we can find q with cost:

$$uW + 2C(\lceil \sqrt{u}\, \rceil + \lceil \sqrt{v}\, \rceil). \tag{8}$$

By assumption, the cost of finding q is at least $\sqrt{n}$, so:

$$uW + 2C(\lceil \sqrt{u}\, \rceil + \lceil \sqrt{v}\, \rceil) \geqslant \sqrt{n}$$

and thus

$$\begin{aligned} W &\geqslant (\sqrt{n} - 2C(\lceil \sqrt{u}\, \rceil + \lceil \sqrt{v}\, \rceil))/u \\ &> \sqrt{n}/u - 2C(\sqrt{u} + 1 + \sqrt{n/u} + 1)/u \\ &= \sqrt{n}(1 - 2C/\sqrt{u})/u - 2C(\sqrt{u} + 2)/u \\ &= \sqrt{n}(1 - (2C)/(3C))/(9C^2) - 2C(3C + 2)/(9C^2) \qquad (9) \\ &= \sqrt{n}/(27C^2) - (6C + 4)/(9C) \\ &\geqslant \sqrt{n}/(27C^2) - (6C + C)/(9C) \\ &> \sqrt{n}/(27C^2) - 1. \end{aligned}$$

□

To see what this means in practice, consider when the group is an elliptic curve group of prime order $n \approx 2^{160}$. Then we take $C = \log_2(n)/4 \approx 40$, if we wish to consider a cost model in which storage is expensive. Assume that a private key q cannot be found with computational cost less than 2^{80}. Then, computing the value of SDHP without access to q cannot be done with computational cost less than

$$\frac{2^{80}}{27 \cdot 40^2} - 1 \approx 2^{80}/2^{15.398\ldots} \approx 2^{64}. \qquad (10)$$

This is a lower bound on the hardness of the SDHP given a lower bound on the hardness of finding q. Ideally, one would like a lower bound of 2^{80}, because the best known generic attacks on the SDHP still cost this much.

If one instead takes a cost model in which storage is less expensive, so that a group operation and a group element storage each cost a unit, then a smaller value of C may be used, for example $C = 5$. Then the cost of breaking the SDHP for a similar sized n is at least:

$$\frac{2^{80}}{27 \cdot 25} \approx 2^{80}/675 \approx 2^{70.6}. \qquad (11)$$

Asymptotically, the cost of finding q is at least $O(\sqrt{n})$ then the cost breaking SDHP is $O(\sqrt{n}/\log(n)^2)$ in the storage-efficient cost model or $O(\sqrt{n})$ in the storage-relaxed cost model. In other words, asymptotically the reduction is quite tight, polynomial time and constant time, respectively, in the two cost models.

For the NIST curves, $n-1$ is not smooth. Therefore den Boer's reduction [4] does not apply. Muzereau, Smart and Vercauteren [15] found auxiliary groups for most of the NIST curves, thereby explicitly demonstrating that Maurer and Wolf's [12] reduction can be made into an efficient reduction.[2] These results establish that the DHP is almost as hard as the DLP in the NIST groups, although a main point of our paper is that one must be careful equating hardness of the DHP with the security of an individual private key repeatedly used in key exchanges.

4.2 A Hierarchy of Hard Problems

The SDHP cannot be harder than the DHP, because a solver of the DHP can be used to solve the SDHP, as follows. Suppose an algorithm $\mathcal{A}$ solves the DHP with success probability p, assessed over the random choices of $x, y \in [1, n-1]$. To solve SDHP, choose random z and let $g^x = (g^q)^z$. Then use $\mathcal{A}$ to solve the DHP with probability p of finding g^{xy}. The solution to the SDHP is $(g^{xy})^{z^{-1} \bmod n}$. We write SDHP $\leqslant$ DHP, or SDHP $\leqslant_P$ DHP to emphasize that this is a polynomial-time reduction.

This relationship between the SDHP and the DHP is akin to the *random self-reducibility* (RSR) property of the DHP. An algorithm for solving the DHP with probability p can be converted into algorithm for solving any instance of the SDHP, except that is slower by factor of $1/p$. To solve a SDHP instance with probability close to 1, even if the probability p is small, we can keep choosing z until we can solve the DHP instance. This will take $1/p$ attempts on average.

Conversely, an algorithm for solving the SDHP can be regarded an algorithm for solving the DHP with a $1/n$ chance of success. Therefore random self-reducibility can be applied. Unfortunately, it is not useful, because the success probability is so low. We would get a DHP solver that is n times slower than the SDHP solver and this is much slower than the $O(\sqrt{n})$-time solution to the DHP that first solves a DLP instance (using Pollard-ρ, say) and then exponentiating.

A subtle point when comparing the SDHP and the DLP is that in Corollary 2 only one instance of the discrete logarithm can be solved, the instance g^q. This leads one to consider the *single-instance Discrete Logarithm Problem* (SDLP), in which one is given fixed g^q, with the task being to find q. Previous work [4, 12, 1] compared the DHP and DLP, while ours compares the SDHP and SDLP. Thus our work can be considered as *parallel* to

[2] Incidentally, the lower bounds in [15] for hardness of the DHP are similar to our lower bound.

previous work, in the sense that it works in the *single-instance setting.*

4.3 Analogy with the Rabin Cryptosystem

Although the RSA cryptosystem is very similar to the Rabin cryptosystem, nobody has proven that duplicating the raw RSA private key operation is as hard as finding the RSA private key. The RSA inversion problem, which must be hard for RSA to be secure, is therefore potentially easier than integer factorization.

The original Rabin cryptosystem [17] has the property that duplicating the private key operation is almost as hard as finding the private key. However the original Rabin cryptosystem is vulnerable to active attacks. The modern Rabin cryptosystem uses hash-based padding to protect against active attacks. Furthermore, this padding also allows for security proofs of the scheme in the random oracle model. Active attacks on the original Rabin cryptosystem are analogous to our attack, and the modern padding mechanism for Rabin cryptosystems is analogous the key derivation function mechanism for DH cryptosystems. Both these mechanisms prevent the reduction algorithms that prove security from being turned into attacks that hurt the security.

With DH based cryptosystems it was potentially the case that breaking DH cryptosystems was potentially much easier than solving the DLP. In other words, the situation was like the situation for RSA today. With the pioneering work of den Boer's and others, DH cryptosystems were put on a better footing: they were based on a problem provably almost as hard as the DLP. Our reduction can be thought of as extending this effort. Specifically, we establish that duplicating an individual entity's private key operation is almost as hard as finding the individual's private key.

5 Acknowledgments

Thanks to Alfred Menezes for many helpful discussions. Scott Vanstone and Certicom generously supported our pursuit of this research topic.

References

[1] D. Boneh and R.J. Lipton, *Algorithms for black-box fields and their application to cryptography*, Advances in Cryptology — CRYPTO '96 (N. Koblitz, ed.), LNCS, vol. 1070, Springer, 1996, pp. 283–297.

[2] D. Chaum and H. van Antwerpen, *Undeniable signatures*, Advances in Cryptology — CRYPTO '89 (G. Brassard, ed.), LNCS, vol. 435, Springer, 1989, pp. 212–217.

[3] R. Cramer and V. Shoup, *A practical public key encryption scheme provably secure against adaptive chosen ciphertext attacks*, Advances in Cryptology — CRYPTO '98, LNCS, vol. 1462, Springer, 1998, Extended version available at www.shoup.net, pp. 13–25.

[4] B. den Boer, *Diffie-Hellman is as strong as discrete log for certain primes*, Advances in Cryptology — CRYPTO '88, LNCS, vol. 403, Springer, 1988, p. 530.

[5] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 644–654.

[6] T. ElGamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*, Advances in Cryptology — Proceedings of CRYPTO '84 (G. Goos and J. Hartmanis, eds.), LNCS, vol. 196, Springer, 1985, pp. 10–18.

[7] W. Ford and B. Kaliski, *Server-assisted generation of a strong secret from a password*, 9th International Workshop on Enabling Technologies — WET ICE 2000, IEEE Press, 2000.

[8] IEEE P1363.2, *Standard specification for password-based public key cryptographic techniques*, IEEE Computer Society, May 2004, Draft version 15.

[9] ISO 11770-4, *Information technology — security techniques — key management — part 4: Mechanisms based on weak secrets*, ISO, November 2004.

[10] D. E. Knuth, *Seminumerical algorithms*, 3rd ed., The Art of Computer Programming, vol. 2, Addison-Wesley, 1998.

[11] P. Rogaway M. Abdalla, M. Bellare, *The oracle diffie-hellman assumptions and an analysis of dhies*, Topics in Cryptology - CT-RSA 01 (D. Naccache, ed.), LNCS, vol. 2020, Springer-Verlag, 2001.

[12] U. M. Maurer and S. Wolf, *Diffie-Hellman oracles*, Advances in Cryptology — CRYPTO '96 (N. Koblitz, ed.), LNCS, vol. 1070, Springer, 1996, pp. 268–282.

[13] ______, *The Diffie-Hellman protocol*, Designs, Codes and Cryptography **19** (2000), 141–171.

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1996, See www.cacr.math.uwaterloo.ca/hac.

[15] A. Muzereau, N. P. Smart, and F. Vercauteren, *The equivalence between the DHP and DLP for elliptic curves used in practical applications*, LMS J. Comput. Math. **7** (2004), 50–72, See www.lms.ac.uk.

[16] J. Pollard, *Monte-carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918–924.

[17] M. O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Tech. Report 212, Massachussets Institute of Technology Laboratory for Computer Science, 1979.