The Static Diffie-Hellman Problem

Daniel R. L. Brown* and Robert P. Gallant*

November 15, 2004†

Abstract

The *static Diffie-Hellman problem* (SDHP) is the special case of the classic Diffie-Hellman problem where one of the public keys is fixed. We establish that the SDHP is almost as hard as the associated discrete logarithm problem. We do this by giving a reduction that shows that if the SDHP can be solved then the associated private key can be found.

The reduction also establishes that certain systems have less security than anticipated. The systems affected are based on static Diffie-Hellman key agreement and do not use a key derivation function. This includes some cryptographic protocols: basic ElGamal encryption; Chaum and van Antwerpen's undeniable signature scheme; and Ford and Kaliski's key retrieval scheme, which is currently being standardized in IEEE P1363.2.

1 Introduction

Let $\mathbb{G}$ be a cyclic group with generator g of prime order n. The classic Diffie-Hellman problem (DHP) in $\mathbb{G}$ is:

Problem 1 (DHP). *Given fixed g and random g^x and g^y, find g^{xy}.*

In Diffie-Hellman (DH) key agreement, 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 to do so. 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 static DH key agreement, Alice has a static private key d, meaning she always uses $x = d$. It is unclear how this reuse affects her security. If the instances of the DHP are arranged in a square matrix indexed by $\mathbb{G} \times \mathbb{G}$, as in Figure 1, then Alice is more concerned about the hardness of the Diffie-Hellman problem instances in the single row defined by $x = d$. 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 2 (SDHP$_d$). *Given fixed g and g^d, and random g^y, find g^{dy}.*

It is not apparent, from existing results, that the SDHP is a hard problem if d is unknown. In Section 3, we establish that the SDHP is hard if the associated discrete logarithm problem of finding d is hard. The relationship of the SDHP to the DHP and other problems are discussed in Section 5.

To show that the SDHP$_d$ is hard, we use an algorithm, given in Section 2, that queries an SDHP$_d$ oracle at certain instances g^y in order to compute d. This reduction establishes that Alice's SDHP$_d$ instances are hard to compute for her private key d, if her private key d is hard to find. If

*Certicom Research

†Preliminary version submitted to Eurocrypt 2005

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

(a) DHP (b) SDHP_d

Figure 1: The SDHP_d as a single row of the DHP matrix

we assume the hardness of the discrete logarithm problem, then it is likely that Alice's randomly chosen private key d cannot be recovered from her public key g^d. From the hardness of finding d and from our reduction, it follows that the SDHP_d must be hard for Alice's key d. Therefore Alice has further assurance in the security of systems based on static DH key agreement: she is assured that the SDHP_d is not weak for her specific key d.

The reduction between solving SDHP_d and finding d also leads to new vulnerabilities in some systems using static DH key agreement, which we describe in Section 4. These vulnerabilities arise when one of the system entities acts as an SDHP_d oracle. Such systems include the Ford-Kaliski server-assisted key generation protocol [FK00], basic ElGamal encryption [ElG85], Chaum and van Antwerpen's Undeniable Signatures [CvA89], and certain smart card implementations. Our reduction, in this setting, becomes an attack because the entity becomes an SDHP_d oracle available to be queried. Our reduction queries this oracle and subsequently computes the private key d. How this vulnerability affects the security level of a system will depend on many factors, and this vulnerability is but one consideration.

We are not aware of any previous work reducing the difficulty of the SDHP to other problems. Nor are we aware of any previous work exploiting an SDHP oracle available in a cryptosystem to find the associated private key.

1.1 Previous Work on the Diffie-Hellman Problem

The *Discrete Logarithm Problem* (DLP), which is to find x given random g^x, has been extensively studied for various groups. Its computational intractability is necessary for the security of many cryptographic protocols including Diffie-Hellman key agreement and the Digital Signature Algorithm.

It was recognized very early that hardness of the DHP is also necessary for security of some protocols. To avoid the additional assumption that the DHP is computationally hard, one might hope to deduce the difficulty of the DHP from the difficulty of the DLP. A number of works compare the DHP to the DLP. A brief summary of these efforts is given below:

- den Boer [dB88] 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 $\mathbb{G}$.

- Maurer and Wolf [MW96] 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 $\mathbb{G}$. 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 [BL96] 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.

- Exact, instead of asymptotic, analyses are available for all of the results above. In particular, Muzereau, Smart and Vercauteren [MSV04] 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.

2 How to Find d with an SDHP_d Oracle

In this section, we show that if $n = uv + 1$, then d can be found with u calls to an SDHP_d oracle, and with further off-line computational work of about $(\sqrt{u} + \sqrt{v})$ group operations.

For our reduction it is easier to work with additive notation for the group on which the SDHP_d oracle acts. This notation is also commonly used for elliptic curve groups. We write group elements with uppercase letters, so the group generator, written in §1 as g, is now written G. Scalar multiplication, the operation corresponding to exponentiation, is written with the integer multiple on the left, usually in lowercase variables. So, if d is a private key, then the public key associated with it is dG, instead of g^d. The order of the generator is still n, which now means $nG = 0$, instead of $g^n = 1$, since the group identity is now written as 0. We assume that n is prime, for simplicity. We write $\mathbb{F}_n$ for the finite field of size n, and without loss of generality, assume that the private key $d \in \mathbb{F}_n$. We now define an SDHP_d oracle more formally:

Definition 1 (SDHP_d Oracle). *Let $\mathbb{G}$ be a group of prime order n with generator G. Let $d \in \mathbb{F}_n$ be arbitrary. An* SDHP_d oracle *(with respect to $\mathbb{G}$) computes the function $\delta : \mathbb{G} \to \mathbb{G}$ defined by:*

$$\delta(X) = dX. \tag{1}$$

The basic idea of the proof is to work in the exponent space. To find d, we instead find z, where $d = g^z$ for some primitive element (generator) g of $\mathbb{F}_n^*$. To do this, we first compute $d^u G$ using the SDHP_d oracle u times. We then find integers $t \in [0, u-1]$ and $y \in [0, v-1]$ such that $z = tv + y$. We accomplish this by solving discrete logarithms in subgroups of $\mathbb{F}_n^*$ of order u and v, represented as subsets of $\mathbb{G}$.

Theorem 1. *Assume the conditions of Definition 1 hold, and that $n = uv + 1$ for positive integers u and v. Then d can be found with an algorithm that makes u calls to the SDHP_d oracle δ, does at most $2(\lceil \sqrt{u}\, \rceil + \lceil \sqrt{v}\, \rceil)$ scalar multiplications and table lookups in the group $\mathbb{G}$, and performs a constant number of operations in the field $\mathbb{F}_n$.*

Proof. Let g be a generator of $\mathbb{F}_n^*$. (Recall that the group of units $\mathbb{F}_n^*$ in a finite field is cyclic, so g exists.) We use multiplicative notation for the group operations in $\mathbb{F}_n^*$. For simplicity, we omit the notation $\bmod n$ in $\mathbb{F}_n^*$.

The case $d = 0$ is easily detected by checking that $\delta(G) = 0$, so henceforth we will assume that $d \in \mathbb{F}_n^*$. Because g generates $\mathbb{F}_n^*$, we have $d = g^z$ for some integer $z \in [0, n-2]$. We will find the value of z.

We first find z modulo v as follows. Let $w = g^u$. The subgroup $\langle w \rangle$ of $\mathbb{F}_n^*$ generated by w is the unique subgroup of $\mathbb{F}_n^*$ with order v, and it consists of precisely those $x \in \mathbb{F}_n^*$ such that $x^v = 1$. We first obtain an element $E \in \{xG | x \in \langle w \rangle\}$ by applying u iterations of δ to G:

$$E = \delta^u(G) = d^u G. \tag{2}$$

Let $e = d^u$, so that $E = eG$. We know $e \in \langle w \rangle$, because $e^v = d^{uv} = d^{n-1} = 1$. Therefore $e = w^y$ for some integer $y \in [0, v-1]$. 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 $0 \leqslant y_r < m$. Thus the group elements $w^{-y_q m}E$ and $w^{y_r}G$ are equal.

To find y, we find the values of y_q, y_r by emulating Shanks' Baby-Step-Giant-Step (BSGS) algorithm [MvOV96] to find an element in each of the lists

$$U = \{(G, 0), (wG, 1), (w^2 G, 2), \ldots, (w^{m-1}G, m-1)\} \tag{3}$$

and

$$V = \{(E, 0), (w^{-m}E, 1), (w^{-2m}E, 2), \ldots, (w^{-(m-1)m}E, m-1)\}. \tag{4}$$

such that the first coordinate of each element is the same. The second coordinate of these pairs determine i and j such that $w^i G = w^{-jm}E$. From such i and j we can solve for y because:

$$E = w^{i+jm}G \quad \Rightarrow \quad e = w^{i+jm} \quad \Rightarrow \quad y = i + jm. \tag{5}$$

For this value of y we have $d^u = w^y$, and so $g^{uz} = g^{uy}$, and so $uz \equiv uy \bmod n-1$, implying $z \equiv y \bmod v$.

Since $z \equiv y \bmod v$, $z \in [0, n-2]$, and $y \in [0, v-1]$, we know $z = tv + y$ for some integer $t \in [0, u-1]$. We will now find t. Let $q = g^v$. Then $\langle q \rangle$ is the order u subgroup of $\mathbb{F}_n^*$, and consists of precisely those $x \in \mathbb{F}_n^*$ such that $x^u = 1$. Compute the point

$$F = g^{-y}\delta(G). \tag{6}$$

Since $F \in \langle G \rangle$, $F = fG$ for some $f \in \mathbb{F}_n$, and in fact we have $f = g^{z-y}$. Since $f^u = d^u/g^{uy} = e/w^y = 1$, we have $f \in \langle q \rangle$. Therefore $f = q^t$ for some integer $t \in [0, u-1]$. 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

$$Q = \{(G, 0), (qG, 1), (q^2 G, 2), \ldots, (q^{k-1}G, k-1)\} \tag{7}$$

and

$$W = \{(F, 0), (q^{-k}F, 1), (q^{-2k}F, 2), \ldots, (q^{-(k-1)k}F, k-1)\}, \tag{8}$$

such that the first coordinates are the same. The second coordinates provide integers h, l such that $q^h G = q^{-lk}F$. To find t, note:

$$F = q^{h+lk}G \quad \Rightarrow \quad f = q^{h+lk} \quad \Rightarrow \quad t = h + lk. \tag{9}$$

Now $z = tv + y$ can be computed, since t, v and y have been determined. Therefore we compute $d = g^z$. Alternatively, from the values t and y, we can calculate the value of d. Recall that $f = q^t$ and $f = dg^{-y}$, so therefore:

$$d = g^y f = g^y q^t. \tag{10}$$

The computations needed to compute d are u queries to the SDHP_d oracle δ, at most $2\lceil \sqrt{u}\, \rceil + 2\lceil \sqrt{v}\, \rceil$ scalar multiplications and table lookups in the group $\mathbb{G}$, six exponentiations in $\mathbb{F}_n^*$ (g^u, w^m, g^{-y}, g^v, q^k, q^t), and two arithmetic operations ($y = i + jm$ and $t = h + lk$). $\square$

Other than the oracle queries, the majority of the cost in a routine implementation of this algorithm is in computing and storing the set U in (3), followed by an iteration over each element in the set V in (4). For many groups of cryptographic interest, the cost of this algorithm will be dominated by the scalar multiplications of integers in $\mathbb{F}_n$ by one of the group elements G or E. So, henceforth, we will ignore the constant number of operations in $\mathbb{F}_n$ and the table-lookups.

The minimum cost of the algorithm is achieved when $u \approx \sqrt[3]{n}$. We will consider this optimum value of u in §4 when we use this algorithm as an attack to find vulnerabilities in certain systems. We note that 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 [Knu98, §4.5.4]. Taking v to be the largest prime factor, then $u \approx n^{1/3}$ is roughly the optimal value for minimizing the number of scalar multiplications.

Values of u smaller or larger than the optimal $\sqrt[3]{n}$ generally require more than the minimum $2\sqrt[3]{n}$ cost. Nevertheless, the cost of our algorithm for finding d will still be less than generic $O(\sqrt{n})$-cost algorithms for finding d such as BSGS and Pollard-ρ. Therefore, our algorithm is still useful when $n-1$ does not have, or is not known to have, a factor $u \approx \sqrt[3]{n}$.

It is also important to consider what happens for small values of u, because in some ways $u \approx \sqrt[3]{n}$ is too large. A cutoff point exists where if u is any smaller than the Theorem 1 algorithm costs more than the $\sqrt{n}$ groups operations needed by generic DLP algorithms for finding d. The cutoff determines the minimal value of u where the reduction can be meaningful. The cutoff determines the rough ratio between the hardness of the SDHP_d and finding d. The cutoff depends on the number of group operations needed per scalar multiplication, which we will call C.

Because so many scalar multiplications by these group elements are required (in some cases $2\sqrt[3]{n}$ scalar multiplications in total), 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 us 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. We can henceforth assume $C \approx \log_2(n)/4$.

For simplicity, the proof of Theorem 1 uses the BSGS algorithm to find d. Other generic algorithms, such as Pollard-ρ can also be used. The fairly straightforward details are left to §B.2.

2.1 Relevance to Some Groups of Cryptographic Interest

In some groups of cryptographic interest, index calculus algorithms for solving discrete logarithms have subexponential cost. Therefore, these algorithms are potentially significantly faster than generic algorithms like BSGS and Pollard-ρ. Moreover, experiments have shown that these improvements do not just exist in the asymptotic long run, but occur for the sizes of groups used in cryptography. 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 in Theorem 1 has a cost of at least $\sqrt[3]{n}$ scalar multiplications, 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. Therefore, the algorithm is not too relevant when index calculus methods are available.

In the groups $\mathbb{F}_p^*$, we select $\mathbb{G}$ as a prime order subgroup. Thus $n \mid p-1$. If $n = (p-1)/2$, then index calculus in $\mathbb{F}_p^*$ is far faster than generic DLP algorithms in $\mathbb{G}$. This requires p to be a Sophie Germaine prime, also known as *strong* prime, which is such that $n = (p-1)/2$ is prime. Strong primes are moderately rare among random primes, but have been used in some cryptographic protocols, such as IPSec. In these groups, our algorithm does not provide any advantage in finding the private key over generic algorithms. Therefore, in these groups, our algorithm is not relevant as an attack or as a proof that the SDHP is hard.

Some standards, however, choose $\mathbb{G}$ as a smaller subgroup of $\mathbb{F}_p^*$. In DSA, the size $\mathbb{G}$ has been chosen so that generic DLP algorithms in $\mathbb{G}$ have approximately the same cost as index calculus algorithms in $\mathbb{F}_p^*$. For these groups, therefore, our algorithm is meaningful, provided that $n-1$ has an appropriate factorization.

Yet another class of groups of cryptographic interest are elliptic curve groups. Elliptic curve groups are generally not known to have index calculus like methods. In almost all cases, the fastest known algorithms for solving the elliptic curve DLP are the generic algorithms which take $O(\sqrt{n})$ group operations. Therefore, with elliptic curve groups, our algorithm is meaningful, provided that a factorization $n-1 = uv$ of the right form is available.

Elliptic curves of considerable interest are those recommended by NIST. Complete factorizations of $n-1$ for most 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 a smaller u can be found. In particular, a u near to the cutoff level can found, which helps establish the ratio between the difficulty of the SDHP_d and finding d.

K163	$2 \cdot 3 \cdot 7 \cdot 89 \cdot 163 \cdot 1141450141721 \cdot 840573026741995224040265841311 3$
B163	$2 \cdot 53 \cdot 383 \cdot 21179 \cdot 6799065232765820831739327781612099975633$
P192	$2^4 \cdot 5 \cdot 2389 \cdot 9564682313913860059195669 \cdot 3433859179316188682119986911$
P224	$2^2 \cdot 3^6 \cdot 5 \cdot 17 \cdot 2153 \cdot 505206062588758187074708601532876667009176960999333893515 07$
K233	$2 \cdot 3^2 \cdot 11 \cdot 233 \cdot 108642473 \cdot 2207506409 \cdot 3118934620982355796923166888341183344649061489 09$
B233	$2 \cdot 7 \cdot 37 \cdot 18979 \cdot 38113 \cdot 202109 \cdot 3033517343 \cdot 300435077863536463044763664226103312370970 41$
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	$2^5 \cdot 3 \cdot 7^2 \cdot 11 \cdot 29 \cdot 71 \cdot 281 \cdot 283 \cdot 2671 \cdot 338407 \cdot 22546501513 \cdot 192823925599 \cdot 1166987951426410268715004011474110935379 81$
B283	$2 \cdot 3 \cdot 5 \cdot 1171 \cdot 21557 \cdot 709278129089 \cdot 77278332289855843 \cdot 270232628419903655237 \cdot 6927558641061659265722900 93$
P384	$2 \cdot 3^2 \cdot 7^2 \cdot 13 \cdot 1124679999981664229965379347 \cdot 30554657881403520027339469061445610906412496061604078843653919797049292684803263904 71$
K409	$2 \cdot 3^3 \cdot 5 \cdot 7^3 \cdot 137 \cdot 193 \cdot 409 \cdot 2957 \cdot 218513 \cdot 884595540581 \cdot 57739872041429577106895912234624093566361026923875850158252998618141178953772327251931650 3$
B409	$2 \cdot 5 \cdot 19 \cdot 5197 \cdot 2967389 \cdot 373915204316167 \cdot 445277563636353902377234 1 \cdot 1058692476773986657054662796559872 9 \cdot 1279922477462777151320957419226924607 7$

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

For the NIST curves, $n-1$ is not smooth so den Boer's reduction [dB88] does not apply. Muzereau, Smart and Vercauteren [MSV04], however, have found the auxiliary groups for most of the NIST curves. Therefore, Maurer and Wolf's [MW96] reduction becomes an efficient reduction, which establishes in these groups that DHP is almost as hard as the DLP. Given that the DHP is presumably hard, it is natural to further presume that static Diffie-Hellman key agreement is secure in this group. However, if the protocol makes a static Diffie-Hellman oracle available, then the private key can be found with less cost than by generic DLP algorithms. Therefore, using these groups with a system like the Ford-Kaliski key retrieval scheme [FK00] is not advisable, despite the positive security results of [MSV04]. More details and examples are given in §4

Although it is good for the SDHP to be hard, in some systems it is better if our algorithm does not work, because otherwise the private key is exposed, as in the Ford-Kaliski protocol. In this case we want to avoid factors u of $n-1$ that make our algorithm more effective than generic algorithms. In particular, we want to avoid the optimal $u \approx \sqrt[3]{n}$. In general, we want the small factors of u to fall below the cutoff level. An extreme case of this is to have $n = 2r + 1$, where r is prime. Such groups do not benefit from our security reduction, however, because the only small factors u are below the cutoff level mentioned earlier. Another possibility is to have $n = ur + 1$ where r is prime and u is in a range small enough to make our security reduction applicable but not big enough to make the attack significant.

3 Hardness of the SDHP$_d$

If we assume that finding d is hard in the sense that it cannot be found with computational cost less than $\sqrt{n}$ group operations in $\mathbb{G}$, then our reduction implies the SDHP$_d$ is hard. The bound on the difficulty of the SDHP$_d$ 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$_d$ is as large as possible, which is done by choosing the u near the cutoff level mentioned earlier.

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

$$\frac{\sqrt{n}}{27C^2} - 1 \tag{11}$$

group operations in $\mathbb{G}$.

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

$$uW + 2C(\lceil \sqrt{u}\, \rceil + \lceil \sqrt{v}\, \rceil). \tag{12}$$

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

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

and thus

$$\begin{aligned}
W &\geqslant (\sqrt{n} - 2C(\lceil \sqrt{u}\, \rceil + \lceil \sqrt{v}\, \rceil))/u \\
&> \sqrt{n}/u - 2C(\sqrt{u} + 1 + \sqrt{n/u} + 1)/u \\
&= \sqrt{n}(1 - 2C/\sqrt{u})/u - 2C(\sqrt{u} + 2)/u \\
&= \sqrt{n}(1 - (2C)/(3C))/(9C^2) - 2C(3C + 2)/(9C^2) \\
&= \sqrt{n}/(27C^2) - (6C + 4)/(9C) \\
&\geqslant \sqrt{n}/(27C^2) - (6C + C)/(9C) \\
&> \sqrt{n}/(27C^2) - 1.
\end{aligned} \tag{13}$$

$\square$

Weaker results are obtained for other values of u. As u approaches 1 or $n-1$, then the lower bound on W approaches $\sqrt{n}$.

To see what this means for the kinds of DH groups used in practice, consider the case when the group is an elliptic curve group of prime order $n \approx 2^{160}$. Then we take $C = \log_2(n)/4 \approx 40$. Assume that a private key d cannot be found with computational cost less than 2^{80} group operations. Then, computing the value of SDHP_d without access to d cannot be done with computational cost less than

$$\frac{2^{80}}{27 \cdot 40^2} - 1 \approx 2^{80}/2^{15.398\ldots} \approx 2^{64}. \tag{14}$$

This is a lower bound on the hardness of the SDHP_d given a lower bound on the hardness of finding d. Ideally, one would like a lower bound of 2^{80}, because the best known generic attacks on the SDHP_d still cost this much. Incidentally, the lower bounds in [MSV04] for hardness of the DHP are similar to our lower bound.

4 Vulnerable Systems

In some real world systems, an SDHP_d oracle is available. These are usually systems that do not use a key derivation function or its equivalent. The consequences of providing an SDHP_d oracle vary depending on the circumstances and the group used. We have seen that for random n, there is generally a factor $u \approx \sqrt[3]{n}$. In this case, u will be approximately optimal for reducing the number of scalar multiplication in the algorithm of Theorem 1.

In systems where one entity can serve as an SDHP_d oracle, then an adversary can use our algorithm to find the private key more quickly than by using generic DLP algorithms. To assess the cost of the attack, we model each query as taking C group operations, representing the situation that the adversary must wait for the entity to perform a scalar multiplication when answering an SDHP_d oracle query. We summarize this situation with the following result.

Corollary 3. *Assume the conditions of Corollary 2, except that $u = \lceil \sqrt[3]{n} \rceil$ and the cost of an SDHP_d oracle call is equivalent to C group operations in $\mathbb{G}$. Then the private key d can be found with computational cost about $3C\sqrt[3]{n}$ group operations in $\mathbb{G}$.*

Proof. Applying Theorem 1, the cost needed to find d is at most:

$$Cu + 2C(\lceil \sqrt{u} \rceil + \lceil \sqrt{v} \rceil) \tag{15}$$

group operations, where the first term Cu corresponds to the cost of the u oracle queries. Now note that $\sqrt{v} \approx \sqrt[3]{n}$. □

To see what this result means in practice, consider again the case that $n \approx 2^{160}$, and $u \approx \sqrt[3]{n}$. We again take $C \approx 40$. Then the cost needed to find d is equivalent to about 2^{61} group operations. Such a large number of queries may not be physically possible for such a u, though an adversary will still gain some benefit with smaller values of u.

Systems where an SDHP_d oracle is available could include certain smart card implementations. Sometimes a smart card design is highly constrained, so that all hashing in the key derivation is done outside the security boundary of the smart card, usually on the smart card reader. In this case, a malicious smart card reader could use the smart card as an SDHP_d oracle.

Such smart card designs could be considered as mere implementation errors, because security boundaries are not properly respected, but standards can also be to blame for this. Cryptographic standards committees cannot agree on key derivation functions. For example, ANSI, TLS, and IKE each uses a different KDF. An implementer of a private key token either has to separately implement each different KDF for each different application protocol that uses the token, or provide raw access to the SDHP_d oracle. Therefore, disagreement between standards increases the cost of a private key token or makes it less secure.

The negative consequences of Corollary 3 are not limited to implementation specifics. Some protocols also make available SDHP_d oracles. No implementation of such protocols could avoid the attacks described in Corollary 3. Some of these protocols are given below.

4.1 ElGamal Encryption

ElGamal encryption, in its original form (see [ElG85], [MvOV96, §8.4.2], [Sti95, §5.1] or [Sch96, §19.6]), makes an SDHP_d oracle available. ElGamal encryption of message m to an entity with public key g^d results in a ciphertext constructed as $c = (c_1, c_2) = (g^x, mg^{dx})$. In a chosen-ciphertext attack against an encryption scheme, an adversary can choose any ciphertext and receive the decryption of the chosen ciphertext. For ElGamal encryption, if this adversary chooses $c = (g^x, c_2)$, then the decryption is $m = c_2/g^{dx}$. The adversary can compute $g^{dx} = c_2/m$, which solves SDHP_d for instance g^x. Therefore with just a little arithmetic a chosen ciphertext adversary can use the victim as an SDHP_d oracle.

ElGamal encryption is already known to be vulnerable to chosen ciphertext attacks. The existing attacks mainly allow information to be learned about previously encrypted messages, but they do not leak anything about the private key. Finding d is considered a total break of the system, because all encryption is compromised. When using the victim as an SDHP_d oracle in this manner the private key d can be found with about $\sqrt[3]{n}$ cost, assuming the adversary knows a factor $u \approx \sqrt[3]{n}$. This cost requires about $\sqrt[3]{n}$ decryption queries to the victim, which may be too large to allow for a realistic attack.

One kind of countermeasure to our attack on ElGamal encryption is to use a key derivation function or to use symmetric encryption rather than multiplication to transform the message. Both these countermeasures of this kind are included in more modern variants of the DH-based encryption, such as DHAES and ECIES. The original rationale for this kind of countermeasure was essentially to exploit the security benefits of using symmetric encryption and a hash function.

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^{dx}, an entity who already knows x does not learn anything new, because g^{dx} is computable from the static public key g^d and the ephemeral private key x. The key to finding d in Corollary 3, is that the adversary chooses ephemeral public keys g^x for which it does not know the corresponding ephemeral private key x. The countermeasure is also part of DHAES and ECIES. Cramer and Shoup's public-key encryption scheme [CS98] 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.

4.2 Ford-Kaliski Key Retrieval

In the Ford-Kaliski key retrieval scheme [FK00], which is currently being standardized in [IEE04], an SDHP_d oracle is available. Part of this scheme requires the server to compute c^d for any value c sent to it by a client, where d is a long-term key (see Figure 2). The server provides an SDHP_d oracle to any client, so the security of d is potentially weakened by a malicious client using our attack. In theory, d can be recovered with computational cost of about $\sqrt[3]{n}$ steps by the adversary, which is significantly faster than existing generic algorithms for finding d.

The generator g_π in Figure 2 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

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

Figure 2: Ford-Kaliski Key Retrieval

because z is derived from a strong secret d kept by the server. If d is revealed, then the user becomes vulnerable to dictionary searches.

The groups used in [FK00] are $\mathbb{F}_p^*$ where p is a strong prime. As already noted in §2.1 , such groups make Corollary 3 irrelevant because index calculus attacks on the DLP in these groups are faster than $\sqrt[3]{n}$ group operations. However, in [IEE04], more general groups are allowed with this protocol, so our attack may be relevant when using some groups, such as elliptic curve groups.

A number of simple countermeasures are available to resist our attack on the Ford-Kaliski protocol. One is to limit the number of queries to the server. This keeps u small, which increase v and thus the cost of the attack, which will be dominated by about $\sqrt{v}$ scalar multiplications. This can be effective, but our attack reduces the computational cost of the adversary even with a moderately small number of queries, so long as the number is above a certain cutoff. Another countermeasure is to use the groups $\mathbb{F}_p^*$ where p is a strong prime, as done in [FK00], or to use some more efficient groups, such as some of those discussed in §2.1.

4.3 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 [CvA89]. In this scheme, the signer has private key d and computes her signature on message m as $s = m^d$. The DH groups suggested for use in [CvA89] are the same as in [FK00], namely $\mathbb{F}_p^*$ where p is a strong prime, and therefore are not affected by Corollary 3. Again, this is because they have the property that index calculus methods solve the DLP faster than our attack. If, however, instead of these groups, other groups were used, such as elliptic curve groups or DSA groups, which is reasonable because they make the scheme much more efficient, then our attacks becomes viable. A very simple countermeasure to this attack is to hash the message before exponentiating. Hashing does not interfere with the functionality.

5 Further Interpretations

Theorem 1 has more subtle interpretations than just an attack or an argument for the hardness of the SDHP. Some of the most important of these are discussed in this section, with less important issues reserved for the appendices.

5.1 Strengthening the Connection between the DHP and DLP

Maurer and Wolf [MW00, §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_d involves a *negligible fraction of instances*. So, according to the notion of strength that Maurer and Wolf set forth above, by considering fewer instances, we are establishing a *stronger form of converse statement*, namely a tighter connection between the DHP and DLP.

5.2 A Hierarchy of Hard Problems

The SDHP_d cannot be harder than the DHP, because a solver of the DHP can be used to solve the SDHP_d, 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_d, choose random z and let $g^x = (g^d)^z$. Then use $\mathcal{A}$ to solve the DHP with probability p of finding g^{xy}. The solution to the SDHP_d is $(g^{xy})^{z^{-1} \bmod n}$. We write $\mathrm{SDHP}_d \leqslant \mathrm{DHP}$, or $\mathrm{SDHP}_d \leqslant_P \mathrm{DHP}$ 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_d 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_d 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 about the comparison above of the SDHP_d and the DLP is that only one instance of the discrete logarithm can be solved, the instance g^d. This leads one to consider the *single-instance Discrete Logarithm Problem* (SDLP_d), in which one is given fixed g^d, with the task being to find d. Previous work [dB88, MW96, BL96] compared the DHP and DLP, while ours compares the SDHP_d and SDLP_d. Thus our work can be considered as *parallel* to previous work, in the sense that it works in the *single-instance setting.*

If we write $P \to Q$ to mean an algorithm that solves problem P can be used to build an algorithm that solves problem Q, which is to say $Q \leqslant P$, then the simple relations between the four problems can be illustrated as:

$$\begin{array}{ccc} \text{DLP} & \xrightarrow{\text{Exponentiation}} & \text{DHP} \\ {\scriptstyle\text{RSR}}\big\downarrow & & \big\downarrow{\scriptstyle\text{RSR}} \\ \text{SDLP} & \xrightarrow{\text{Exponentiation}} & \text{SDHP} \end{array} \tag{16}$$

The horizontal arrows are simple in the sense that an algorithm to solve the DLP or SDLP allows one to find a private key that can be used to solve the DHP or SDHP, respectively. The vertical arrows are simple in the sense that an algorithm solving the DLP or DHP problems on random inputs can obviously be used to solve for specific inputs, using the random self-reducibility property when the success is likely but not guaranteed, as already noted.

In addition to the simple relations above, there are more complicated relations in the opposite direction. We write $P \stackrel{*}{\to} Q$ to mean an algorithm that solves any random member of a family of problems P can be used to solve problem Q.

$$\begin{array}{ccc} \text{DLP} & \xleftarrow{\text{[dB88, MW96, BL96]}} & \text{DHP} \\ {\scriptstyle *}\big\uparrow & & \big\uparrow{\scriptstyle *} \\ \text{SDLP} & \xleftarrow{\text{Corollary 2}} & \text{SDHP} \end{array} \tag{17}$$

Such diagrams can help envision the relations between the four problems, and the relation between our work and previous work. However, they do not give the whole story, as described in §C.4.

5.2.1 Existential Easiness of the SDHP_d

The SDHP_d can be solved efficiently given only a small piece of information, the private key d. In other words, for a each d, there *exists* an efficient algorithm that solves the SDHP_d. In this sense the SDHP_d is an *existentially easy* problem. The DHP and the DLP are, on the other hand, is not existentially easy, as far as we know.

An existentially easy problem is not necessarily *practically easily*. If d is a random secret, it is hard to solve the SDHP_d. Otherwise, the DHP problem would be easy, and thus the DLP problem. Although existentially easy problems are not not usually considered, we have found it useful to consider such a problem. For further discussion of this issue, see §C.2.

5.3 Analogy with the Rabin Cryptosystem

The original Rabin cryptosystem [Rab79] has the property that duplicating the private key operation is almost as hard as finding the private key. This makes the original Rabin cryptosystem vulnerable to active attacks. The modern Rabin cryptosystem uses hash-based padding. This padding prevents the active attacks. Furthermore, it also makes security proofs in the random oracle model work against active attacks.

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

With DH based cryptosystems, until the pioneering work of den Boer [dB88] and others that followed, 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 den Boer's work and the subsequent work, DH cryptosystems were put on a better footing: they were based on a problem almost as hard as the DLP. In other words, they have brought DH cryptosystems up to the level of Rabin cryptosystems. Our work can be thought of as extending this effort. Specifically, we establish an individual entity's private key operation is almost as secure as the individual's private key.

Active attacks on the original Rabin cryptosystem are analogous to our Corollary 3 attacks on unprotected DH cryptosystems. Analogous to the modern padding mechanism for Rabin cryptosystems is 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. Until our work, the risks of such attacks on DH cryptosystems without such a protection mechanism was not well understood.

Some more technical details on this subject are given §C.3.

References

[BL96] 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.

[BSL01] D. Boneh, H. Shacham, and B. Lynn, *Short signatures from the Weil pairing*, Advances in Cryptology — ASIACRYPT '01, LNCS, vol. 2248, Springer, 2001, pp. 514–532.

[BV98] D. Boneh and R. Venkatesan, *Breaking RSA may not be equivalent to factoring*, Advances in Cryptology — EUROCRYPT '98, LNCS, vol. 1233, Springer, 1998, pp. 59–71.

[CS98] 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.

[CvA89] D. Chaum and H. van Antwerpen, *Undeniable signatures*, Advances in Cryptology — CRYPTO '89 (G. Brassard, ed.), LNCS, vol. 435, Springer, 1989, pp. 212–217.

[dB88] 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.

[ElG85] 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.

[FK00] 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.

[GJ03] E. J. Goh and S. Jarecki, *A signature scheme as secure as the Diffie-Hellman problem*, Advances in Cryptology — EUROCRYPT 2003, LNCS, vol. 2567, Springer, 2003, pp. 116–129.

[IEE04] IEEE P1363.2, *Standard specification for password-based public key cryptographic techniques*, IEEE Computer Society, May 2004, Draft version 15.

[Knu98] D. E. Knuth, *Seminumerical algorithms*, 3rd ed., The Art of Computer Programming, vol. 2, Addison-Wesley, 1998.

[LMQ+03] L. Law, A. J. Menezes, M. Qu, J. Solinas, and S. A. Vanstone, *An efficient protocol for key agreement*, Designs, Codes and Cryptography **28** (2003), no. 2, 119–134.

[MSV04] 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.

[MvOV96] 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.

[MW96] 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.

[MW00] U. M. Maurer and S. Wolf, *The Diffie-Hellman protocol*, Designs, Codes and Cryptography **19** (2000), 141–171.

[Rab79] 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.

[Sch96] B. Schneier, *Applied cryptography*, Wiley & Sons, 1996.

[Sti95] D. Stinson, *Cryptography: Theory and practice*, CRC Press, 1995.

[WFLY04] X. Wang, D. Feng, X. Liu, and H. Yu, *Collisions for MD4, MD5, HAVAL-128, and RIPEMD*, Tech. Report 199, IACR, 2004, See eprint.iacr.org/2004/199.

A Practical Considerations

A.1 Impact on Other Key Agreement Schemes

Ephemeral DH key agreement uses the DH private key only once. Alice and Bob send each g^x and g^y, respectively. To have security, they need the DHP to be hard. Technically, they also need the SDHP_x and SDHP_y to be hard, because if SDHP_x, say, were easy then an adversary compute the shared secret g^{xy}. Arguably, hardness of the SDHP is not as important as in static DH private key agreement, because x and y are only used to agree on one key, which is likely to have a limited life span. Furthermore, the algorithm used in Theorem 1 cannot be used effectively to attack ephemeral DH key agreement, because the adversary only gets one query to the SDHP oracle.

Menezes-Qu-Vanstone (MQV) key agreement [LMQ+03] enhances Diffie-Hellman key agreement. Each DH public key is constructed from two MQV public keys. The construction is $g^x = g^{x_1+f(g^{x_1})x_2}$, where f is a function from the group $\langle g\rangle$ to integers and g^{x_1} and the g^{x_2} are the MQV public keys. The second g^{x_2} is always static and the g^{x_1} is usually ephemeral. The MQV problem is to compute the MQV shared secret g^{xy} from the four MQV public keys g^{x_1},g^{x_2},g^{y_1}, and g^{y_2}.

Informally, at least, we can relate the hardness of the MQV problem to the DHP and to the SDHP. Fix x_1 and x_2, and consider an oracle that receives input g^{y_1} and g^{y_2} and gives output g^{xy}. Let this oracle be the MQV_{x_1,x_2} oracle. This oracle can be used to construct an SDHP_x oracle, as follows. Set $z = g^y$, and choose random h and compute $k = (z/h)^{1/f(h)}$. Send input (h, k) to the MQV_{x_1,x_2} oracle. The response will be $z^x = g^{xy}$.

Therefore, if the MQV problem is easy, then so are the DHP and SDHP. Conversely, if the SDHP_d is hard, then so is the MQV problem. If x_1 is static, which is possible for 1-Pass MQV, which sets $x_1 = x_2$, then the attack Corollary 3 is potentially applicable. Fortunately, most standards for MQV require a KDF, which prevents this attack.

A.2 Impact on Signature Schemes

We know of no impact of this work on the signature schemes DSA, ECDSA and Nyberg-Rueppel. These signature schemes are based on the ElGamal signature scheme, and use exponentiation of a fixed based point to an ephemeral private key.

Other signatures schemes, however, are based on different mechanisms. These signature schemes are based on static DH key agreement, and exponentiate a value derived from a message by a static private key. Such schemes include: Chaum and van Antwerpen's undeniable signatures [CvA89]; Boneh, Shacham and Lynn's short signatures [BSL01]; and Goh and Jarecki's [GJ03] short signatures. (Verification requires participation of the signer, a bilinear pairing, or a proof of knowledge, respectively.) As already noted in §4.3, the signer in such signature schemes risks becoming an SDHP oracle, which in turn risks exposure of the signing key. The more modern versions of these schemes [BSL01, GJ03] apply a hash to the message to form the base of the exponentiation. Thereby, they thwart the attack of using the signer as an SDHP oracle, unless the implementation does not apply the hash within the security boundary. Provided such hashing mechanisms are enforced, such schemes benefit from our result that SDHP_d is hard.

A.3 Pseudorandom Private Key

In principle, Alice will choose $d \in [1, n-1]$ uniformly at random. In the $\mathbb{G} \times \mathbb{G}$ matrix metaphor, Figure 1, Alice needs her row to be hard, that is, that the SDHP_d is hard.. If the row is random, however, then hardness of the DHP ensures that the row is hard with high probability. Alice gets some assurance from the hardness of the DHP. Recall that random self-reducibility of the DHP ensures that the rows cannot be too easy on average, unless the whole DHP is easy. Arguably, therefore, Alice may feel hardness of the DHP is enough to protect her.

In practice, however, Alice does not choose d uniformly at random. This is because it is difficult and expensive to do so. Generally, Alice chooses d pseudorandomly. Consider the following simplified situation, which is not dissimilar with common practice. Suppose that $n \approx 2^{160}$, and that Alice chooses d as follows. She applies SHA-1 to a truly random 80-bit seed. Suppose that Alice does this because her constrained system cannot afford to gather any more entropy in the allotted time. The question to ask is whether this is secure against adversaries with computational complexity less than 2^{80}.

Eighty bits of entropy is enough to resist exhaustive search. The fact that d is the output of a hash, means that d is pseudorandom. (We will further assume some additional processing is done to make d a pseudorandom integer in the interval $[0, n-1]$.) Pseudorandomness of d ensures that there is no way to accelerate generic DLP solving algorithms, such as Pollard-ρ, because otherwise d could be distinguished from random. Therefore, against known attacks, choosing d in this way seems secure.

Unquestionably, however, this key can only take 2^{80} values which is a negligible fraction of the whole key space of size $n \approx 2^{160}$. Hardness of the DHP does not ensure that the SDHP is hard. It is possible that the SDHP is easy for Alice's private key: a fast SDHP solver could be known for each of Alice's keys. Random self-reducibility between the DHP and the SDHP only produces a very slow DHP solver with complexity 2^{80}.

Corollary 2 establishes, for the first time, that Alice is assured that choosing her private key in this manner is safe, provided n has the right form, d is pseudorandom and non-exhaustible. This is not the ideal way of choosing d, but if it is the only way available on a certain system, then it is acceptable.

If speed of exponentiation is critical, then one may want d with a special form that makes exponentiation faster. Special forms automatically reduce entropy, but as long as d has enough entropy, it resists the main known attack, exhaustive search. Private keys of special form are not generally pseudorandom. Therefore relying on the hardness of the DHP is inapplicable. Some forms enable acceleration of generic DLP algorithms like Pollard-ρ and BSGS. Small size for d makes these algorithms run faster. Other non-pseudorandom forms, however, may have sufficient entropy to resist exhaustive search, permit exponentiation speedup, and not enable acceleration of the DLP algorithms. Hardness of the SDHP_d is especially important for such d. Again, such special forms are not the ideal, and should only be used when better forms are infeasible.

B Technical Considerations

B.1 Composite Order Groups

In most DH systems, the group order n is prime. A few systems use a composite n. When the group order n is composite, then a modified version of the reduction can be used. We sketch below how this would be done.

Suppose that $n = pq$ for relatively prime p and q. To solve the SDHP_d in $\mathbb{G}$, we solve the SDHP_{pd} in $p\mathbb{G}$, which is a group of order q. Define a function $\delta_p(X) = p\delta(X) = pdX$ on $p\mathbb{G}$, where δ is an SDHP_d oracle. Then δ_p is an SDHP_{pd} oracle in the group $p\mathbb{G}$. This oracle can be used to determine $pd \bmod q$. Similarly we can find $qd \bmod p$. The Chinese Remainder Theorem can then be used to find d. We can factor any n into relatively prime prime powers. By above, each relatively prime factor can be handled individually. Therefore, we just need to consider prime powers $n = p^m$.

When n is a prime power, then $\mathbb{Z}_n^*$ is cyclic. If $d \in \mathbb{Z}_n^*$, then our reduction works without modification, as it did for prime n. The remaining possibility is the degenerate case that $p|d$. Let $d = d_1 p + d_2 p^2 + \dots$. If p is small enough to permit exhaustive search, compute $\delta(g^{p^{m-2}}) = g^{d_1 p^{m-1}}$ and search for all values of d_1. Once d_1 is known, a similar technique can be used to find d_2, and so on. If p is too large to permit exhaustive search, which might be the case if $m = 2$, then define $\delta'(x) = x\delta(x)$, and use δ' to solve the SDHP_{d+1}, which can be done by the method above because $p \nmid d + 1$.

B.2 Using Pollard-ρ in the Reduction

For simplicity, the proof of Theorem 1 uses the BSGS algorithm to find d. Other generic algorithms, such as Pollard-ρ can also be used. For example, we could compute the element E as in the proof, and then determine the value y by iterating a function:

$$\phi(X) = \begin{cases} w^{\alpha(X)}G, & \text{if } \beta(X), \\ w^{\alpha(X)}E, & \text{otherwise}, \end{cases} \tag{18}$$

where $\alpha(X)$ and $\beta(X)$ are deterministic, pseudorandom integer and boolean valued functions, respectively. The function ϕ acts on the space $\langle w \rangle G$ of size v, and has a good chance of repeating after about $\sqrt{v}$ iterations.

Once an iteration repeats, we will have found X and Y such that $\phi(X) = \phi(Y)$. If $\beta(X) \neq \beta(Y)$, then then we can solve

$$E = w^{\pm(\alpha(X)-\alpha(Y))}G \tag{19}$$

where the sign depends on $\beta(X)$. If $\beta(X) = \beta(Y)$, then we must start over, iterating from a new starting point. We can find the value of t similarly.

As with traditional Pollard-ρ algorithms, large amounts of storage can be avoided. One method, Floyd's cycle-finding algorithm, is to remember only two elements of G: $\phi^t(X_0)$ and $\phi^{2t}(X_0)$, where X_0 is the starting point. Another is to use distinguished elements, which is also an effective means for parallelizing a part of the algorithm. The running time of the reduction is slightly different when Pollard-ρ is used. The running time is probabilistic, it can vary considerably. It averages to a constant factor times that of the version using the BSGS.

B.3 Choosing Custom Groups to Make Reductions

Because of our results and den Boer's result on the relationship between the DHP and DLP, one may want to choose a group $\mathbb{G}$ of prime order n such that $n-1$ admits factorizations of a certain form. For example, to apply Corollary 2, one wants $n-1 = uv$, with u of a certain size. Similarly, to apply den Boer's reduction, we want $n-1$ to be smooth.

To avoid Corollary 3, one wants to avoid such factors u of a certain size. Generally these attacks can be avoided by other means, such as a by using a key derivation function, but in some special protocols, such as the Ford-Kaliski protocol, there is no other obvious way to avoid these attacks. In this case, a smooth factorization of $n-1$ is not good, because it ensures a factor u that optimizes the attack. However, it is possible to have a factorization of $n-1$ such that Corollary 2 is applicable but Corollary 3 is not.

B.3.1 Mod p Groups

It is relatively easy to find groups $\mathbb{G}$ of prime order n that are subgroups of $\mathbb{F}_p^*$ for prime p, such that $n-1$ has a factorization of a particular form. This is done by choosing n first, then choosing $p = tn+1$ for some t. Furthermore, for any such n, one can choose t such that p has a form that has low Hamming weight, which means it is somewhat more efficient for exponentiation.

B.3.2 Elliptic Curve Groups

The complex multiplication method constructs elliptic curves of a given order n. One can start from an arbitrary $n-1$, pick some arbitrary small or medium discriminant D, and then find for a prime p defining the coefficient field $\mathbb{F}_p$ of the elliptic curve. Both p and n need to be prime, so several different choices for $n-1$ and D will generally need to be tried. The method above results in less than ideal primes p, however, because the primes are more or less random distributed in the Hasse interval of n. For faster implementations, a field size p of special structure can help.

To find such a special form for p, one can search for p and n simultaneously. This is made easier if one allows a cofactor $h < 1$. The following parameters

$$p = 2^{288}55 + 9, \quad n = 2^{286}55 + 1, \quad h = 4 \tag{20}$$

admit curves of discriminant $D = 55$. Coefficients of the curve can be computed using the CM method as:

$$\begin{aligned} a &= 27228472517654392839423150699310359352727681288846049469547882780614298246794702605839963 \\ b &= 19862846360388611577546848247471636335510997190076544762888002392556099227817390151460715 \end{aligned} \tag{21}$$

This curve has a smooth factorization for $n-1$, so den Boer's reduction applies as does Corollary 2. If one also wants to avoid the attack of Corollary 3, say for implementation of the Ford-Kaliski protocol, then one would apply similar techniques with a different shape of the factorization of $n-1$.

B.4 Unknown Secret n

When n is an unknown secret, then at least one of u and v would also be an unknown secret. Moreover, it may also be an unknown secret whether u and v of certain sizes exist. The reductionist proof of security in Corollary 2 theoretically applies if u and v merely exist, although knowledge of u and v can be considered as a definitive certification that the corollary is applicable. An attacker using Corollary 3, however, definitely needs to know u and v, because the values u and v are used explicitly in the algorithm used to prove Theorem 1.

When n is secret, it is still possible for u to be publicly known, because v can remain secret. In this case, the reduction algorithm can be modified. The second application of the BSGS uses scalar multiplication by $q = g^v$, which cannot be computed directly now, because v is secret. However, if the first of part of the algorithm is repeated with d replaced by a known value d', then one learns some y' corresponding to y and one can then compute $f' = g^{-y'}d'$, which equals $q^{t'}$ for some t'. (A random value g has a good chance of being a generator.) Let $q' = f' = q^{t'}$, and there's a good probability that q' also generates $\langle q \rangle$. Now run the BSGS algorithm on F using q' instead of q, where k is based on an estimate of the size of v. We then learn that $f = (q')^{h+lk}$, which allows us to recover d. Therefore, while knowledge of u is fundamental to our results, knowledge of v is not really necessary.

A system with a secret n secret is RSA, where the public key N is used to define the group $\mathbb{Z}_N^*$ of secret order $\phi(N)$, where ϕ is Euler's totient function. Typically, the group $\mathbb{Z}_N^*$ is the direct product of a large cyclic group (of composite order) with a small cylic group. The basic RSA private key operation is essentially exponentiation by a private key d. Therefore the SDHP$_d$ needs to be hard in the RSA key system. Furthermore, an RSA private key holder potentially provides access to an SDHP$_d$ oracle, if the basic private key operation is unprotected. If a factor u of the largest cyclic group is made public, then our results may be applicable. However, with RSA, index calculus methods can generally find d faster, so our results may not be applicable to RSA.

C Theoretical Considerations

C.1 Making the Weakest Assumptions Possible

Because the SDHP$_d$ is a potentially easier problem than the DHP, hardness of the SDHP$_d$ is a stronger assumption than hardness of the DHP. In reductionist security, it is always desirable to make the weakest assumptions possible. Hence, to prove that a particular cryptographic protocol is secure, it is preferable to only assume that the DHP is hard, instead of relying on the assumption that the SDHP$_d$ being hard. Nevertheless, as discussed above, if security specific down to the level of a single public key is desired, rather than an average over random keys, then hardness of the SDHP$_d$ is a necessary assumption for security of some protocols. Our results establish that directly *assuming* hardness of the SDHP_d is not necessary, because it can be deduced from a more basic assumption that finding d is hard.

C.2 Existential Easiness of the SDHP

In theoretical computer science, the hardness of a problem can be measured by the minimum running time of any program for a universal Turing machine that solves the problem. Problems for which it is known that there *exist* algorithms that are fast are generally considered easy in theoretical computer science. This is considered to be true even if the only known algorithms are slow.

We believe that in cryptography, there are important problems that would be considered easy under this theoretical computer science measure. In order to distinguish the practical cryptographic notion from the theoretical computer science notions, we call such problems *existentially easy*. (Conversely, a problem for which there does not exist a fast solution can be called *universally hard*.)

The SDHP_d is existentially easy because there exists a relatively efficient algorithm, namely exponentiation by d, that solves the problems. Cryptographers like to believe that problems like the DLP and DHP are not existentially easy. This means any algorithm that solves them takes a long time when run on a universal Turing machine. (Table-lookup algorithms solve the DLP and DHP quickly, but not on a universal Turing machine because the table must be included in the program and searched by the universal Turing machine.) It is possible that these problems are universally hard, but nobody has proven it. The closest results towards showing that these problems are universally hard are that the *generic group* algorithms for solving these problems are slow. Generic group algorithms are those only involve the group via oracles for its operations.

Cryptographers have considered existentially easy problems before. Finding a collision in a fixed hash function such as SHA-1 or MD5 is existentially easy. For this reason, many theorists prefer to consider the hash collision problem as defined only for a *family* of hash functions rather than a single hash function. But because a fixed hash function will get used in practice, the collision problem for a fixed hash function is very important, even though this is an existentially easy problem. The recently discovered collisions in MD5 [WFLY04] have attracted much attention, even though this is just a solution to an existentially easy problem.

C.3 More Comparison with the Rabin Cryptosystem

The Rabin [Rab79] cryptosystem works essentially as follows (see also [MvOV96, §11.3.4]). We describe the original version, which is *raw* in the sense that further processing such as hashing is applied to the inputs. An entity chooses a private key (p, q), which is a pair of primes, and a corresponding public key $N = pq$. The raw public key operation on input x is to compute $y \equiv x^2 \bmod N$. The raw private key operation on input y is one of the four square roots of y modulo N, if one exists. (The Rabin-Williams cryptosystem is a deterministic variation of the Rabin system.)

C.3.1 Necessary Underlying Hard Problems

Similarly to the raw RSA public key and private key operations, the Rabin operations can be used for digital signatures and asymmetric encryption. For several reasons, both raw Rabin and raw RSA are combined with complementary padding techniques before being applied to signatures or encryption.

Unlike RSA, the Rabin cryptosystem has the property that the raw private key operation is as hard as finding the private key. Finding the private key means factoring N. Therefore the underlying hard problem for the Rabin cryptosystem is integer factorization.

For the RSA cryptosystem, the underlying hard problem is the RSA problem, which is potentially easier than integer factorization. In fact, Boneh and Venkatesan [BV98] have done some work towards showing that the RSA problem could be weaker than integer factorization, at least that nobody will be able to prove their equivalence. To have security in any RSA cryptosystem, it is necessary to assume that the RSA problem is hard.

Because the integer factorization problem has been extensively studied, but the RSA problem has not, it is acceptable to rely on the hardness of the integer factorization but not acceptable to rely on the hardness of the RSA problem. Comparatively little study has gone into the RSA problem as a distinct problem the integer factorization problem. The advantage of the Rabin system over RSA is that it does not to rely on the hardness of such an under-studied problem.

Similarly, for a time, DH key agreement and other cryptosystems based on DH key agreement had to rely to on the hardness of an under-studied problem, namely the DHP. The DLP had been and continues to be a far better studied problem. Therefore, it is preferable not to make rely on the DHP if at all necessary. Fortunately, good work was done towards establishing that the DHP was almost as hard as the DLP. This gave systems based on DH key agreement a more stable foundation, in the same sense that the Rabin cryptosystem has a more stable foundation than the RSA cryptosystem.

Our work strengthens this foundation beyond the previous work, further approaching the stability of the Rabin foundation. In the Rabin cryptosystem, the security reduction is specific down to the level of single private key. If the Rabin private key operation is breakable for a single private key, then that private key must be findable. Previous work on the DHP did not apply to the level of a single private key, but our work does.

C.3.2 Modern Day Security of the Rabin Cryptosystem

In the modern notion of public-key cryptosystem security, resistance to active attacks is necessary. An entity's private key operations should not expose the private key. The adversary should not be able to exploit in any way at all an entity's private key operation if the adversary can choose the inputs. This fundamental principle applies to both digital signature schemes and to asymmetric encryption schemes.

The raw Rabin cryptosystem does not achieve this modern notion of security because the private key operation exposes the private key if the adversary chooses the inputs appropriately. Therefore modern variants of the Rabin cryptosystem apply further processing to the inputs before applying the raw Rabin public-key and private-key operations. The further processing usually involves a secure hash function.

In the Rabin cryptosystem, the further processing prevents the adversary from selecting inputs in such way that the private key is exposed. In the RSA cryptosystem, the raw private key operation is multiplicative, so further processing is necessary to prevent attacks where the adversary tampers with the inputs in such a way to apply the private key operation that the private key owner does intend. In Diffie-Hellman key agreement, the further processing done to the raw operations is a key derivation function, which as a pseudorandom function, makes the shared secret indistinguishable from random, which is necessary for modern security definitions. Our algorithm demonstrates another reason reason further processing is necessary, which is to prevent eventual exposure the private key.

The reduction algorithms for Rabin and static Diffie-Hellman only become attacks when the further processing is omitted. The further processing helps ensure that the reduction algorithms cannot be used as attacks. This is because the further processing is a one-way function. The process private key operation does not give access to the raw private key operation. Therefore, both the Rabin and static Diffie-Hellman cryptosystems benefit from the private key operation being hard as finding the private key, and having no known attacks.

C.3.3 Provable Security of the Rabin Cryptosystem

The Rabin cryptosystem is provably secure, when the hash functions used in the further processing are modeled as a random oracle. In a security proof for a cryptosystem, a reduction is used to show that an adversary could be used as an oracle to solve some underlying problem. Because active adversaries must be considered, such an oracle must provide a simulation of the entity owning the private key.

In the random oracle model, simulating the private key operation for the adversary is sometimes quite straightforward. Being able to simulate the private key operation, without the private key, can seem paradoxical, but in the random oracle model it can done because the private key operation depends on both the private key and the random oracle. In the simulation, the random oracle is modified in such a way as to be fully random, yet to have its answer pre-suited to make the simulated private key operations correct. The adversary cannot detect these modification to the random oracle. This apparent paradox is somewhat similar to the apparent paradox that the algorithm of Theorem 1 can be used both as an attack and as reduction that establishes that one problem is almost as hard as another.

C.4 A Looser DHP-DLP Reduction and Subtlety of Diagrams

The algorithm in Theorem 1 can also be used to show that the DHP is almost as hard as the DLP. To make this reduction, suppose that we are given a DHP oracle. Fix one of the inputs to the DHP to be g^d. Now apply the algorithm to solve the DLP. This reduction is much looser than Maurer and Wolf's reduction because the DLP is solved in exponential time, not polynomial time. Therefore, our result can also be regarded as an alternate reduction between the DHP and DLP. The reduction is looser, but does not depend on auxiliary groups. Instead, it depends on a factorization of $n-1$.

Consider the following two very similar diagrams:

$$
\begin{tikzcd}
\text{DLP} & \text{DHP} \arrow[l, "!!!"'] \arrow[d, "\text{RSR}"] \\
\text{SDLP} \arrow[u, "*"] & \text{SDHP} \arrow[l, "\text{Corollary 2}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\text{DLP} \arrow[d, "\text{RSR}"'] & \text{DHP} \arrow[l, "{[\text{dB88, MW96, BL96}]}"'] \\
\text{SDLP} & \text{SDHP} \arrow[l, "???"'] \arrow[u, "*"']
\end{tikzcd}
\tag{22}
$$

The left diagram correctly suggests that Corollary 2 can be used to establish DHP $\rightarrow$ DLP via the traversal DHP $\rightarrow$ SDHP$_d$ $\rightarrow$ SDLP$_d$ $\overset{*}{\rightarrow}$ DLP, as noted above.

The right diagram is misleading because it incorrectly suggests that [dB88, MW96, BL96] can be used to show SDHP $\rightarrow$ SDLP via the traversal SDHP$_d$ $\overset{*}{\rightarrow}$ DHP $\rightarrow$ DLP $\rightarrow$ SDLP$_d$. This reduction happens to be invalid, because the reductions in [dB88, MW96, BL96] make fundamental use of a two-sided DHP oracle, which is not applicable when d is made static. Evidently, some subtlety is involved when determining whether an arrow $\overset{*}{\rightarrow}$ can be reasonably traversed.

D Hypothetical Considerations

D.1 Quantum Computers and Attacking the DLP

When an adversary has a very powerful oracle like a DHP oracle, then the adversary can solve hard problems like the DLP, as previous work has shown. Although solving the DLP would break the security of the system, nobody considers this an attack, because DHP oracles are not known to exist.

Our work, however, can be interpreted both as a positive security and as an attack, because SDHP are known to exist. If some algorithm were found to solve the DHP, then work such as Maurer and Wolf's becomes an attack on the DLP. A setting in which such an attack could be plausible is quantum computing.

The DLP can be solved in polynomial time on a quantum computer. Unfortunately, the best quantum computers so far have just 7 qubit registers, while the algorithms for solving the DLP require a register of length approximately $5\log_2(n)$ qubits.

Quantum algorithms for solving the DHP have not been as well studied as those for the DLP. If algorithms that use sufficiently fewer qubits are found, then the results of den Boer, Maurer, Wolf, Boneh and Lipton become attacks on the DLP, rather than positive security results.

D.2 Probabilistic and Faulty SDHP oracles

For simplicity, we have assumed throughout that the SDHP oracle always gives the correct answer for any input. Another possibility is that the oracle only provides an answer with a certain probability, but any answer it does provide is correct. Such an oracle can be converted to the type we have assumed using random self-reducibility, albeit it will be a slower oracle. Yet another possibility is that the oracle provides false answers with a certain probability. For DHP oracle, there are techniques for self-correction, meaning one can be convert such a faulty oracle into a correct one. We have not considered whether faulty SDHP can similarly be self-corrected, but suspect that they can be.

D.3 On Trapdoor One-Way Permutations

Recall that a trapdoor one-way permutation E is a one-way function, with a trapdoor inverse. A trapdoor inverse is an efficient algorithm for computing the inverse function D, but which is difficult to reconstruct from the algorithm specifying E alone. The most famous example of a trapdoor one-way permutation is RSA function.

Consider the following generalization of the RSA function. Let $E(y) = y^e$, where exponentiation is performed in some abelian group of order n. The inverse function is $D(y) = a^{d^{-1} \bmod n}$. This generalizes the RSA function, because we can set $n = (p-1)(q-1)$ where the group is $\mathbb{Z}_{pq}^*$ for primes p and q. The RSA public key is (N, e), where $N = pq$. In the case of RSA, D is believed to be trapdoor inverse, and because N and e are public, the function E is easily computed. Therefore, in the special case of RSA, the function E is a one-way trapdoor permutation.

The main property of the RSA function that makes it is usable for public-key encryption and digital signatures is that it is a trapdoor one-way permutation. Any other trapdoor one-way permutation could be used in essentially the same way as RSA to achieve public-key encryption and digital signatures. In RSA, the group order n is secret and e is public. If both n and e are public, then $d^{-1} \bmod n$ can be computed, which makes D is easy to compute and thus means E is not a one-way trapdoor permutation. It appears that one of n or e must be secret for E to have a trapdoor. We now consider the opposite case to RSA: namely, when n is public but e is secret. We investigate if it is possible for such E to be trapdoor one-way function.

For simplicity, we assume that n is prime. Suppose that E was a trapdoor one-way permutation. Because the function E is easily computed, it can be used as an oracle for the SDHP_e. We have shown in Corollary 2, that such an oracle allows e to be found with a computation of about $\sqrt[3]{n}$ steps. Once e is found, then D can be computed. If $\sqrt[3]{n}$ is a feasible amount of computation for the adversaries one is concerned with, then E is not a one-way trapdoor permutation. Another way to say this is that it is generally possible to find the exponent of an easily computable exponentiation function.

The discussion above assumes $u \approx \sqrt[3]{n}$, but obviously there are groups in which this is not the case, such as when $n = 2m + 1$ for prime m. We have not ruled out E being a trapdoor one-way permutation in such groups. Such an E needs to have an efficient algorithm that does not reveal e, otherwise it would not be useful. We formally propose some problems about constructing such E:

Problem 3. *Construct a trapdoor one-way permutation E of the form $E(x) = x^d$ for a group of known prime order n.*

It can be especially useful to have such functions on groups whose elements have small representations, because they consume less bandwidth and potentially offer other efficiencies. Therefore, we formally propose an extension to the previous problem:

Problem 4. *Find an E as in Problem 3, where the best known algorithms to solve the DLP in the group are not much faster than $\sqrt{n}$ group operations and the group elements can be represented as elements of bit size approximately* $\log_2(n)$.

With a solution to Problem 4, we would have an RSA-like cryptosystem where the plaintexts, ciphertexts, and signatures are elliptic curve points, which are generally much smaller than the integers used in RSA. Signatures in this instance would be half the size of an ECDSA signature, or about the same size as Boneh, Shacham and Lynn's short signatures [BSL01].

One candidate is code obfuscation, applied to an ordinary implementation of exponentiation by e. Code obfuscation, though, has known limitations, so a more sophisticated method would be desirable. Besides, if code obfuscation is to be used, then E could be a symmetric cipher.

A more sophisticated candidate for Problem 4 is complex multiplication on elliptic curves. Unfortunately, in the simple cases that we understand, it is generally easy to determine the exponent d from the algebraic description of complex multiplication.