Solving Discrete Logarithms from Partial Knowledge of the Key

K. Gopalakrishnan[1,⋆], Nicolas Thériault[2,⋆⋆], and Chui Zhi Yao[3]

[1] Department of Computer Science, East Carolina University, Greenville, NC 27858
[2] Instituto de Matemática y Física, Universidad de Talca, Casilla 747, Talca, Chile
[3] Department of Mathematics, University of California - Riverside, CA 92521

Abstract. For elliptic curve based cryptosystems, the discrete logarithm problem must be hard to solve. But even when this is true from a mathematical point of view, side-channel attacks could be used to reveal information about the key if proper countermeasures are not used. In this paper, we study the difficulty of the discrete logarithm problem when partial information about the key is revealed by side channel attacks. We provide algorithms to solve the discrete logarithm problem for generic groups with partial knowledge of the key which are considerably better than using a square-root attack on the whole key or doing an exhaustive search using the extra information, under two different scenarios. In the first scenario, we assume that a sequence of contiguous bits of the key is revealed. In the second scenario, we assume that partial information on the "Square and Multiply Chain" is revealed.

Keywords: Discrete Logarithm Problem, Generic Groups, Side Channel Attacks.

1 Introduction

The discrete logarithm problem (DLP) is an important problem in modern cryptography. The security of various cryptosystems and protocols (such as Diffie-Hellman key exchange protocol, ElGamal cryptosystem, ElGamal signature scheme, DSA, cryptosystems and signature schemes based on elliptic and hyperelliptic curves) relies on the presumed computational difficulty of solving the discrete logarithm problem. For a survey of the discrete logarithm problem, the reader is referred to [13].

However, even if the DLP is indeed difficult to solve, one has to take other aspects into account in practical implementations. If proper countermeasures are not used, side-channel attacks could be used to reveal partial information about the key. In this paper, we address the problem of how to utilize the partial information effectively when solving the DLP.

⋆ This work was done in parts while the author was at the Institute of Pure and Applied Mathematics, UCLA.

⋆⋆ This work was done in parts while the author was at the Fields Institute, Toronto, Canada.

K. Srinathan, C. Pandu Rangan, M. Yung (Eds.): Indocrypt 2007, LNCS 4859, pp. 224–237, 2007.
© Springer-Verlag Berlin Heidelberg 2007

When one wants to break a system based on DLP, one can of course, ignore the partial information revealed by side channel attacks and simply use a generic algorithm. Alternatively, one can use an exhaustive search using the partial information made available to us. The primary question that we address in this paper is whether we can do something in between? i.e., can we use the partial information revealed by side channel attacks in an intelligent way to break the system?

In some cases, side channel attacks could reveal a string of contiguous bits of the secret key. In this situation, it is always possible to adapt Shank's baby-step giant-step algorithm [19] to perform the search in the remaining possible keyspace; However the memory requirements could make this approach impractical. For example, if 100 bits remain to be identified, computing to the order of 2^{50} group operations can be considered reasonable, but handling (and storing) a table of 2^{50} entries is much more problematic. To avoid this issue, we need a different algorithm, not necessarily deterministic, which has a lower memory requirement.

A number of papers address the question when a large number of observations are available [6,11,12,9]. When only one observation is possible, probabilistic algorithms are known, but they usually assumed that the known bitstring is either in the most or the least significant bits of the key [15,16,23]. In Section 3, we look at what happens when a contiguous sequence of bits is known somewhere in the middle of the binary representation of the key.

In most cases, side channel attacks will reveal information on the square and multiply chain (see the beginning of Section 4 for a definition), and not the bitstring. Extracting the key from partial information on the square and multiply chain requires different approaches than those used when some of the bits are known. In this situation, no "fast" algorithm is known, no matter what the memory requirement is, hence any "fast" algorithm can be considered an improvement.

If uniform formulas are used for the group arithmetic (see [2,1] for example), then a side channel attack will reveal the hamming weight of the key, but not the position of the nonzero bits. If the hamming weight is low enough, fast algorithms are available [22,3], although they can be slower than general searches if the hamming weight is even moderately high. If the field arithmetic is not secured as well, some parts of the square and multiply chain may also be leaked [24,21].

In that situation, no algorithm was known that could improve on the exhaustive search from the partial information, or a search based solely on the hamming weight (note that the two approaches are not compatible). In Section 4, we will show how to significantly improve on the exhaustive search in this context.

2 Background

First, we define the discrete logarithm problem as follows: Let G be a cyclic group of prime order p. Let g be a generator of G. Given $\beta \in G$, determine $\alpha \in \{0, 1, 2, \ldots, p-1\}$ such that $g^\alpha = \beta$. Here, g and p are public information

known to everybody. Although our description is in terms of a multiplicative group, all of the arguments in this paper are essentially identical when applied to additive groups (for example groups coming from elliptic curves).

It is also possible to define the DLP on groups whose order n is not a prime number. However, one could then use the well known technique due to Pohlig and Hellman [14], and reduce the problem to a number of DLPs in groups of order p, where p runs through all the prime factors of n. Hence, without any loss of generality, we will focus on the case when the order of the group is a prime number.

2.1 Generic Algorithms for Solving DLP

In this paper, we only consider *generic algorithms* for solving the DLP. A generic algorithm for solving the DLP is an algorithm that does not exploit the structure of the underlying group in solving the DLP. As a consequence, this algorithm could be used to solve the DLP in any group.

In a generic algorithm, we want to think of the group as though it is presented by means of a black box. More specifically, each group element has a unique encoding or labeling and we have an *oracle* (a black box) which is capable of doing the following things:

- Given the encoding of two elements g and h of the group, the oracle can compute their product $g * h$, in unit time.
- Given the encoding of two elements g and h, the oracle can decide whether $g = h$, in unit time.
- Given the encoding of an element g, the oracle can compute any given power of g (including the inverse g^{-1}), in time $O(\log p)$.

We also note that in some groups, for example those coming from elliptic curves, the inverse operation can be performed in unit time. The time complexity of a generic algorithm is determined by counting the number of times it needs access to the black box.

There are a few well-known generic algorithms to solve the DLP. The baby-step giant-step method due to Shanks [19] is a deterministic generic algorithm that can solve the DLP in time $O(p^{1/2} \log p)$ using space $O(p^{1/2} \log p)$. This algorithm is based on a time-memory trade off technique. The rho method due to Pollard [15,16] is a probabilistic generic algorithm that can solve the DLP in expected running time $O(p^{1/2})$ (under certain assumptions) using only $O(\log p)$ amount of space (requiring the storage of a constant number of group elements), and is based on the birthday paradox. The space efficiency of this algorithm makes it more attractive in comparison to Shanks' method. For an excellent survey of the state of the art in these two methods, the reader is referred to [23].

Victor Shoup [20] established a lower bound of $\Omega(p^{1/2})$ on the time complexity of any probabilistic (and therefore on any deterministic) generic algorithm that can solve the DLP. Hence, both the baby-step giant-step method and the rho method are essentially optimal algorithms with respect to their time complexity and can only be improved in terms of constant factors.

It should be noted that the fastest known algorithms that can solve the DLP for most elliptic curve groups are generic algorithms and thus of exponential complexity (note that the size of the input is $\log p$, whereas the algorithm has $O(p^{1/2})$ complexity). In contrast, subexponential algorithms exist for the factoring problem which is the basis for the RSA cryptosystem. As a consequence, cryptographers believe that elliptic curve based cryptosystems can provide better security (provided the curves and the parameters are chosen appropriately). This is the reason for increasing interest in elliptic curve based cryptography.

2.2 Side Channel Attacks

A side channel attack (on a cryptosystem or a signature scheme) is an attack that focuses on the physical implementation of the algorithm as opposed to the specification of the algorithm. By observing an implementation being executed, an attacker can make correlations between the events that occur in the processor and the data being processed. The first well known versions of side channel attack were based on timing [7] and power [8] analysis (and more recently EM analysis [4,17]).

In timing analysis based attacks, an attacker uses the execution timings to infer about the flow of control and thus about the key. For example, an implementation might take longer to run if a conditional branch is taken than if it is not taken. If the branch is taken or not depending on a bit of the secret key, then the attacker might work out the corresponding bit of the secret key.

In power analysis based attacks, an attacker uses the amount of power consumed by a processor to infer what operations are being performed and thus about the key (EM based attacks use a similar approach on the electromagnetic trace produced by the processor). For example, a multiplication operation would have a distinct power usage profile and will differ considerably from the power usage profile of a squaring operation. The attacker can then use that knowledge to break the system.

The interesting thing about side channel attacks is that they do not contradict the mathematical security provided by the system (even assuming the underlying computational problems are provably difficult to solve) but they simply bypass it.

3 Scenario I – Contiguous Bits of the Key Is Revealed

In this section, we deal with the scenario where the partial information revealed is a sequence of contiguous bits of the key.

Let G be a cyclic group of prime order p and let g be a generator of G. Given $\beta \in G$, recall that the Discrete Logarithm Problem (DLP) is to determine $\alpha \in \{0, 1, 2, \ldots, p-1\}$ such that $g^\alpha = \beta$. In this section, we assume that a sequence of contiguous bits of α is revealed by side channel attacks. Although a variation of the baby-step giant-step method is always possible, we are looking for an algorithm with memory requirements similar to the rho method, i.e. of size $O(\log p)$.

There are three possible cases to consider; the sequence of contiguous bits known may be in the left part, right part or somewhere in the middle (i.e. located away from the extremities, but not necessarily centered). The first two cases are known results and can be found in Appendices A and B. The third case does not appear to have received as much attention, and a new method to approach it is presented below.

3.1 Case III – Middle Part

Let us assume that we have some positive integers M and N such that we can write α in the form

$$\alpha = \alpha_1 MN + \alpha_2 M + \alpha_3 \tag{1}$$

where $0 \leq \alpha_2 < N$ is known and with $0 \leq \alpha_3 < M$. We also assume that $0 \leq \alpha_1 < p/MN$, i.e. that α is reduced modulo p. Note that α_1 is really bounded by $\left\lfloor \frac{p-\alpha_2 M}{MN} \right\rfloor$, but the error introduced is insignificant (a difference of at most 1 on the bound, which vanishes in the O-notation), and it makes the analysis easier to read. In terms of exhaustive search, we have to search through a set of size p/N ($\lfloor (p - \alpha_2 M)/N \rfloor$ to be exact), which requires $O(p/N)$ calls to the oracle, whereas a generic attack on the whole group would require $O(p^{1/2})$ calls to the oracle.

In practice we may be more interested in the case where M and N are powers of 2 – i.e. where $N = 2^{l_2}$ and $M = 2^{l_3}$, so the first l_1 and the last l_3 bits are unknown (we assume that $l_1 + l_2 + l_3$ is the bitlength of the key) – but the arguments presented here will hold for any positive integers N and M.

For now, let us assume that we are given an integer r, $0 < r < p$, such that we can write rMN as $kp + s$ with $|s| < p/2$. We will discuss how to choose r in a few paragraphs. Multiplying both sides of Equation (1) by r, we get

$$\begin{aligned} r\alpha &= r\alpha_1 MN + r\alpha_2 M + r\alpha_3 \\ &= \alpha_1 kp + s\alpha_1 + r\alpha_2 M + r\alpha_3 \\ &= \alpha_1 kp + r\alpha_2 M + \alpha', \end{aligned} \tag{2}$$

where $\alpha' = s\alpha_1 + r\alpha_3$. Raising g to both sides of Equation (2), we get

$$\begin{aligned} g^{\alpha r} &= g^{\alpha_1 kp + r\alpha_2 M + \alpha'} \\ (g^{\alpha})^r &= (g^p)^{\alpha_1 k} g^{r\alpha_2 M} g^{\alpha'} \\ \beta^r &= g^{r\alpha_2 M} g^{\alpha'}. \end{aligned} \tag{3}$$

Denoting $\left(\beta \times g^{-\alpha_2 M}\right)^r$ by β', Equation (3) can be written in the form $\beta' = g^{\alpha'}$. Note that β' can be computed from β as r, α_2, and M are known. We can then view determining α' as solving a DLP. When s is positive, $\alpha' = \alpha_3 r + \alpha_1 s$ must be in the interval

$$\left[0, r(M-1) + s\left(\frac{p}{MN} - 1\right)\right],$$

on which we can use Pollard's kangaroo method. Similarly, if s is negative we must consider the interval

$$\left[s\left(\frac{p}{MN}-1\right), r(M-1)\right].$$

In both cases we can restrict the value of α' to an interval of length $rM+|s|\frac{p}{MN}$.

To minimize the cost of the kangaroo method, we must therefore choose $r>0$ to minimize the value of

$$rM+|s|\frac{p}{MN} \tag{4}$$

under the condition $s \equiv rMN \bmod p$.

Although it is not possible in general to choose r (and s) such that (4) is of the form $O(p/N)$, some situations are more favorable than others.

A perfect example (although a rather unlikely one) occurs when working in the bitstring of a Mersenne prime $p=2^l-1$ (with $N=2^{l_2}$ and $M=2^{l_3}$), in which case we can choose $r=2^{l_1}=2^{l-l_2-l_3}$ and $s=1$ and we get an interval of length $O(2^{l_1+l_3})$. Similarly, if the difference between p and 2^l is of size $O(2^{l_3})$, we can set $r=2^{l_1}$ and $s=2^l-p$ and obtain an interval of length $O(2^{l_1+l_3})$. In both of these situations, using Pollard's kangaroo method would allow us to compute α' in time $O(2^{(l_1+l_3)/2})=O(\sqrt{p/N})$.

Unfortunately, such an optimal choice of r (and s) is impossible in general. We will now consider how to choose r in order to minimize the range of possible values for $\alpha_1 s+\alpha_3 r$. To do this, we will determine a value T $(0<T<p)$ such that we can ensure both $\alpha_3 r<T$ and $\alpha_1|s|<T$ by choosing r carefully.

We first consider the inequality $\alpha_1|s|<T$. Replacing s by $rMN-kp$ (from the definition of s) and bounding α_1 by $\frac{p}{MN}$ gives us

$$\frac{p}{MN}\,|rMN-kp|<T$$

and a few simple manipulations turn the inequality into

$$\left|\frac{MN}{p}-\frac{k}{r}\right|<\frac{(\frac{MNT}{p^2})}{r}\ .$$

Thinking of $\frac{MN}{p}$ as the real number γ, and $\frac{MNT}{p^2}$ as ϵ, we recognize Dirichlet's Theorem on rational approximation (see [18] page 60, for example). We can then say that there exists two integers k and r satisfying the inequality and such that $1\le r\le\frac{1}{\epsilon}=\frac{p^2}{MNT}$. We also know that k and r can be found using the continued fraction method (see D page 237, for a brief description).

Since the upper bound on r from Dirichlet's Theorem is tight, and since $0\le\alpha_3<M$, the best bounds we can give on $\alpha_3 r$ in general are $0\le\alpha_3 r<\frac{p^2}{NT}$. To ensure that $\alpha_3 r<T$, we must therefore require

$$\frac{p^2}{NT}\le T\ ,$$

or equivalently $T\ge p/\sqrt{N}$. Since we want T as small as possible (we will end up with an interval of size $2T$ for the kangaroo method), we fix $T=p/\sqrt{N}$.

This means that r can be selected such that computing α' with the kangaroo method can be done in a time bounded above by $O(\sqrt{2}p^{1/2}/N^{1/4})$. From $\alpha' = r\alpha_3 + s\alpha_1$, we have to solve an easy diophantine equation to obtain α_1 and α_3 (see Appendix C for details), and Equation (1) gives us α.

We can therefore reduce the search time for the discrete log from $O(\sqrt{p})$ for Pollard rho (or the kangaroo method applied directly on the possible values of α) by a factor of at least $O(N^{1/4})$ in general, and up to $O(\sqrt{N})$ in the best situations, while keeping the memory requirement of $O(\log p)$ of the Kangaroo algorithm.

4 Scenario II – Partial Information About the Square and Multiply Chain Is Revealed

In order to do modular exponentiation efficiently, typically one uses the *square and multiply algorithm*. We will illustrate the working of this algorithm by means of an example here and refer the reader to [10], page 71, for its formal description and analysis.

For example, suppose we want to compute g^{43}. We will first write down 43 in binary as 101011. Starting after the leading 1, replace each 0 by S (Square) and each 1 by SM (Square and Multiply) to get the string $SSMSSMSM$. Such a string goes by the name *square and multiply chain*. We start with $h = g$ (i.e. with g^1, corresponding to the leading bit) and do the operations (Squaring h and Multiplying h by g) specified by a scan of the above string from left to right, storing the result back in h each time. At the end, h will have the desired result. In this particular example, the successive values assumed by h would be $g \to g^2 \to g^4 \to g^5 \to g^{10} \to g^{20} \to g^{21} \to g^{42} \to g^{43}$. Note that, the final value of h is g^{43} as desired.

In this section, we assume that exponentiation is done using the square and multiply algorithm. As the power consumption profile of squaring operation is often considerably different from that of multiplication operation, one could figure out which one of the two operations is being performed using side channel information (unless sufficient countermeasures are used). In the following, we assume that some partial information about the square and multiply chain is revealed by side channel attacks.

Specifically, we assume that a side channel attack revealed the position of some of the multiplications (M) and squares (S) of the square and multiply chain. Note that once a multiplication is identified, the operations next to it are known to be squares (i.e. we have the substring SMS) since there are no consecutive multiplications in the square and multiply chain. We will assume that n elements of the square and multiply chain have not been identified, of which i are multiplications (M). The problem that we address is how to exploit the partial information effectively to figure out the entire square and multiply chain.

As the chain is made up of only S's and M's, if we can figure out the positions of all the M's, the string is completely determined, so we need to figure out the

exact positions of the remaining i M's. A naive approach to solving the problem consists in guessing the positions of the i remaining M's. This will determine the chain completely and hence the key. We can then verify whether our guess is correct by checking if g^α equals β. If we guessed correctly we can stop, otherwise we can make new guesses until we eventually succeed. In this approach, we are essentially doing an exhaustive search, so the complexity would be $O(n^i \log p)$ as there are $\binom{n}{i}$ possible guesses for the missing M's, each of which requires $O(\log p)$ time to test.

Also note that even though we may know the relative position of some of the multiplications in the square and multiply chain, this does not readily translate into information on the bitstring as the number of the remaining M's after and between two known M's will change the position of the corresponding nonzero bits in the binary representation of α (in particular, this is why the algorithms of Stinson [22] and Cheng [3] cannot easily be adapted to work in this situation).

We could use the fact that no two M's are next to one another in the square and multiply chain to reduce the number of possible guesses to test. However, the overall effect will usually be small, and the worst case complexity will continue to be essentially $O(n^i \log p)$. To solve this, we develop a more sophisticated approach of exploiting the partial information that will make an impact on the worst case complexity.

First, we assume that we can somehow split the chain into two parts, left and right, such that $\frac{i}{2}$ of the remaining M's are on the left part and the other $\frac{i}{2}$ are on the right part. We can now make use of the time-memory trade off technique and determine the entire chain in time $O(n^{\frac{i}{2}} \log p)$. The details are explained below.

Suppose we make a specific guess for the $\frac{i}{2}$ M's on the left part and another guess for the $\frac{i}{2}$ M's on the right part. This determines the square and multiply chain completely and hence the bit string representation of the key α. Let a be the number represented by the bit string corresponding to the left part and let b be the number represented by the bit string corresponding to the right part. Let x be the length of the bitstring corresponding to the right part. Note that, we do know x as we are assuming, for the moment, that the position of the split is given to us. Then, clearly

$$\begin{aligned} \beta &= g^\alpha \\ &= g^{a2^x+b} \\ &= \left(g^{2^x}\right)^a g^b \ . \end{aligned} \tag{5}$$

If we denote g^{-2^x} by h, then the above equation reduces to

$$g^b = h^a \beta \ . \tag{6}$$

We can use Equation (6) to check whether our guess is correct. However, even if we use Equation (6) to verify a guess, the worst case complexity will still be $O(n^i \log p)$. This is because there will be $O(n^{\frac{i}{2}})$ guesses for a, $O(n^{\frac{i}{2}})$ guesses for b and any guess for a can be paired with any guess for b to make up a complete guess.

Instead, we shall use a time-memory trade off technique. Consider all different possible guesses for the left part. This will yield all different possible guesses for a. For each such guess we compute h^a and we record the pairs (a, h^a). We then sort all the pairs into an ordered table based on the second column, viz. the value of h^a.

Next we make a guess for the right part. This will yield a guess for b. We can now compute $y = \beta^{-1} \times g^b$. If our guess for b is indeed correct, then y will be present in the second column of some row in the table we built. The first column of that row will produce the matching guess for a. So, all that we have to do is search for the presence of y in the second column of the table. This can be done using the *binary search* algorithm as the table is already sorted as per the second column. If y is present, we are done and have determined the key. If y is not present, we make a different guess for the right part and continue until we eventually succeed. Since there are $n^{\frac{i}{2}}$ guesses for the right part, the time complexity of this algorithm will be $O(n^{\frac{i}{2}} \log p)$. As there are $n^{\frac{i}{2}}$ guesses for the left part, the table that we are building will have that many entries and so the space complexity of our algorithm will be $O(n^{\frac{i}{2}} \log p)$.

Hence, both the time and space complexity of our algorithm will be $O(n^{\frac{i}{2}})$ (ignoring the $\log p$ term). In contrast, the naive approach would have a time complexity of $O(n^i)$ and space complexity of $O(\log p)$ as only constant amount of storage space is needed. This is why this is called a time-memory trade off technique.

In the analysis above, we ignored the costs associated with handling the table (sorting and searching). Let $m = n^{\frac{i}{2}}$. Whereas it takes time $O(m \log p)$ (oracle operations) to compute the elements of the table, sorting it will require a time of $O(m \log m)$ bit operations even with an optimal sorting algorithm (such as merge sort). Similarly, to search in a sorted table of size $O(m)$ even with an optimal searching algorithm (such as binary search) will take $O(\log m)$ bit operations. Note that in practice it is common to use a hash table when m is large, but this does not change the form of the asymptotic cost. As we have $O(m)$ guesses for the right part and a search is needed for each guess, the total time spent after the table is built would be $O(m \log m)$. So, technically speaking, the true complexity of our algorithm is $O(m \log m) = O(n^{i/2} \log p \log \log p)$ bit operations (since both n and i are $O(\log p)$). However, in practice our "unit time" of oracle (group) operation is more expensive than a bit operation (requiring at least $\log p$ bit operations), whereas the $\log \log p$ terms grows extremely slowly, and we can safely assume that the main cost (in oracle time) also covers the table costs.

Recall that we assumed that we can somehow split the chain into two parts, left and right, such that $\frac{i}{2}$ of the remaining M's are in each of the two parts. This is, of course, an unjustified assumption. Although we made this assumption for ease of exposition, this assumption is not really needed. If we consider the set of remaining M's as ordered, then we can easily define one of them as the "middle one". We will use the position of the "middle" M as our splitting position. There are $n - i = O(n)$ possible positions for the "middle" M (of which only one is the true position).

For each of the possible positions, we try to obtain a match by assuming the "middle" M is in this position and placing the $i-1$ others. If i is odd, then we have $\frac{i-1}{2} = \lfloor \frac{i}{2} \rfloor$ of the remaining M's on each side, and if i is even then we have $\frac{i}{2} - 1$ of the remaining M's on one side (say, on the left) and $\frac{i}{2}$ on the other (with $\frac{i}{2} = \lfloor \frac{i}{2} \rfloor$). Using the time-memory trade off technique, we have a complexity of $O(n^{\lfloor \frac{i}{2} \rfloor} \log p)$ for each of the possible positions for the "middle" M, and we obtain a total complexity of $O(n^{\lfloor \frac{i}{2} \rfloor + 1} \log p)$.

With our algorithm, we are able to cut down the complexity from $O(n^i)$ for the naive approach to $O(n^{\lfloor \frac{i}{2} \rfloor + 1})$ (ignoring logarithmic terms). This is a significant improvement considering that the exponent of n has been reduced to about half the original value and n will typically be very large (but still $O(\log p)$) in practical implementations of elliptic curve based cryptosystems.

5 Concluding Remarks

To summarize, in this article we considered the problem of determining the key used in discrete logarithm based systems when partial knowledge of the key is obtained by side channel attacks. We considered two different scenarios of partial information viz. knowing a sequence of contiguous bits in the key and knowing some part of the square and multiply chain. In both scenarios, we were able to develop better algorithms in comparison to both using a square-root algorithm (ignoring the partial information available to us) and doing an exhaustive search using the extra information available. In particular, in the second scenario, our algorithm is almost asymptotically optimal considering that its complexity is very close to the square root of the order of the remaining key space.

Although we have made some progress, many more situations could be considered. We give the following as examples:

1. Consider the first scenario where we assume that a sequence of contiguous bits in the middle of the key corresponding to a set of size N have been revealed by side channel attacks (Section 3.1). Although we were able to reduce the search time, only some situations will match the optimal search time of $O(\sqrt{p/N})$. For a general combination of p, M and N, we would still have to reduce the search time by a factor of $O(N^{1/4})$ to obtain an asymptotically optimal algorithm.
2. In the first scenario, we assumed that the known bits are contiguous bits. It is possible that in some circumstances, we may get to know some bits of the key, but the known bits may not be contiguous.
3. Finally, the Non-adjacent Form Representation (NAF) of the key is sometimes used to do the exponentiation operation more efficiently (in the average case) [5]. If a sequence of bits is known, the situation is very similar to that of Section 3. When partial information about the square and multiply chain is obtained, the situation changes significantly compared to the binary square and multiply, since it is usually assumed that multiplications by g and g^{-1} are indistinguishable. Although it is easy to adapt the algorithm presented in

Section 4 to locate the position of the multiplications coming from nonzero bits, a factor of $O(2^m)$ (where m is the number of nonzero bits in the NAF representation of the key) would be included in the complexity to deal with the signs of the bits, which often cancels any gains we obtained.

In these situations, we leave the development of optimal algorithms, whose complexity would be the square root of the order of the remaining key space (or close to it), as an open problem.

Acknowledgments. This paper is an outcome of a research project proposed at the RMMC Summer School in Computational Number Theory and Cryptography which was held at the University of Wyoming in 2006. We would like to thank the sponsors for their support.

References

1. Brier, É., Déchène, I., Joye, M.: Unified point addition formulæ for elliptic curve cryptosystems. In: Embedded Cryptographic Hardware: Methodologies and Architectures, pp. 247–256. Nova Science Publishers (2004)
2. Brier, É., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
3. Cheng, Q.: On the bounded sum-of-digits discrete logarithm problem in finite fields. SIAM J. Comput. 34(6), 1432–1442 (2005)
4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
5. Gordon, D.M.: A survey of fast exponentiation methods. Journal of Algorithms 27, 129–146 (1998)
6. Howgrave-Graham, N., Smart, N.P.: Lattice attacks on digital signature schemes. Des. Codes Cryptogr. 23(3), 283–290 (2001)
7. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
8. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
9. Leadbitter, P.J., Page, D., Smart, N.P.: Attacking DSA under a repeated bits assumption. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 428–440. Springer, Heidelberg (2004)
10. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton (1996)
11. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the digital signature algorithm with partially known nonces. J. Cryptology 15(3), 151–176 (2002)
12. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Des. Codes Cryptogr. 30(2), 201–217 (2003)
13. Odlyzko, A.M.: Discrete logarithms: The past and the future. Designs, Codes and Cryptography 19, 129–145 (2000)

14. Pohlig, S.C., Hellman, M.E.: An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. IEEE Transactions on Information Theory 24, 106–110 (1978)
15. Pollard, J.M.: Monte Carlo methods for index computation (mod p). Mathematics of Computation 32(143), 918–924 (1978)
16. Pollard, J.M.: Kangaroos, Monopoly and discrete logarithms. Journal of Cryptology 13(4), 437–447 (2000)
17. Quisquater, J.-J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
18. Schrijver, A.: Theory of Linear and Integer Programming. In: Wiley-Interscience Series in Discrete Mathematics, John Wiley & Sons, Chichester (1986)
19. Shanks, D.: Class number, a theory of factorization and genera. In: Proc. Symp. Pure Math., vol. 20, pp. 415–440 (1971)
20. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
21. Stebila, D., Thériault, N.: Unified point addition formulae and side-channel attacks. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 354–368. Springer, Heidelberg (2006)
22. Stinson, D.: Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. Math. Comp. 71(237), 379–391 (2002)
23. Teske, E.: Square-root algorithms for the discrete logarithm problem (a survey). In: Public-Key Cryptography and Computational Number Theory, pp. 283–301. Walter de Gruyter, Berlin (2001)
24. Walter, C.D.: Simple power analysis of unified code for ECC double and add. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 191–204. Springer, Heidelberg (2004)

A Case I – Left Part

Here we assume that contiguous most significant bits of the key are known. Let z denote the bit string formed by the sequence of known bits. Suppose that l is the length of the key. Suppose that l_1 is the length of the known sequence of contiguous bits and l_2 is the length of the remaining bits so that $l = l_1 + l_2$.

Then the smallest possible value for α is the number a represented (in unsigned binary notation) by z concatenated with a sequence of l_2 zeroes. The largest possible value for α is the number b represented by z concatenated with a sequence of l_2 ones. We do know that $a \leq \alpha \leq b$.

Since α could take all the values in the interval $[a, b]$, we are in the ideal situation for Pollard's *Kangaroo Algorithm*. This probabilistic algorithm [15,16] was developed to compute the discrete logarithm when it is known to lie in an interval $[a, b]$. It also can be implemented in a space efficient manner and has expected running time of $O(\sqrt{b-a})$ under some heuristic assumptions.

Since we know both a and b, we can make use of Pollard's Kangaroo algorithm in this case. Note that the binary representation of $b - a$ in our case is simply a sequence of l_2 1's and so $b - a$ is $2^{l_2} - 1$. The expected running time to determine α using the Kangaroo algorithm will then be $O(2^{l_2/2})$.

Observe that if we ignored the partial information available to us and solved the DLP by using the rho method or the baby-step giant-step method, the running time would be $O(2^{l/2})$ which is much higher. Also, observe that if we had exhaustively searched for a key consistent with our partial knowledge, the running time would be $O(2^{l_2})$. So, we are able to do better than these two obvious ways.

B Case II – Right Part

Here we assume that contiguous least significant bits of the key are known. Let z denote the bit string formed by the sequence of known bits and let α_2 denote the number represented (in unsigned binary notation) by z. Suppose that l is the length of the key. Suppose that l_2 is the length of the known sequence of contiguous bits and l_1 is the length of the remaining bits so that $l = l_1 + l_2$.

Observe that if we ignored the partial information available to us and solved the DLP by using the rho method or the baby-step giant-step method, the running time would be $O(2^{l/2})$. Also, observe that if we had exhaustively searched for a key consistent with our partial knowledge, the running time would be $O(2^{l_1})$.

We know that $\alpha \equiv \alpha_2 \bmod 2^{l_2}$ (since we know the l_2 right-most bits of α), and we let α_1 be the integer corresponding to the l_1 left-most bits of α, i.e.

$$\alpha = \alpha_1 \times 2^{l_2} + \alpha_2 \ . \tag{7}$$

Let $M = \left\lfloor \frac{p-\alpha_2-1}{2^{l_2}} \right\rfloor$, then we know that $0 \leq \alpha_1 \leq M$ since $0 \leq \alpha \leq p-1$. Raising g to both sides of Equation (7), we can write

$$\begin{aligned} \beta = g^{\alpha} &= g^{\alpha_1 \times 2^{l_2} + \alpha_2} \\ &= g^{\alpha_1 \times 2^{l_2}} g^{\alpha_2} \\ &= \left(g^{2^{l_2}}\right)^{\alpha_1} g^{\alpha_2} \ . \end{aligned} \tag{8}$$

If we denote $g^{2^{l_2}}$ by g' and $\beta \times g^{-\alpha_2}$ by β', then Equation (8) reduces to $\beta' = (g')^{\alpha_1}$. As Teske [23] observed, we can then solve this DLP by using Pollard's Kangaroo Algorithm on g' and β' in $O(\sqrt{M})$ time as $0 \leq \alpha_1 \leq M$. Once α_1 is known, Equation (7) gives the value of α. As $M = \left\lfloor \frac{p-\alpha_2-1}{2^{l_2}} \right\rfloor$, the complexity is easily seen to be $O(\sqrt{\frac{p}{2^{l_2}}})$, which is same as $O(2^{l_1/2})$. Thus, we are able to cut down the complexity to square root of the size of the remaining key space.

C Solving the Diophantine Equation

In Section 3.1, we use Pollard's kangaroo algorithm to compute $\alpha' = \alpha_3 r + \alpha_1 s$, for some carefully chosen r and s. However, this raises the question of how to extract α_1 and α_3 from α', after which we can use Equation (1) to obtain α.

To do this, we first show that r and s may be assumed to be coprime. Since $s = rMN - kp$ with

$$\left|\frac{MN}{p} - \frac{k}{r}\right| < \frac{(\frac{MNT}{p^2})}{r} \ ,$$

we can assume that k and r are coprime: if $\gcd(k, r) = d > 1$, we can replace r and k with $\tilde{r} = r/d$ and $\tilde{k} = k/d$, which gives us $\tilde{s} = \tilde{r}MN - \tilde{k}p = s/d$. The inequality clearly still holds for $\tilde{r}$ and $\tilde{k}$ since $\frac{\tilde{k}}{\tilde{r}} = \frac{k}{r}$ and $\frac{(\frac{MNT}{p^2})}{r} < \frac{(\frac{MNT}{p^2})}{\tilde{r}}$, but we have smaller values for the interval, which is clearly more advantageous for the search. Furthermore, r is also coprime to p (since p is a prime and $0 < r < p$), so we have

$$\gcd(r, s) = \gcd(r, rMN - kp) = \gcd(r, kp) = 1 \ .$$

Once we have that $\gcd(r, s) = 1$, finding all integer solutions of $\alpha' = s\alpha_1 + r\alpha_3$ is straightforward. Well known number theoretic techniques give us that all solutions are of the form

$$\alpha_1 = b + ir$$
$$\alpha_3 = \frac{\alpha' - sb}{r} - is$$

where $b \equiv \alpha' s^{-1} \bmod r$. The problem is then to restrict the number of possible solutions, and the choice of r and s helps us once again. Recall that we started with the condition $|s| < p/2$, which forces $r > \frac{p}{2MN}$. Since $0 \leq \alpha_1 < \frac{p}{MN}$, there are at most two possible (and easily determined) values of i, and we can verify each one in time $O(\log p)$.

D Computing r and k

We now give a brief description of how to compute r and k using the continued fraction method. For the theoretical background, the reader can refer to [18]. In our context, we are trying to approximate the number $\gamma = \frac{MN}{p}$ with a rational $\frac{k}{r}$ such that the approximation error is less than $\frac{\epsilon}{r}$ with $\frac{1}{\epsilon} = \frac{p^2}{MNT} = \frac{p}{M\sqrt{N}}$.

The continued fraction method works iteratively, giving approximations k_i/r_i of γ ($i \geq 0$). Since $0 < \gamma < 1$, we can initialize the process with $\gamma_0 = \gamma$, $r_0 = 1$, $k_0 = 0$, $r_{-1} = 0$ and $k_{-1} = 1$. At each iterative step, we let $a_i = \lfloor \gamma_{i-1}^{-1} \rfloor$ and we compute $r_i = a_i r_{i-1} + r_{i-2}$, $k_i = a_i k_{i-1} + k_{i-2}$, and $\gamma_i = \gamma_{i-1}^{-1} - a_i$. The continued fraction expansion of γ will be $[0; a_1, a_2, a_3, \ldots]$.

To find an optimal pair (r, s), i.e. a pair that minimizes $Mr + \frac{p}{MN}|s|$ (the size of the interval that will be searched with the kangaroo method), we proceed as follows. Once we have a first approximation $\frac{k_i}{r_i}$ of γ such that $r_{i+1} < \frac{1}{\epsilon}$, we evaluate $L_i = Mr_i + \frac{p}{MN}|s_i|$ (with $s_i = r_i MN - k_i p$). We then continue the iterations, keeping track of the best pair r, s found so far (in the form of a triple (r_j, s_j, L_j)). Once $Mr_{i+1} > L_j$, we are done and we can set $r = r_j$, $s = s_j$. To see that we find the optimal pair, observe that the value of r_i never decreases, so once $Mr_{i+1} > L_j$ all further iterations will produce an L_i greater than L_j. This will take no more than $O(\log p)$ steps.