LMS J. Comput. Math. 17 (Special issue A) (2014) 148–156
doi:10.1112/S1461157014000230

The discrete logarithm problem for exponents of bounded height

Simon R. Blackburn and Sam Scott

Abstract

Let G be a cyclic group written multiplicatively (and represented in some concrete way). Let n be a positive integer (much smaller than the order of G). Let $g, h \in G$. The *bounded height discrete logarithm problem* is the task of finding positive integers a and b (if they exist) such that $a \leqslant n$, $b \leqslant n$ and $g^a = h^b$. (Provided that b is coprime to the order of g, we have $h = g^{a/b}$ where a/b is a rational number of height at most n. This motivates the terminology.)

The paper provides a reduction to the two-dimensional discrete logarithm problem, so the bounded height discrete logarithm problem can be solved using a low-memory heuristic algorithm for the two-dimensional discrete logarithm problem due to Gaudry and Schost. The paper also provides a low-memory heuristic algorithm to solve the bounded height discrete logarithm problem in a generic group directly, without using a reduction to the two-dimensional discrete logarithm problem. This new algorithm is inspired by (but differs from) the Gaudry–Schost algorithm. Both algorithms use $O(n)$ group operations, but the new algorithm is faster and simpler than the Gaudry–Schost algorithm when used to solve the bounded height discrete logarithm problem. Like the Gaudry–Schost algorithm, the new algorithm can easily be carried out in a distributed fashion.

The bounded height discrete logarithm problem is relevant to a class of attacks on the privacy of a key establishment protocol recently published by EMVCo for comment. This protocol is intended to protect the communications between a chip-based payment card and a terminal using elliptic curve cryptography. The paper comments on the implications of these attacks for the design of any final version of the EMV protocol.

1. *Introduction*

Let G be a cyclic group (written multiplicatively). We suppose that G is represented in such a way that every element has a unique normal form, and the operations of multiplication, inversion and the computation of the normal form of an element can all be carried out efficiently. (For example, G might be a subgroup of the multiplicative group of the integers modulo p, or a suitable subgroup of points on an elliptic curve.) For simplicity, we assume that G has prime order (which will be true in the applications we have in mind).

We define the *bounded height discrete logarithm problem on G* to be the task of, given group elements $g, h \in G$ (in normal form) and an integer n, computing positive integers a and b such that $a \leqslant n$, $b \leqslant n$ and

$$g^a = h^b$$

if such integers exist. We will assume that $n \leqslant (1/5)\sqrt{|G|}$, so n is significantly smaller than the order of G. Note that if a and b have a common factor d, then we may divide a and b by d to produce another solution to the bounded height discrete logarithm problem (as d is coprime to the order of G). So we will assume that a and b are coprime.

Recall that the height $h(a/b)$ of a rational number a/b (where a and b are coprime integers) is defined by $h(a/b) = \max\{|a|, |b|\}$. Thus (as b and the order of g are coprime) the bounded

Received 27 February 2014; revised 23 May 2014.

2010 Mathematics Subject Classification 11T71 (primary), 94A60 (secondary).

Contributed to the Algorithmic Number Theory Symposium XI, GyeongJu, Korea, 6–11 August 2014.

The second author gratefully acknowledges financial support from EPSRC grant no. EP/K035584/1.

height discrete logarithm problem asks for a positive rational number q with $h(q) \leqslant n$ such that $h = g^q$.

The bounded height discrete logarithm problem is a natural generalization of the problem where the discrete logarithm is an *integer* known to lie in an interval of length n. (There is an easy reduction of the latter problem to the case when the interval the logarithm lies in is $[1, n]$. So we are asked to solve the bounded height discrete logarithm problem where we are given the extra information that $b = 1$.) This special case, when the discrete logarithm lies in a short interval, can be solved using $O(\sqrt{n})$ group operations and $O(1)$ memory using Pollard's kangaroo method [7, 8].

A naive algorithm to solve the bounded height discrete logarithm problem exhaustively searches the n^2 possibilities for the pair (a, b) until a solution is found. But there is clearly a much faster algorithm based on 'baby-step giant-step' techniques. In this algorithm, we compute all powers g^a where $1 \leqslant a \leqslant n$ and store these values in an easily searched list. For each b such that $1 \leqslant b \leqslant n$, we compute h^b and check whether it is equal to some g^a in our list. This algorithm requires $O(n)$ group operations; this is of the order of the square root of the solution space, just as for Pollard's kangaroo method. But, in contrast to the kangaroo method, the algorithm has a very large memory requirement (which is a significant problem in practice). We aim to show that the bounded height discrete logarithm problem may be solved in $O(n)$ group operations using a heuristic Las Vegas algorithm with low memory requirements. The algorithm we describe can be easily implemented in a distributed environment.

Our algorithm to solve the bounded height discrete logarithm problem is inspired by the heuristic Las Vegas algorithm due to Gaudry and Schost [5] that solves the two-dimensional discrete logarithm problem; see Galbraith [4, Chapter 14] for an exposition of this algorithm. In fact, we will provide a reduction from the bounded height discrete logarithm problem to the two-dimensional discrete logarithm problem, and so the Gaudry–Schost algorithm can itself be used to solve the bounded height discrete logarithm problem (though less efficiently than by using the algorithm we provide in this paper).

We were motivated to study the bounded height discrete logarithm problem after examining the recent draft specification for a key establishment protocol which has been published for comment by EMVCo [2]. This protocol is based on elliptic curve Diffie–Hellman key agreement, and is intended to be used to establish secure communications between an EMV card (a chip-based payment card, for example) and a banking terminal. Communication is by direct contact between devices, or over the air. The final version of the protocol will be very widely used when implemented: there are currently over 1.6 billion EMV cards and over 23 million terminals in use [3]; these figures will rise significantly when chip-and-PIN cards are fully rolled out in the USA. One aim of the protocol is to provide privacy by protecting against card tracking: an eavesdropper should not be able to determine whether two intercepted communication sessions involve the same card. Brzuska, Smart, Warinschi and Watson [1] have shown that card tracking is possible for the proposed protocol parameters if an adversary can impersonate a terminal and interact with the card at some stage. We observe that there is a card-tracking attack that can be mounted by a passive adversary (without needing to impersonate a terminal at any stage), with the penalty of a higher, though still realistic, computational cost. In fact the protocol design is such that if the bounded height discrete logarithm problem can be solved, the privacy of this protocol can be compromised by a passive adversary. So the algorithm in this paper has implications for parameter choices for this protocol. Indeed, the parameters proposed in the draft protocol specification are not sufficiently large to prevent card tracking by a passive adversary. We provide more details below.

The structure of the remainder of this paper is as follows. In § 2 we provide a reduction from the bounded height discrete logarithm problem to the two-dimensional discrete logarithm problem, thus enabling the Gaudry–Schost algorithm to be used to solve the former problem. In § 3 we provide our heuristic algorithm to solve the bounded height discrete logarithm problem

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

directly. Finally, in § 4, we provide some details of the EMV protocol and discuss the impact of our results on the parameter choices that are to be made.

2. *A reduction to the two-dimensional discrete logarithm problem*

The *two-dimensional discrete logarithm problem* takes as input a positive integer ν and elements g_1, g_2, g_3 of a (not necessarily cyclic) group G. It outputs integers x_1 and x_2 such that $0 \leqslant x_1 < \nu$, $0 \leqslant x_2 < \nu$ and $g_3 = g_1^{x_1} g_2^{x_2}$ (if these integers exist). The low-memory heuristic algorithm due to Gaudry and Schost [5] solves this problem using $O(\nu)$ group operations.

This section describes a reduction of the bounded height discrete logarithm problem to the two-dimensional discrete logarithm problem. We use the notation for the bounded height discrete logarithm problem established in the introduction above. For the purposes of our reduction, we assume that we have an oracle for the two-dimensional discrete logarithm problem. We will show that there exists a positive real number p so that a single call to the oracle will allow us to solve an instance of the bounded height discrete logarithm problem with probability at least p. (So to increase the probability of success to any given desired threshold, a bounded number of oracle calls are needed.)

2.1. *A first (unsuccessful) attempt*

A first attempt at a reduction sets $\nu = n$, $g_1 = g$, $g_2 = h^{-1}$ and $g_3 = 1$. If (a, b) is a solution to the bounded height discrete logarithm problem, then

$$g_1^a g_2^b = g^a h^{-b} = 1 = g_3,$$

and so we might hope that an oracle for the two-dimensional discrete logarithm problem will return the solution $x_1 = a$, $x_2 = b$ that we require. Unfortunately, we cannot guarantee that the oracle does not return the trivial solution $x_1 = x_2 = 0$.

2.2. *A second (unsuccessful) attempt*

For a second attempt at a reduction, we choose $\nu = n$, $g_1 = g$, $g_2 = h^{-1}$ as before. But then we choose two random integers r_1 and r_2 with $0 \leqslant r_1 < \nu$ and $0 \leqslant r_2 < \nu$ and set $g_3 = g_1^{r_1} g_2^{r_2}$. An oracle for the two-dimensional discrete logarithm problem returns some solution x_1 and x_2. Note that $|x_i - r_i| \leqslant \nu$ and

$$g_1^{x_1 - r_1} g_2^{x_2 - r_2} = g_1^{x_1} g_2^{x_2} (g_1^{r_1} g_2^{r_2})^{-1} = g_3 g_3^{-1} = 1.$$

Recall that we are assuming that $n \leqslant (1/5)\sqrt{|G|}$. Lemma 2.1 below (with $\beta = 1$ and, say, $\alpha = 1/4$) shows that g_1 and g_2 have no small non-trivial relations, so $(x_1 - r_1, x_2 - r_2) = (da, db)$ for some integer d. As long as $d \neq 0$ (in other words, as long as the two-dimensional discrete logarithm problem returns a non-trivial solution), we may calculate $d = \gcd(x_1 - r_1, x_2 - r_2)$ and then return the required solution $a = (x_1 - r_1)/d$, $b = (x_2 - r_2)/d$ to the bounded height discrete logarithm problem. However, it might be the case that no non-trivial solutions exist. Indeed, if a or b is very large then this will be the case with high probability. (To see this note that if (for example) a is large then it is likely that $a > r_1 \geqslant \nu - a$. But then if $d > 0$ we find $x_1 = da + r_1 \geqslant a + \nu - a = \nu$, and if $d < 0$ then $x_1 = da + r_1 < -a + a = 0$. Since $0 \leqslant x_1 < \nu$ we have a contradiction in both cases, and so we must have $d = 0$.)

LEMMA 2.1. *Let $\alpha, \beta \in \mathbb{R}$ be positive real numbers such that $2\alpha^2\beta < 1$. Let G be a cyclic group of prime order and let $g, h \in G \setminus \{1\}$. Let n be a positive integer where $n \leqslant \alpha\sqrt{|G|}$. Suppose g and h satisfy a small relation: $g^a h^{-b} = 1$ for coprime integers a and b such that*

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

$1 \leqslant a \leqslant n$ and $1 \leqslant b \leqslant n$. Then there are no other non-trivial relations between g and h. More precisely, suppose y_1 and y_2 are integers such that $g^{y_1}h^{y_2} = 1$ and such that $|y_1| \leqslant \beta n$ and $|y_2| \leqslant \beta n$. Then there exists an integer d such that $y_1 = da$ and $y_2 = -db$.

Proof. Note that

$$g^{y_2 a + y_1 b} = (g^a h^{-b})^{y_2} (g^{y_1} h^{y_2})^b = 1.$$

Since g is a non-identity element in a group of prime order, we see that $|G|$ divides $y_2 a + y_1 b$. But

$$|y_2 a + y_1 b| \leqslant 2\beta n^2 \leqslant 2\alpha^2 \beta |G| < |G|,$$

and so $y_2 a + y_1 b = 0$. Thus the matrix A defined by

$$A = \begin{pmatrix} a & -b \\ y_1 & y_2 \end{pmatrix}$$

has determinant zero. The first row of A is non-zero, so the second row of A must be a (rational) multiple of the first. Indeed, since a and b are coprime, the second row must be an integer multiple d of the first. So the conclusion of the lemma follows. □

2.3. *A final (successful) attempt*

Our final attempt at a reduction adjusts the parameters of the problem to make sure that our oracle outputs non-trivial solutions with high probability. We fix a real number β where $\beta > 1$ (we might use $\beta = 2$ in practice). We set $g_1 = g$ and $g_2 = h^{-1}$ as before, but we set $\nu = \beta n$ (so our oracle expects to receive inputs within a larger range). We then proceed as before: we choose two random integers r_1 and r_2 with $0 \leqslant r_1 < \nu$ and $0 \leqslant r_2 < \nu$, and $g_3 = g_1^{r_1} g_2^{r_2}$. An oracle for the two-dimensional discrete logarithm problem returns some solution x_1 and x_2. The argument above, using Lemma 2.1 with $\alpha < 1/\sqrt{2\beta}$, shows that $(x_1 - r_1, x_2 - r_2) = (da, db)$ for some integer d. As before, when $d \neq 0$ we may calculate $d = \gcd(x_1 - r_1, x_2 - r_2)$ and return the required solution $a = (x_1 - r_1)/d$, $b = (x_2 - r_2)/d$ to the bounded height discrete logarithm problem. To show we have our reduction, it therefore remains to show that d is non-zero with positive probability.

For a pair $(r_1, r_2) \in \mathbb{Z} \times \mathbb{Z}$ with $0 \leqslant r_i < \beta n$, define

$$\mu(r_1, r_2) = |\{(x_1, x_2) \in \mathbb{Z} \times \mathbb{Z} : 0 \leqslant x_i < \beta n \text{ and } g_1^{x_1} g_2^{x_2} = g_1^{r_1} g_2^{r_2}\}|.$$

Define p_i to be the probability that $\mu(r_1, r_2) = i$, when r_1 and r_2 are chosen randomly and independently. The probability that our oracle returns a non-trivial solution is

$$\sum_{i=2}^{\infty} p_i((i-1)/i) > \frac{1}{2} \sum_{i=2}^{\infty} p_i = \frac{1}{2}(1 - p_1).$$

When $r_1 > n$ and $r_2 < \beta n - n$ we have $\mu(r_1, r_2) \geqslant 2$, since both (r_1, r_2) and $(r_1 - a, r_2 + b)$ lie in the set of pairs counted by $\mu(r_1, r_2)$. So $p_1 \leqslant 1 - (\beta - 1)^2/\beta^2 < 1$. Thus the probability that the oracle returns a non-trivial solution is bounded away from 0 (by a function of β). Since β is a constant (we suggested $\beta = 2$ above) we have an efficient reduction of the bounded height discrete logarithm problem to the two-dimensional discrete logarithm problem.

3. *An algorithm for the bounded height discrete logarithm problem*

3.1. *The algorithm*

We now describe our algorithm for the bounded height discrete logarithm problem. We take a similar approach to describing the Gaudry–Schost algorithm as Galbraith [4]. The

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

Gaudry–Schost algorithm generates sets of 'tame' and 'wild' pseudorandom walks, and looks for a collision between a tame and a wild walk. In contrast, the algorithm below generates pseudorandom walks of a single type and looks for a collision between any pair of walks, making it more efficient for the problem we consider.

Let G, g, h and n be the input to the algorithm. We must choose four parameters: a real number $p \in [0,1]$, a real number $\beta > 1$, and positive integers k and m. The choice of these parameters will affect the complexity of the algorithm; typical choices might be $p \approx (10 \log n)/n$, $\beta = 2$, $k \approx 2 \log n$ and $m \approx n/(1000 \log n)$.

Let $D \subseteq G$ be a set of size approximately $p|G|$. We say that a group element in D is a *distinguished point*. We choose D so that it is easy to check whether a group element lies in D. (In practice we define D by specifying a hash function δ from the set of normal forms for G to some set X of size approximately p^{-1}, choosing an element $x \in X$ and setting $D = \delta^{-1}(x)$.)

Let $S = \{(i,j) \in \mathbb{Z}^2 : i,j \in [1, \beta n]\}$. Let $\gamma : S \to G$ be defined by $\gamma((i,j)) = g^i h^j$.

We choose pairs $(r_1, s_1), (r_2, s_2), \ldots, (r_k, s_k) \in \mathbb{Z} \times \mathbb{Z}$, where the integers r_i and s_i are picked uniformly and independently from the interval $[-m, m]$. We choose an (efficiently computable) selection function $\sigma : G \to \{1, 2, \ldots, k\}$, and we define next-state function ω (which we will use to define pseudorandom walks on S) by

$$\omega((i,j)) = ((i + r_\ell, j + s_\ell)),$$

where $\ell = \sigma(\gamma((i,j)))$. Note that the next state function has the crucial property that if $(i,j), (i',j') \in S$ are such that $(i,j) \neq (i',j')$ but $\gamma((i,j)) = \gamma((i',j'))$ then the same is true after ω is applied to (i,j) and (i',j'): we have $\omega((i,j)) \neq \omega((i',j'))$ but $\gamma(\omega((i,j))) = \gamma(\omega((i',j'))$.

We generate a sequence of pseudorandom walks on S as follows. We choose integers $a', b' \in [1, \beta n]$ uniformly and independently at random, and start a walk at the point (a', b'). We iterate the walk using the function ω until we reach a point (i,j) where $g' := \gamma((i,j))$ is distinguished. We then terminate the walk, storing the value $(g', (i,j))$ in a list. (If we ever walk outside the set S, we terminate the walk without storing any value.) We continue generating random walks until we obtain a *non-trivial collision*, by which we mean we have elements $(g', (i,j))$ and $(g', (i',j'))$ in our list with $\gamma((i,j)) = g' = \gamma((i',j'))$ (a *collision*) and $(i,j) \neq (i',j')$ (the non-triviality condition). Without loss of generality $i \geqslant i'$. By the definition of γ,

$$g^i h^j = g' = g^{i'} h^{j'} \quad \text{and so } g^{i-i'} = h^{j'-j}.$$

We compute $d = \gcd(i - i', j' - j)$ and set $a = (i - i')/d$ and $b = (j' - j)/d$. We output the pair (a, b).

3.2. *A heuristic complexity analysis*

If our algorithm terminates, the argument in § 2 (see Lemma 2.1) shows that it will output a correct pair (a, b) (in particular, a and b lie in the correct range). So it remains to estimate the expected number of group operations that are carried out before the algorithm terminates.

We analyse the complexity of our algorithm under several simplifying assumptions and heuristics, which are essentially identical to the assumptions made in the analysis of the Gaudry–Schost algorithm. We assume that the parameters p, β, k and m are chosen as recommended above: $p \approx (10 \log n)/n$, $\beta = 2$, $k \approx 2 \log n$, and $m \approx n/(1000 \log n)$.

We assume that the probability that a pseudorandom walk strays from the set S is insignificantly small. (This assumption is valid if the algorithm's parameters are chosen appropriately, for example when the parameters are chosen as suggested above and n is sufficiently large.) We assume that, with high probability, no pseudorandom walk enters a cycle before termination. (Again this assumption is very reasonable for the suggested parameters;

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

fewer than 0.25% of walks ended in cycles in our experiments.) Less realistically, we assume that our pseudorandom walks may be modelled by walks which are uniformly distributed throughout S, and are randomly chosen until a collision takes place.

Note that, though our random walks consist of pairs $(i, j) \in S$, collisions are defined in terms of the images $\gamma(i, j)$ of these pairs in G, so we cannot use a direct application of the birthday paradox in S in our analysis of our algorithm's complexity. Let c be the total number of steps, summing over all our random walks, before and including any collisions (trivial or non-trivial). The expected number of distinct pairs $(i, j), (i', j') \in S$ with $\gamma((i, j)) = \gamma((i', j'))$ that we reach in our pseudorandom walk is $\binom{c}{2}|\gamma(S)|^{-1}$. Clearly $|\gamma(S)| \leqslant \beta^2 n^2$. In fact, $|\gamma(S)| \leqslant (2\beta - 1)n^2 = 3n^2$. To see this, note that if $i > n$ and $j < \beta n - n$ and $g^a = h^b$ then $g^i h^j = g^{i-a} h^{j+b}$. make Thus

$$\gamma(S) = \gamma(\{(i, j) \in S : i \leqslant n\}) \cup \{(i, j) \in S : j \geqslant \beta n - n\})$$

and so $|\gamma(S)| \leqslant \beta n^2 + (\beta - 1)n^2 = 3n^2$. Hence we expect to have a total of at least $\binom{c}{2}(3n^2)^{-1}$ (trivial or non-trivial) collisions.

For $(i, j) \in S$, define $\ell(i, j)$ to be the number of pairs $(i', j') \in S$ that collide with (i, j). In other words $\ell(i, j)$ is the number of $(i', j') \in S$ such that $\gamma((i', j')) = \gamma((i, j))$. The argument above shows that when $i > n$ and $j < \beta n - n$ we have that $\ell(i, j) \geqslant 2$. For the same reasons, when $j > n$ and $i \leqslant \beta n - n$ we have $\ell(i, j) \geqslant 2$. So when $\beta = 2$ we find that $\ell(i, j) \geqslant 2$ for at least half of the pairs $(i, j) \in S$. Thus if we have a collision, we would expect it to be non-trivial with probability at least $\frac{1}{4}$. (This is an underestimate, as we would expect elements $(i, j) \in S$ with $\ell(i, j) \geqslant 2$ to appear more often as in a collision than those with $\ell(i, j) = 1$.) Thus the expected number of non-trivial collisions is at least $\binom{c}{2}(12n^2)^{-1}$.

So we expect our algorithm to terminate when $c \approx \sqrt{24}\, n$. Note that we expect a constant number of collisions before the algorithm terminates (the analysis above suggests an expected number of 4 or less). We expect each collision to use approximately $2p^{-1}$ extra group operations in addition to those counted by c. Since $p \approx (10 \log n)/n$, this is an insignificant number of operations, which can safely be ignored.

To summarise, our heuristic assumptions predict that the expected number of group operations used by our algorithm is at most $(\sqrt{24} + o(1))n = O(n)$.

3.3. *Experimental results*

In order to implement our algorithm, we used the SAGE computer algebra system [9]. The key purpose of our experimentation was to compare the performance of our algorithm against the Gaudry–Schost algorithm. As a result, the emphasis was on comparing the relative runtime of the two algorithms, as opposed to optimizing the code in order to achieve the theoretical predictions. Neither our analysis nor the analysis performed by Gaudry and Schost gives any information about the impact of our choice of 'next-step' pairs (r_i, s_i) on the efficiency of the algorithm, though this seems to have a large effect in practice. Therefore, we did not make any significant effort to fine-tune the parameters for either algorithm, but used the same parameters and next-step pairs in each run of both algorithms so as to better compare their run-time.

To compare our algorithm to Gaudry–Schost, 1000 tests were performed when $n = 2^{16}$. On average, our algorithm required 111 664 steps, compared to 141 303 in the Gaudry–Schost algorithm. This represents an increase of speed of approximately 1.265 times. Additionally, the average number of stored distinguished points was 220 for our algorithm compared with 278 for the Gaudry–Schost algorithm. A test constituted generating an instance of the bounded height discrete logarithm problem, and running both algorithms on the instance. We now discuss our experiments in more detail.

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

The group used for running all experiments was a fixed cyclic group G. The order of G was a 256-bit prime.

The same pseudorandom map was used for both algorithms, and was defined as follows. The number of choices for the next state, k, was set to $k = 2 \log n$, for example $k = 32$ when $n = 2^{16}$. Increasing k did not appear to have any noticeable effect. The next state was chosen by hashing the representation of the point, and taking the $\log k$ most significant bits of the result. The 32-bit CRC provided by the Python library zlib was used as the hash function. The random walk offsets (r_i, s_i) were chosen randomly from the interval $[-m, m]$, where $m = n/10 \log n$. (We found the value $m \approx n/1000 \log n$ that has been previously suggested [4] for this parameter to be too small for the lower end of the parameter range we were considering. For example, when $n = 2^{16}$ we would have had $m = 4$ and this value would have caused some random walk pairs to be equal, or even zero, with significant probability.) The probability a point is distinguished is given by $p = (10 \log n)/n$ in our heuristic analysis. In our experiments, a distinguished point was determined by hashing the representation of the point and checking if the last $\lceil \log p \rceil$ least significant bits were zero.

To determine a good choice of β in our algorithm, experiments were run with $n = 2^{16}$. We performed 150 tests at each value β at increments of 0.01 for $1 < \beta < 1.2$ and increments of 0.1 for $1.2 \leqslant \beta < 5$. There appeared to be a minima at approximately 1.2, with the number of steps in an average run increasing linearly (and gradually) for greater values. However, when β was close to 1, the number of steps in a run varied to a much greater extent when compared to runs when a larger value of β was used. Therefore, we chose $\beta = 2$ as a value which has comparable performance, but more consistent results.

To test the performance of the algorithms as n varies, we performed experiments for values of n with $2^{12} \leqslant n \leqslant 2^{24}$. Both our algorithm and the Gaudry–Schost algorithm required approximately $n^{1.15}$ steps before terminating, rather than approximately n steps as predicted by the heuristic analysis; this might be expected, as we did not attempt to optimize our choice of offsets (r_i, s_i) in any way. The low-memory nature of our algorithm was confirmed by these experiments: the number of group elements which needs to be stored grows linearly in $\log n$, and on average less than 1000 elements were stored.

We have been able to perform a limited number of experiments for the larger value $n = 2^{32}$. The algorithm does terminate successfully, but we do not have enough data to perform a sensible analysis for this value of n.

4. *The EMV protocol*

This section describes the relevance of the bounded height discrete logarithm problem to a 'blinded Diffie–Hellmann' key establishment protocol, which has been published in 'Request for Comment' form by EMVCo [2].

The blinded Diffie–Hellman protocol is designed to secure communications between an EMV card (such as a bank card) and another device such as a point-of-sale terminal using elliptic curve cryptography. The goals of the protocol are to authenticate the card to the terminal, to detect modifications to any communications, and to protect against eavesdropping and card tracking.

The relevant part of the blinded Diffie–Hellman protocol is described as follows. Let G be a cyclic group of points on an elliptic curve. The suggested choice is a point on the curve P-256 as defined in the relevant NIST standard [6], so G is cyclic of prime order and $2^{255} < |G| < 2^{256}$.

Let x be a generator of G. The card possesses a 256-bit integer d_c (its private key) and the group element $y = x^{d_c}$ (its 'public' key, though this key is not revealed in any obviously public way), and certificates to prove the public key have been authorized. Let $k = 32$. (The EMV proposal chooses $k = 32$ for efficiency reasons, to reduce the computational cost of Step 1 below.)

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

(1) The card generates a random k-bit integer r, and sends $g = y^r$ to the terminal.
(2) The terminal generates a 256-bit integer d_t, and sends x^{d_t} to the card.
(3) Both the card and the terminal can compute the group element $x^{rd_cd_t}$, since

$$x^{rd_cd_t} = (x^{d_t})^{rd_c} = g^{d_t}.$$

The card and terminal derive a common key K from this group element.
(4) Using an authenticated encryption scheme and their common key K, the card sends its certified public key x^{d_c}, certificates and r to the terminal.
(5) The terminal checks whether $(x^{d_c})^r = g$, and authenticates the card's public key using the certificates.

A recent paper of Brzuska, Smart, Warinschi and Watson [1] provides a realistic security model for the protocol, and shows that so long as the protocol design is modified so that $k = 256$ the protocol is secure in this model.

However, there are problems with the goal of preventing card tracking when $k = 32$. To prevent card tracking, it must be hard for a third party to distinguish between runs of the protocol involving the same card, and runs involving different cards. Brzuska *et al.* point out that it is possible to obtain the public key y of a card by an active attack in which the adversary impersonates a terminal (note that the terminal is not authenticated). Once the public key y of a card is known, other runs of the protocol that are observed may (with high probability) be linked to the card by taking the value of g that is transmitted in Step 1 of the protocol and solving $g = y^r$ for r a k-bit integer. (Note that if the protocol involves another card, then with high probability the equation $g = y^r$ with r a k-bit integer will not have a solution.) The equation $g = y^r$ can be solved using $O(2^{k/2})$ group operations (and negligible memory) using Pollard's kangaroo method [7, 8], and so we have a feasible attack when $k = 32$.

We point out a feasible attack that allows card tracking even under a passive adversary model. Suppose we observe the group elements transmitted in Step 1 of two runs of the protocol. Let these elements be g and h, respectively. If the two runs involve the same card, we know that $g = y^b$ and $h = y^a$ where a and b are k-bit integers and where y is the public key of the card. But then $g^a = h^b$, and so the bounded height discrete logarithm problem has a solution. If the two runs are initiated by different cards, it is very unlikely that the bounded height discrete logarithm problem has a k-bit solution. So the algorithms in this paper provide a low-memory passive method for distinguishing whether two runs involve the same card, using $O(2^k)$ operations. Note that if the solution (a, b) to the bounded height discrete logarithm problem is unique (which is quite likely), we have actually recovered the card's public key, as $y = g^{1/b}$. The analysis in Lemma 2.1 shows that in fact all solutions to the bounded height discrete logarithm problem are multiples of each other (at least when k is smaller than about 128). So with very high probability we have reduced the number of possibilities for the public key to a small set (that could be exhaustively searched). One consequence of finding the public key of the card is that Pollard's kangaroo method can be used to tell whether any other runs of the protocol are associated with the card.

The analysis above shows that the value of $k = 32$ should not be used if card tracking needs to be prevented. Indeed, Brzuska *et al.* point out that small values of k cause problems with entity authentication too, as a corrupt terminal can manipulate a card into establishing two sessions with the same common key K. We recommend that k should be larger than its current value; the larger the better. Mandating that $k = 256$ would prevent the card tracking attacks discussed here. The security proofs of Brzuska *et al.* [1] provide guarantees that card tracking is hard, and that (one-sided) entity authentication, message authentication and privacy are achieved, under a well-defined and realistic security model which requires $k = 256$. So using $k = 256$ would add significant additional assurance to the protocol's security. Of course, choosing $k = 256$ significantly increases the computational cost of the protocol.

https://doi.org/10.1112/S1461157014000230 Published online by Cambridge University Press

Acknowledgements. The authors would like to thank Steven Galbraith for his encouragement at an early stage, and K. Paterson for his useful comments that have improved our exposition.

References

1. C. Brzuska, N. Smart, B. Warinschi and G. J. Watson, 'An analysis of the EMV channel establishment protocol', *Proceedings of 2013 ACM SIGSAC Conference on Computer and Communications Security* (ACM, New York, 2013) 373–386.
2. EMVCo, EMV ECC key establishment protocols, draft for comments, November 2012. Available from http://www.emvco.com/specifications.aspx?id=243.
3. EMVCo, Worldwide EMV card and terminal deployment, http://www.emvco.com/about_emvco.aspx?id=202, retrieved 5th February 2014.
4. S. D. Galbraith, *Mathematics of public key cryptography* (Cambridge University Press, Cambridge, 2012).
5. P. Gaudry and É. Schost, 'A low-memory parallel version of Matsuo, Chao and Tsujii's algorithm', *ANTS VI*, Lecture Notes in Computer Science 3076 (ed. D. A. Buell; Spinger, Berlin, 2004) 208–222.
6. National Institute of Standards and Technology (NIST), *Recommended elliptic curves for federal government use*, July 1999.
7. P. C. van Oorschot and M. J. Wiener, 'On Diffie–Hellman key agreement with short exponents', *Eurocrypt '96*, Lecture Notes in Computer Science 1070 (ed. U. Maurer; Springer, Berlin, 1996) 332–343.
8. J. M. Pollard, 'Monte Carlo methods for index computation (mod p)', *Math. Comp.* 32 (1978) 918–924.
9. W. A. Stein *et al.*, Sage mathematics software (version 5.11), The Sage Development Team, 2013, http://www.sagemath.org.

Simon R. Blackburn
Department of Mathematics
Royal Holloway University of London
Egham, Surrey
TW20 0EX
United Kingdom

s.blackburn@rhul.ac.uk

Sam Scott
Department of Mathematics
Royal Holloway University of London
Egham, Surrey
TW20 0EX
United Kingdom

sam.scott.2012@live.rhul.ac.uk