Kangaroos in Side-Channel Attacks

Tanja Lange[1], Christine van Vredendaal[1,2], and Marnix Wakker[2]

[1] Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tanja@hyperelliptic.org, c.v.vredendaal@tue.nl
[2] Brightsight B.V.
Delftechpark 1, 2628 XJ Delft, The Netherlands
wakker@brightsight.nl

Abstract. Side-channel attacks are a powerful tool to discover the cryptographic secrets of a chip or other device but only too often do they require too many traces or leave too many possible keys to explore. In this paper we show that for side channel attacks on discrete-logarithm-based systems significantly more unknown bits can be handled by using Pollard's kangaroo method: if b bits are unknown then the attack runs in $2^{b/2}$ instead of 2^b. If an attacker has many targets in the same group and thus has reasons to invest in precomputation, the costs can even be brought down to $2^{b/3}$.
Usually the separation between known and unknown keybits is not this clear cut – they are known with probabilities ranging between 100% and 0%. Enumeration and rank estimation of cryptographic keys based on partial information derived from cryptanalysis have become important tools for security evaluations. They make the line between a broken and secure device more clear and thus help security evaluators determine how high the security of a device is. For symmetric-key cryptography there has been some recent work on key enumeration and rank estimation, but for discrete-logarithm-based systems these algorithms fail because the subkeys are not independent and the algorithms cannot take advantage of the above-mentioned faster attacks. We present ϵ-enumeration as a new method to compute the rank of a key by using the probabilities together with (variations of) Pollard's kangaroo algorithm and give experimental evidence.

Keywords: Side-channel attacks, template attacks, key enumeration, rank estimation, discrete logarithms, Pollard-kangaroo method, precomputation

1 Introduction

In security evaluations it is important to investigate how well cryptographic implementations fare against side-channel attacks (SCA's). Numerous of these attacks are known: In this paper we will be most interested in the Template Attacks introduced by Chari, Rao and Rohatgi [3]. These attacks are among the strongest possible attacks in the sense that they extract all possible information of the samples $\mathcal{S}$ measured from a secret-key operation. They assume access to an identical device as used by the target and use it to produce precise multivariate characterizations of the noise of encryptions with different keys. By a process of iterative classification they then attempt to derive the key used when measuring $\mathcal{S}$.
In [3] iterative classification was performed until only a few candidates for the key were left. We however accept being left with a larger number of possibilities of which we are 100% sure that they are the likeliest keys according to our attack results and then continue to enumerate the remaining keys. This might be faster than continuing to attack and hoping to get more significant results. The motivation for this approach is the guy in the security evaluation lab. He starts his template attack and if he were to spend a long enough time he might find the key. However if he is able

This work was supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005. Permanent ID of this document: c1c4c98f98c7ca3cb1b1f4208b95e8b8. Date: July 21, 2014.

to enumerate 2^b keys, then the lab guy might as well stop the measurements after reducing the space of the remaining keys to an interval of size 2^b. By pruning the space further he will only risk throwing out the actual key even though he could have enumerated it. Another motivation for our approach is that there are some implementations where it does not matter how long the guy stays in the lab, the power traces will not give away more information. In this case we will still have a (larger) space of possibilities we need to enumerate based on all the information we were able to get.

When we have the results of such a SCA we can take two approaches. The first is the black-box approach of key enumeration. A key enumeration algorithm takes the SCA results as input and return keys k in order of likelihood. The position of a k in such an ordering is called its rank r. For symmetric-key cryptography some research has been done on this subject. Pan, van Woudenberg, den Hartog and Witteman [9] described a sub-optimal enumeration method with low storage requirements and an optimal method that required more storage. Veyrat-Charvillon, Gérard, Renauld, and Standaert [16] improved the optimal algorithm to require less storage and be able to enumerate more keys faster.

The second way of looking at a SCA result is the white-box approach of rank estimation. This method is particularly relevant for security evaluation labs. Using modern technology it is feasible to enumerate 2^{50} to 2^{60} keys, but when a key is ranked higher, we can only say that its rank is higher than $\sim 2^{60}$ (the rank at which the memory is exceeded). For security evaluations of the encryption algorithm however more accuracy is required. A rank estimation algorithm is able to extract information about the rank r of k without enumerating all the keys. For symmetric cryptography like AES such an algorithm was put forth by Veyrat-Charvillion, Gérard and Standaert in [17].

Our contribution in this paper is to extend these works to public-key cryptography, specifically Elliptic-Curve Cryptology (ECC [8]). Using the (partial) information from a template attack on the subkeys of a key k used in Diffie-Hellman Key Exchange, k might be recovered by enumerating the most likely candidates. Contrary to the assumptions in the previously mentioned algorithms, the information on the subkeys is not independent for ECC and therefore we cannot use the existing algorithms. On the bright side, in ECC we are working with cyclic groups and we can use this structure to speed up enumeration. This way, enumeration can be interpreted as finding the solution to the Discrete Logarithm Problem (DLP) by using partial information on the subkeys. We present our own algorithm which utilizes Pollard's kangaroo methods (see [10], [11], [4], [13], [15]) to enumerate with an error margin ϵ. In trade for this error margin we are able to enumerate a space of ℓ keys in $O(\sqrt{\ell})$ group operations.

If we make use of a precomputation table like was proposed in [1] we can reduce the expense per key to $O(\sqrt[3]{\ell})$ operations. This improvement of Pollard's methods lends itself particularly well to use in side-channel attacks. The creation of the precomputation table costs $O(\sqrt[3]{\ell^2})$ group operations and a specific table can only be used to speed up solving of DLPs in one particular group and interval length. So creating the table is only useful if we want to perform an attack a lot of times on the same length interval of a certain subgroup, but this is a fairly common scenario since many smart card vendors implement the NIST P-256 curve for which the curve equation and the base point are standardized. This means that security evaluation labs can create the most commonly needed tables beforehand and re-use them every time an attack is performed on an implementation for this curve.

We end this introduction with noting that even though the results in this paper are posed in the setting of elliptic curves, the techniques are applicable to solving a DLP in any cyclic group.

2 Background

This section gives a short description of methods to solve the Discrete Logarithm Problem (DLP) in a group of prime order n. The "square-root methods" solve this problem on average in $O(\sqrt{n})$ group operations. We will use additive notation since the main application is to is to solve the DLP on an elliptic curve E over a finite field $\mathbb{F}_p$ but the methods work in any group. Let $P, Q \in E(\mathbb{F}_p)$ be in a cyclic group; the goal is to find an integer k such that $Q = kP$.

2.1 A Short History of Discrete Logarithm Algorithms

A well known method is Shanks' Baby-Step-Giant-Step (BSGS) method [12]. It uses a table to find collisions between baby steps $0P, P, \ldots, (m-1)P$ and giant steps $Q - 0P, Q - mP, Q - 2mP, Q - 3mP, \ldots$, where $m \approx \sqrt{n}$. This finds $k = k_0 + k_1 m$ as the collision of $k_0 P$ and $Q - k_1 mP$ in $O(m)$ steps. A drawback of this method it that it has a storage requirement of m elements, which is a more serious limitation than $O(m)$ computation. If the discrete logarithm k is known to lie in an interval $[a, b]$ of length ℓ then choosing $m \approx \sqrt{\ell}$ gives a runtime of $O(\sqrt{\ell})$.

The Pollard-ρ method [10] gives a solution to the storage problem. It uses a deterministic random walk on the group elements with the goal of ending up in a cycle (which can be detected by Floyd's cycle finding algorithm). The walk is defined in such a way that the next step in the walk depends solely on the representation of the current point and that a collision on the walk reveals the discrete logarithm k. Van Oorschot and Wiener [15] introduced a parallel version of the Pollard-ρ method which gives a linear speed-up in the number of processors used. They use distinguished points: a point is a distinguished point if its representation exhibits a certain bit pattern, e.g., has the top 20 bits equal to zero. Whenever one of the parallel random walks reaches such a point it is stored on a central processor. A collision between two of these distinguished points almost surely reveals the value of the key. This method is an improvement over BSGS in that the storage requirements are minimal, but the algorithm is probabilistic and it cannot be adapted to search an interval efficiently.

Pollard's kangaroo method solves the latter problem. It reduces the storage to a constant and is devised to search for the solution of a DLP in an interval of length ℓ. The mathematical ingredients, the algorithm and improvements are the topic of the remainder of this section.

2.2 Mathematical Aspects of Kangaroos

To adequately explain Pollard's kangaroo method we first have to dive into the notion of a mathematical kangaroo. We define a kangaroo by the sequence of its positions $X_i \in \langle P \rangle$. Its starting point is $X_0 = s_0 P$ for a certain starting value s_0 and the elements that follow are a pseudo-random walk. The steps (or rather jumps) of a kangaroo are additions with points from a finite set of group elements $S = \{s_1 P, \ldots, s_L P\}$.

The step sizes s_i are taken such that their average is $s = \beta\sqrt{\ell}$ for some scalar β. To select the next step we use a hash function $H : \langle P \rangle \to \{1, 2, \ldots, L\}$ and we compute the distance by defining $d_0 = 0$ and then updating it for every step as follows

$$\begin{aligned} d_{i+1} &= d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots, \\ X_{i+1} &= X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots. \end{aligned}$$

This results in a kangaroo which after i jumps has travelled a distance of d_i and has value $(s_0 + d_i)P$.

2.3 Pollard's Kangaroo Method

The original algorithm that Pollard presented in [10] works as follows: Suppose we know that the value k in $Q = kP$ is in the interval $[a, b]$ of length $\ell = b - a + 1$. We introduce two kangaroos. The first one is the tame kangaroo T and we set it down at the point bP. We tell him to take $N = O(\sqrt{\ell})$ jumps and then stop. The point at which he stops and how far the kangaroo travelled to get there are recorded. This information can be seen as a trap meant to catch the second kangaroo. The trap consists of the endpoint $X_N = (b + d_N)P$ and the travelled distance d_N. Then a second, wild, kangaroo W is let loose at the unknown starting point $X'_0 = Q = kP$ following the same instructions determining jumps. The crucial fact upon which this algorithm is based is that if at any point the path of the wild kangaroo crosses with that of the tame one, meaning that they land on the same point, their remaining paths are the same. So if the wild kangaroo starts

jumping and crosses the tame one's path, then there is a jump M at which $X'_M = X_N$. From this we have $(k + d'_M)P = (b + d_N)P$ and $k = b + d_N - d'_M$ and we will detect this collision since $X'_{M+j} = X_{N+j}$, so the wild kangaroo will eventually meet the tame one.

Van Oorschot and Wiener [15] also presented a parallel version of this algorithm which even for just two kangaroos gives an improvement. Here instead of one trap, multiple traps are set: a fraction $1/w$ of the group elements which satisfy a certain distinguishing property $\mathcal{D}$ are defined as the distinguished set. Here w is taken to be $\alpha\sqrt{\ell}$, where α is some small constant (usually smaller than 1). A central server then records the distance and location of any kangaroo reaching a point in $\mathcal{D}$. We again have a wild and a tame kangaroo. Instead of starting at the end of the interval however, the tame kangaroo now starts in the middle at some point cP. Instead of finishing their entire paths, the kangaroos now jump alternately. Whenever one of them jumps to a distinguished point, their relevant information (position, distance, offset of starting point, kangaroo type) $= (X_i, d_i, c_i, Y)$ is recorded in a hash table which hashes on the X_i. When two kangaroos of different types have been recorded in the same entry of the hash table, we can derive the answer to the DLP.
Let us analyze this algorithm. The distance between the two kangaroos at the starting position is at most $(a - b + 1)/2 = \ell/2$ and on average $\ell/4$. If we take $s = \beta\sqrt{\ell}$, then the average number of jumps needed for the trailing kangaroo to catch up to the starting point of the front kangaroo is $\ell/4s$. Now we use that the probability of missing the front kangaroo's trail after passing its starting point for i steps is $(1-1/s)^i \approx e^{-i/s}$ and get that it takes s steps on average to hit the trail. Lastly, both kangaroos have to hop far enough to hit a distinguished point. If $1/w$ is the fraction of group elements that are distinguished, then w steps are needed on average before hitting a distinguished point. The total average number of steps is the $2(\ell/4s + s + w) = 2(\alpha + \beta + 1/4\beta)\sqrt{\ell}$, in which α and β can be optimized experimentally.
This algorithm can be improved even further by using 3 or 4 kangaroos (see [4]), but in this paper we consider the 2-kangaroo version.

2.4 Pollard's kangaroo method with precomputation

Pollard's kangaroo method can be sped up using precomputation. Bernstein and Lange [1] suggest to first produce a table of T distinguished points. Selecting the distinguished-point property and creating the table is then similar to setting a trap at every distinguished point in a desert, then sending a bunch of kangaroos into said desert and recording in a table which traps are most popular. Then when we are ready to send in the wild kangaroo we really want to trap we already know where he is most likely to fall in.
The algorithm works as follows. In the precomputation phase we start a lot of walks from random values yP and continue these walks until they reach a distinguished point at $(y + d)P$. We record $(y + d, (y + d)P)$ in the precomputation table $\mathcal{T}$. We keep starting new walks until T different distinguished points are found (or sample for longer and keep the most popular ones). As [1] notes, these walks should be independent from Q.
In the main computation phase we start random walks that are dependent on Q. We let a kangaroo on $X'_0 = Q$ hop until it reaches a distinguished point. If this point is in the table we have solved our DLP. If not we start a new walk at $Q + xP$ for some small random x. For each new walk we use a new randomization of Q and continue this way until a collision is found.

If enough DLPs in this group need to be solved so that precomputation is not an issue — or if a real-time break of the DLP is required — they propose to use $T \approx \sqrt[3]{\ell}$ precomputed distinguished points, walk length $w \approx \alpha\sqrt{\ell/T}$, i.e., $w \approx \alpha\sqrt[3]{\ell}$, with α an algorithm parameter to be optimized, and a random step set $\mathcal{S}$ with mean $s \approx \ell/4w$. This means that the precomputed table $\mathcal{T}$ takes $O(\sqrt[3]{\ell^2})$ group operations to create but can be stored in $\approx \sqrt[3]{\ell}$. This reduces the number of group operation required to solve a DLP to $O(\sqrt[3]{\ell})$ group operations, essentially a small number of walks.

3 ϵ-Enumeration

Now that we have the mathematical background covered, we continue to enumeration in side-channel attacks. Our goal is to enumerate through SCA results and give a rank estimation for a key similar to what was done in [16] and [17] for SCAs on implementations of symmetric cryptography. To do this we first have to model the result of SCA on ECC.

3.1 The Attack Model

We will assume a template attack like Chari, Rao and Rohatgi performed in [3]. In this attack we use our own device to make a multivariate model for each of the possibilities for the first bits of the key. When we then get a power sample of the actual key, we compute the probability of getting this shape of power sample given the template for each possible subkey. These probabilities will be our SCA results.
We can iterate this process by taking a few bits more and creating new templates, but making these requires a lot of storage and work. At the same time, they will either confirm earlier guesses or show that the wrong choice was made. At each iteration we only include the most likely subset of the subkeys from the previous iteration. Discarding the other possibilities creates a margin of error that we want to keep as small as possible.
In Chari [3] the aim was to recover *all* the bits of the key with a high success rate. Our goal is to minimize the overall attack time — the time for the measurements plus the time for testing (and generating) key candidates. We do not require to only be left with a couple of options at the end of the attack of which one is correct with a high probability. We accept ending the experiments being left with a larger number of possibilities of which we are close to 100% sure that they are the likeliest keys according to our attack results. After this we wish to enumerate them in the order of their posterior probabilities. We show how this can be faster than continuing to measure and hoping to get more significant results.
The results of the measurement and evaluation can be visualized as depicted in figure 1.

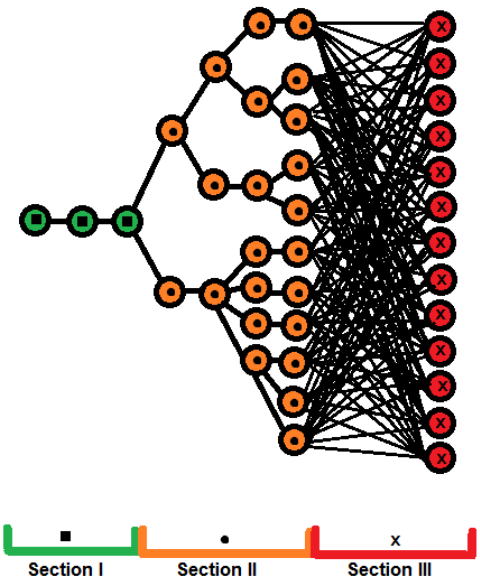

Fig. 1. The graphical representation the SCA result

We will call this visualization the enumeration tree. It consists of three sections. The first section consists of the subkey bits that we consider recovered. After the iterations of the template attack all options but one were discarded. Section II consists of the subkeys that we have partial information on, but not enough to reduce the number of likely subkeys to one. The last section consists of the subkeys that we have very little to no information on. It contains all possible subkeys per layer.
The idea is that each node $n_{i,j}$ in the representation is located in the i'th level (starting with 1 at the root) and corresponds to the j'th choice for subkeys $k_{i,1}, k_{i,2} \ldots,$. For each node $n_{i,j}$ there is an associated subkey $k_{i,j}$ and a posterior probability $q_{i,j}$. This is the probability that subkey i is equal to $k_{i,j}$ given that the path up to its parent node is correct. So if the nodes on the path to $n_{i,j}$ are $n_{1,x_1}, n_{2,x_2}, \ldots, n_{i-1,x_{i-1}}, n_{i,j}$, then the probability $q_{i,j}$ associated with this node is

$$q_{i,j} = \Pr[k_i = k_{i,j} | k_1 = k_{1,x_1}, k_2 = k_{2,x_2}, \ldots, k_{i-1} = k_{i-1,x_{i-1}}]. \tag{1}$$

Then we can also associate with each node a probability $p_{i,j}$ that represents the posterior probability of the key consisting of the subkeys represented by the nodes leading up to it (including itself). This probability is then

$$p_{i,j} = q_{i,j} \cdot \prod_{h=1}^{i-1} q_{1,x_h}. \tag{2}$$

In sections I and II the subkeys that were discarded during the attack and are not in the model might have a combined small probability p_ϵ. We assume that these probabilities are negligible (otherwise more nodes should be included in section II or the transition between section I and II should have moved closer to the root) and thus assume that the sum of the probabilities $p_{i,j}$ of each level of the tree is 1.

3.2 Enumeration in an Interval

The brute-force approach an attacker could take to enumerate the keys is to sort the nodes of the rightmost layer of section II by posterior probability p_{ij} and then for each choice brute-force all the options in section III. However using the algorithms from Sections 2.3 and 2.4 we can do better. Enumerating section III of a node is equivalent to computing the keys in an interval. Therefore we can use the Pollard-kangaroo algorithms to speed up the enumeration. The downside of this approach is that without searching the whole interval we can never say with 100% certainty that the key is not in the interval. However, in return we are able to speed up enumeration in an interval of size ℓ to $O(\sqrt{\ell})$ group operations or even to $O(\sqrt[3]{\ell})$ if we have the luxury of a precomputation table. We do have to note that even though we will call this process of searching the interval enumeration, it is a different kind than the enumeration in [16]. In that algorithm each key enumerated had to be checked for correctness against the encryption in an exhaustive-search manner. Using the kangaroo algorithms means that we search for a collision between group elements and only after this happens we can compute and double-check correctness against the public key of the cryptosystem attacked. This is much more sophisticated and much faster than the brute-force approach of having to check every key. The rank r of k now reflects the number of group operations required to find k after the end of the experimental session. It also means that we have only $O(\sqrt{\ell})$ ranks and they are dependent on the parameters used in the algorithm. To accurately reflect the uncertainty in this kind of enumeration, we introduce the following definition.

Definition 1. *Let the key $\hat{k}$ that we wish to find have rank $\hat{r}$. In an **ϵ-enumeration** we check keys in such a way that when we have enumerated up to rank r, then there is a $(1-\epsilon)$ probability that $\hat{r} > r$.*

If we want to perform such an ϵ-enumeration we have to have a stopping criterion. This criterion dictates how many group operations we have to do in order to get the probability of having missed our actual key below the ϵ bound. We have the following theorem.

Theorem 1. *Assume that the private key $\hat{k}$ lies in the interval of size ℓ. Let the average step size of the kangaroos be $s = \beta\sqrt{\ell}$. Let the probability of hitting a distinguished point be $\theta = c/\sqrt{\ell}$ and assume the distinguished points are spread uniformly over the whole group. Lastly we assume that the hash function H and the step set in the improved Pollard-kangaroo algorithm of Section 2.3 is sufficiently random. Then for $x > \ell/(4s)$ the average probability of not finding $\hat{k}$ in $2x$ steps, i.e. x hops per kangaroo, of that algorithm is*

$$\epsilon(x) = e^{-\frac{x}{s}+\frac{\ell}{4s^2}} + (e^{\theta\left(\frac{\ell}{4s}+2-x\right)} - e^{2\theta-\frac{1}{s}\left(x-\frac{\ell}{4s}\right)})/(s - se^{\left(\theta-\frac{1}{s}\right)}). \tag{3}$$

Proof. Recall that in this algorithm we had 2 kangaroos placed in the interval and they alternate their jumps. In this proof we analyze the x steps of the back kangaroo and compute the average probability that it does not collide with the front kangaroo even though they were both placed in the same interval. First the back kangaroo needs to catch up to the front one. The number of steps to do this is on average $\ell/4s$. Given that the back kangaroo takes x steps we now have $y = x - \ell/(4s)$ steps left on average. To avoid a recorded collision in these remaining y steps we either have to avoid the trail of the front kangaroo, or hit it after i steps and avoid distinguished points for the next $y - i$ steps. We assumed the hash function H to be sufficiently random, so the average probability of avoiding the trail is $(1 - 1/s)$ for each step taken and the chance of missing a distinguished point is $(1 - \theta)$ in each step. Thus we have the average approximate probability of avoiding detected collisions as follows

$$\left(1-\frac{1}{s}\right)^{x-\frac{\ell}{4s}} + \sum_{i=0}^{x-\frac{\ell}{4s}-1}\left(1-\frac{1}{s}\right)^{i}\frac{1}{s}\,(1-\theta)^{x-\frac{\ell}{4s}-i-2}.$$

We can approximate the second part of this equation as follows

$$\sum_{i=0}^{x-\frac{\ell}{4s}-1}\left(1-\frac{1}{s}\right)^{i}\frac{1}{s}\,(1-\theta)^{x-\frac{\ell}{4s}-i-2} \approx \frac{1}{s}\sum_{i=0}^{x-\frac{\ell}{4s}-1} e^{\frac{-i}{s}}e^{-\theta\left(x-\frac{\ell}{4s}-i-2\right)} = \frac{e^{\theta\left(\frac{\ell}{4s}+2-x\right)}}{s}\sum_{i=0}^{x-\frac{\ell}{4s}-1} e^{\frac{-i}{s}}e^{\theta i}.$$

This in turn then evaluates to

$$\frac{e^{\theta\left(\frac{\ell}{4s}+2-x\right)}}{s}\sum_{i=0}^{x-\frac{\ell}{4s}-1}\left(e^{\theta-\frac{1}{s}}\right)^{i} = \frac{e^{\theta\left(\frac{\ell}{4s}+2-x\right)}}{s}\cdot\frac{1-e^{\left(\theta-\frac{1}{s}\right)\left(x-\frac{\ell}{4s}\right)}}{1-e^{\left(\theta-\frac{1}{s}\right)}} = \frac{e^{\theta\left(\frac{\ell}{4s}+2-x\right)} - e^{2\theta-\frac{1}{s}\left(x-\frac{\ell}{4s}\right)}}{s - se^{\left(\theta-\frac{1}{s}\right)}}.$$

So indeed we have our average probability of

$$\epsilon(x) = \left(1-\frac{1}{s}\right)^{x-\frac{\ell}{4s}} + \frac{e^{\theta\left(\frac{\ell}{4s}+2-x\right)} - e^{2\theta-\frac{1}{s}\left(x-\frac{\ell}{4s}\right)}}{s - se^{\left(\theta-\frac{1}{s}\right)}} \approx e^{-\frac{x}{s}+\frac{\ell}{4s^2}} + \frac{e^{\theta\left(\frac{\ell}{4s}+2-x\right)} - e^{2\theta-\frac{1}{s}\left(x-\frac{\ell}{4s}\right)}}{s - se^{\left(\theta-\frac{1}{s}\right)}}.$$

□

Note that this equation is only valid for values of $x > \frac{\ell}{4s}$, otherwise $\epsilon(x) = 1$.

We now analyze the situation of using Pollard's kangaroo algorithm with a precomputation table. For this we make a hypothesis on the distribution of distinguished points and the number of points covered by each walk: Let the precomputation table $\mathcal{T}$ consist of the first found T different distinguished points $D_i = t_i P$. Let the average walk length be $w = \alpha\sqrt{\ell/T}$ and the average step size of the kangaroos be $s \approx \ell/(4w)$ such that the average distance of a walk is $\approx \ell/4$. Since the points in $\mathcal{T}$ are different their paths are disjoint. They cover on average Tw points. Assume that these points are uniformly distributed over $\{P, 2P, \ldots, \gamma\ell P\}$ for some value of γ. In Section 4 we will present experiments showing that $\gamma = \max_{1\le i\le T} t_i/\ell - \min_{1\le i\le T} t_i/\ell$ is a good fit.

Theorem 2. *Let $\hat{k}$ lie in an interval of size ℓ. Let the average walk length be $w = \alpha\sqrt{\ell/T}$ and the average step size of the kangaroos be $s \approx \ell/(4w)$. Under the hypothesis made above, $\mathcal{T}$ represents tW points distributed uniformly over $\{P, 2P, \ldots, \gamma\ell P\}$ for some value of γ. The average probability that the Pollard-kangaroo algorithm with precomputation (Section 2.4) does not find $\hat{k}$ in y independent walks of the algorithm is*

$$\epsilon(x) = e^{\frac{-\alpha^2 y}{\gamma}}. \tag{4}$$

Proof. Under the hypothesis the probability of the wild kangaroo hitting the trail of one of the table points' kangaroos is on average $(Tw)/(\gamma\ell) = \alpha^2/(\gamma w)$ at each step. Since the walk takes on average w steps the probability of avoiding a collision is

$$\left(1 - \alpha^2/(\gamma w)\right)^w \approx e^{\frac{-\alpha^2}{\gamma}}.$$

We assume independent walks, so we have that the probability that after y walks we have not found a collision is

$$\prod_{i=1}^{y} e^{\frac{-\alpha^2}{\gamma}} = e^{\frac{-\alpha^2 y}{\gamma}}.$$

which is the result we desired. □

3.3 Further Considerations and Optimizations

Combining Intervals If we have adjacent intervals in the enumeration tree we might combine these intervals to speed up the search. If they are of the same length then searching the intervals separately simply means searching twice as long. Combining two intervals in the kangaroo method reduces the search time by a factor $\sqrt{2}$. When we do this we do have to take the posterior probabilities of the intervals into account. If we simply combine all adjacent intervals in the enumeration tree and search them in the order of the subinterval with the highest posterior probability then it might happen that an interval ranked high separately is not searched because it is not part of some large combined interval. We therefore only combine intervals if they also have subsequent posterior probabilities. For the precomputation case of the algorithm we also have to take the availability of tables into account. We only combine intervals if we have a table and step set corresponding to that newly created interval length.

Restarts We described the general kangaroo algorithm to have the kangaroos continue along their paths after finding a distinguished point. For the standard rho method [2] show the benefits of restarting walks after a distinguished point is found. If we do not use a precomputation table then doing restarts means redoing the initial phase of the two kangaroos catching up to each other and the error function will decrease at a slower rate. This is only advantageous if the kangaroos ended up in a loop they cannot get out. If we detect such a loop, the kangaroo(s) can be restarted. If $\ell \ll n$ the probability of ending in a loop is very small. On the other hand, we do not have the problem of the initial catching up phase. Therefore we restarted walks if they exceeded $20w$ steps. An advantage of using the improved Pollard-kangaroo algorithm without precomputation tables is that there is a probability of finding the solution of a DLP in an adjacent interval because the kangaroos naturally venture out in the direction of larger discrete logarithms. This is also an argument against doing restarts. Even though the current interval was chosen for the good reason of having the highest posterior probability among those not considered, yet, it is an added benefit that one might accidentally find a solution in another interval. If the tame kangaroo is started in interval I_1 of size ℓ, but the key was actually in adjacent interval I_2, then after a longer initial catch-up phase there is a probability of collisions. We could estimate this probability with an error function like we did for I_1 to reduce search time, but the longer the kangaroos jump the bigger the variance gets and the less accurate the error function is going to be. Therefore we do not advise to include this extra probability into the considerations.

Parallelization There are two levels at which the ϵ-enumeration method can be parallelized: One level is the underlying kangaroo algorithm using distinguished points; the second level is dividing the intervals over the processors used, i.e., we could simply place one interval search on each processor, or we could have all processors search one of the intervals, or use a combination of the two. Using multiple processors for a single interval only makes sense if the interval is sufficiently large and many walks are needed (so rarely with precomputation) and if the posterior probability is significantly higher. If a lot of intervals have a similar probability it might be better to search them in parallel.

ϵ-Rank Estimation Now that we have a definition of ϵ-enumeration we can easily extend it to estimating ranks of keys that we cannot ϵ-enumerate in feasible time. To do this we have to adapt the attack on the device using the key $\hat{k}$. When discarding keys of a too low probability from future templates we do store the subkey in the enumeration tree with their probabilities. They are however not included in new templates, so the corresponding branch of the enumeration tree will not grow any more. After finishing the measurement we can determine with the error function for each interval with a higher posterior probability than the one that contains $\hat{k}$ how many steps we would (on average) take in this interval. The sum of these quantities is then an estimated lower bound for the rank of $\hat{k}$. We can use a similar method to determine an estimated upper bound.

4 Experimental Results

This section presents representative examples of our implementations. We ran our experiments on a Dell Optiplex 980 using one core of an Intel Core i5 Processor 650 / 3.20GHz. We re-used parts of the Bernstein/Lange kangaroo C++ code used for [1]. Our adaptations will be posted at `http://www.scarecryptow.org/publications/sckangaroos`. For ease of implementation we used the group $\mathbb{F}_p^*$ as was used in [1], which uses a "strong" 256-bit prime (strong meaning that $\frac{p-1}{2}$ is also prime) and a generator g, which is a large square modulo p. Although in the previous sections we focussed on elliptic curves, both those and $\mathbb{F}_p^*$ are cyclic groups and thus these results hold for both. We set the interval size to $\ell = 2^{48}$ and at each run took a random h in the interval for a new DLP.

For the experiments without precomputation we made use of distinguished points to find the collision, which were recorded in a vector table that was searched each time a new distinguished point was found. We chose the probability of landing in a distinguished point to be $2^{-19} = \frac{2^5}{\sqrt{\ell}}$ by defining a point as distinguished if the least-significant 19 bits in its representation were zero, i.e., if the value modulo $w = 2^{19}$ was zero. The step function selected the next step based on the value modulo 128, the 128 step sizes were taken randomly around $\sqrt{\ell}$.

Step sets The goal of this paper is not to find the optimal parameters for the kangaroo algorithm. The choice of step set is however relevant for usability of the error function of Theorem 3. As can be seen equation 3 only uses the mean of the step set and not its actual values, so it is possible to create one that will not adhere to the error function at all. Even if we choose the step set randomly it can contain dependencies and this makes the error function less accurate. We can see this in the top graph of figure 2.

We did 8192 experiments and saw that the error function in blue is a rather good approximation for the fraction of unsolved DLPs in red for the first 10 million steps of step set $\mathcal{S}_1$ of the wild kangaroo and from 25 million onward. In between these values we see some unexpected behavior. It might be that our step set contains some dependencies, e.g., it might be that the step set contains too many steps in a certain equivalence class; meaning that the probability of missing the trail is larger than $(1 - 1/s)$ per step. We were not able to visually identify what the problem was. By trying a different seed for the random step set we found $\mathcal{S}_2$ which behaved nicely according to our expectations as can be observed in the bottom graph of figure 2. For concrete attacks it is advisable to run a few tests to check the quality of the step function.

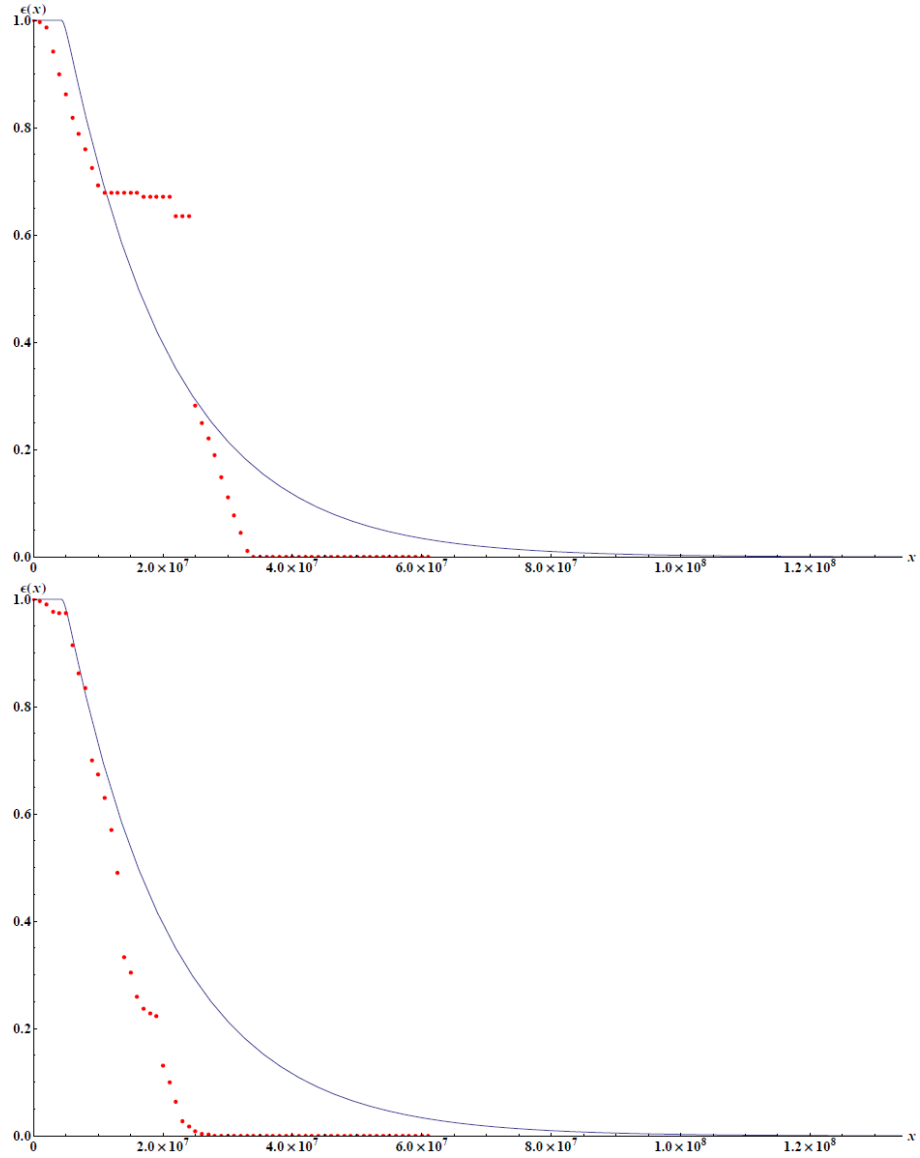

Fig. 2. The theoretic function $\epsilon(x)$ for the kangaroo method without precomputations and the experimental results using two random step sets $\mathcal{S}_1$ and $\mathcal{S}_2$ with $\beta \approx 1$. Top: $\beta = 0.978$. Bottom: $\beta = 0.971$

Combining intervals We were able to find similar results for intervals where we combined two adjacent intervals of length 2^{48} to one of size 2^{49}. We saw the same problem of good and bad step sets. With trying three step sets, we got a step set that had the error function as an upper bound for the experiments. These graphs confirm that by combining the intervals we can search twice the keyspace in approximately $\sqrt{2}$ times the steps.

Using precomputation We again searched for DLPs in an interval of length 2^{48}. Our 128 step sizes however were now uniformly chosen between 0 and $\ell/4w$ instead of around $\beta\sqrt{\ell}$ for some β. Each DLP $h = g^y$ was chosen randomly and each walk starting from it was randomized in the interval between $y - 2^{40}$ and $y + 2^{40}$. For the precomputation we used a table of size $N = T = \sqrt[3]{\ell} = 2^{16}$. We used the first T distinguished points found as table points and computed $\gamma = 1.923$ as $\max_{1\le i\le T} t_i/\ell - \min_{1\le i\le T} t_i/\ell$, for the T table elements of the form g^{t_i}. We used an average walklength of $w = 2^{15}$ such that $\alpha = 0.5$. Using 2097152 experiments we got the results on the top of figure 3.

We see that the error function is a good upper bound for the experimental results. We continued with the same experiment for an interval of length $\ell = 2^{50}$. We used a table of $T = 104032 \approx \sqrt[3]{\ell}$ table points and found $\gamma = 1.853$. We used an average step set of $w = 2^{16}$ such that $\alpha \approx 0.630$. Using 2097152 experiments we got the results on the bottom of figure 3. We again see that the error function is a good approximation for the experiments.
Other parameters than α and γ can influence the performance of the algorithm. The error function does not reflect information on the step set other than its mean, nor on how often distinguished points were found but it relies on the hypothesis that the table covers about wT points and these are uniformly distributed over $\{P, 2P, \ldots, \gamma P\}$.

Considerations about the step set are even more relevant when preparing the precomputation table by making $N > T$ walks and selecting for $\mathcal{T}$ the T points with the largest number of ancestors (longest walks leading up to them, distinguished points found with multiplicity).

For the ϵ-enumeration we suggest to include a parameter σ in the exponent of the error function $\epsilon(y) = e^{\sigma\alpha^2 y/\gamma}$ that is determined experimentally. After determining the step set and table we can run the algorithm on different randomly chosen DLPs, much like we did in our experiments, and determine a value for σ. After this the error function is ready to be used in a security evaluation.

ϵ-Enumeration The result is that according to our best found results we can ϵ-enumerate in an interval of length ℓ in the steps displayed in table 1.

ϵ	$1.0 \cdot 10^{-1}$	$1.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-5}$	$1.0 \cdot 10^{-7}$	$1.0 \cdot 10^{-9}$
$N = T = 0, \sigma = 1$	$4.2 \cdot \sqrt{\ell}$	$10.8 \cdot \sqrt{\ell}$	$17.6 \cdot \sqrt{\ell}$	$24.3 \cdot \sqrt{\ell}$	$31.0 \cdot \sqrt{\ell}$
$N = T = \sqrt[3]{\ell}, \sigma = 1$	$18 \cdot \sqrt[3]{\ell}$	$54 \cdot \sqrt[3]{\ell}$	$89 \cdot \sqrt[3]{\ell}$	$124 \cdot \sqrt[3]{\ell}$	$160 \cdot \sqrt[3]{\ell}$
$N = T = \sqrt[3]{\ell}, \sigma = 1.12$	$16 \cdot \sqrt[3]{\ell}$	$48 \cdot \sqrt[3]{\ell}$	$79 \cdot \sqrt[3]{\ell}$	$111 \cdot \sqrt[3]{\ell}$	$142 \cdot \sqrt[3]{\ell}$
$N = 2T = 2 \cdot \sqrt[3]{\ell}, \sigma = 1$	$5 \cdot \sqrt[3]{\ell}$	$14 \cdot \sqrt[3]{\ell}$	$23 \cdot \sqrt[3]{\ell}$	$32 \cdot \sqrt[3]{\ell}$	$41 \cdot \sqrt[3]{\ell}$
$N = 2T = 2 \cdot \sqrt[3]{\ell}, \sigma = 1.28$	$4 \cdot \sqrt[3]{\ell}$	$11 \cdot \sqrt[3]{\ell}$	$18 \cdot \sqrt[3]{\ell}$	$25 \cdot \sqrt[3]{\ell}$	$32 \cdot \sqrt[3]{\ell}$
$N = 8T = 8 \cdot \sqrt[3]{\ell}, \sigma = 1$	$5 \cdot \sqrt[3]{\ell}$	$14 \cdot \sqrt[3]{\ell}$	$23 \cdot \sqrt[3]{\ell}$	$31 \cdot \sqrt[3]{\ell}$	$40 \cdot \sqrt[3]{\ell}$
$N = 8T = 8 \cdot \sqrt[3]{\ell}, \sigma = 1.40$	$4 \cdot \sqrt[3]{\ell}$	$10 \cdot \sqrt[3]{\ell}$	$16 \cdot \sqrt[3]{\ell}$	$23 \cdot \sqrt[3]{\ell}$	$29 \cdot \sqrt[3]{\ell}$

Table 1. Required group operations for ϵ-enumeration

There are a couple of remarks to be made. We took only semi-optimized parameters. What we mean by this is that we did some experiments to find a reasonable step set, but as the purpose of our research was not to find the best parameters for Pollard-kangaroo algorithms, we did not fully optimize the step set. The results might thus be improved with optimal parameters. We observe that the higher N is relative to T, the fewer group operations are necessary to drop below the margin of error. Increasing the value of N improves the functionality of the algorithm more

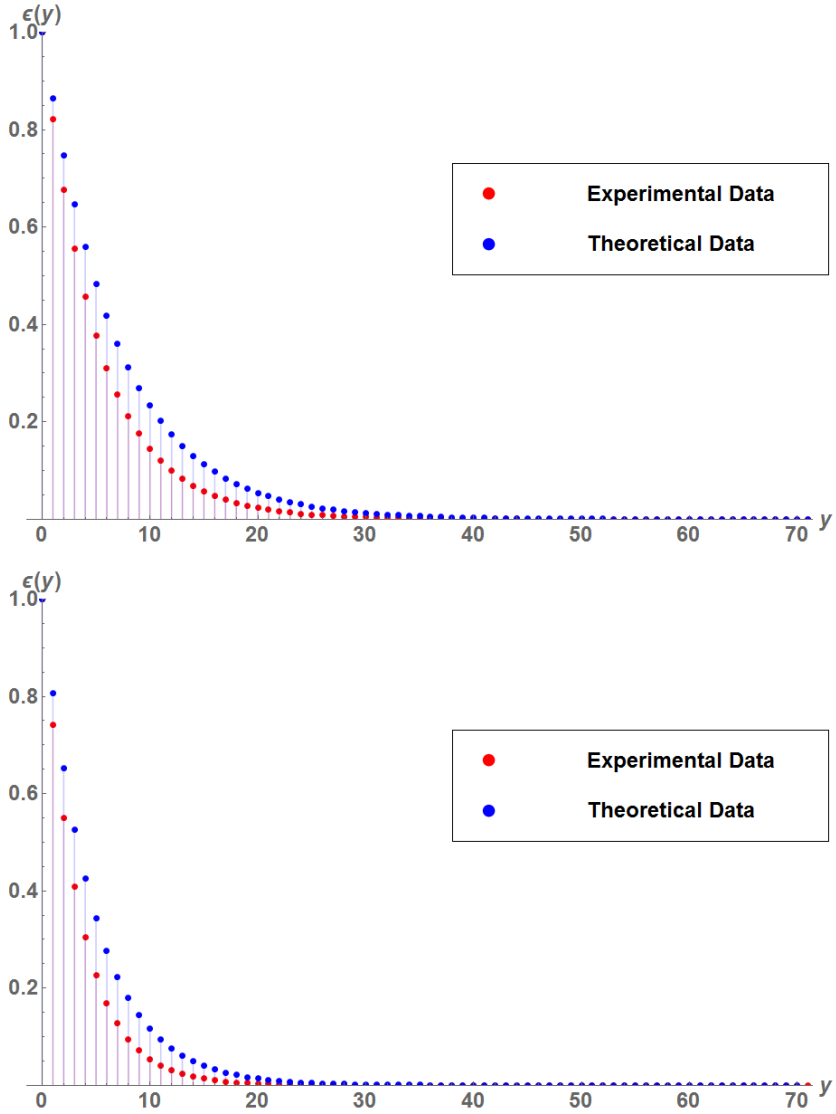

Fig. 3. The theoretic function $\epsilon(y)$ for the kangaroo method with precomputation and the experimental results using a step set with $s \approx \frac{\ell}{4w}$. Top: $\ell = 2^{48}$. Bottom: $\ell = 2^{50}$.

than the variables α and γ reflect. This is seen in the experimental values of σ. It increases as N increases. We also see that increasing N from T to $2T$ makes a big difference in the required group operations. The effect of increasing N even further to $8T$ does not have the same magnitude.

Although even small speed-ups are always nice, we also have to take the time it takes to create the tables into account. For $N = T$ it took us just over 9 billion group operations and under 19 minutes to create the table. This is equal to $1.4\sqrt[3]{\ell^2}$ multiplications. When we increased N to $2T$ it took about 50 minutes and $3.8\sqrt[3]{\ell^2}$ group operations. Finally, when we took N up to $8T$ it took approximately 9 hours and 2.5 million walks of in total $161544244922 \approx 37.6\sqrt[3]{\ell^2}$ group operations. This is doable for evaluation companies, even if they have to make a lot of tables, but doing many more might not have enough yield for the time it takes.

We see that we can determine how many group operations we have to do on average for different degrees of confidence. If we increase the confidence by a factor 100 the constant c in $c\sqrt{\ell}$ or $c\sqrt[3]{\ell}$ increases linearly. This means that if we use the $N = 8T$ precomputation table and we do $170\sqrt[3]{\ell}$ steps in an interval of length 2^{48} we can 2^{-128}-enumerate it in less than $2^{23.5}$ group operations. This is a massive improvement over brute-force enumerating all 2^{48} keys in an interval. The new ranking method that is induced by such an enumeration is also a much more accurate measure of the security of a device. Security evaluation labs could more confidently estimate how secure an implementation is.

5 Comparison and Conclusion

This is the first paper studying key enumeration and rank estimates for public key cryptosystems. Gopalakrishnan, Thériault, and Yao [7] studied key recovery for ECC if a side-channel attack only provided some bits of the key. In contrast to our model they assume that the known bits are absolutely correct and do not discuss the possibility that we might have partial information on a subkey. If we were to make an enumeration tree of such a result it would solely consist of sections I and III. Although their assumption makes enumeration a lot easier it is not very realistic. Often there are too few bits fully recovered to make searching the remaining key space feasible. Using not only fully recovered bits but also the partial information we can search an interval smartly and possibly recover the solution to the DLP where [7] could not. Finally, they do not consider enumeration and rank computation.

5.1 Comparison

One important assumption of the model covered in this paper so far is that we have not only information about specific subkeys, but also that these keys are adjacent and start from the most significant bits. This is true for the very common case of implementations using windowing methods (including signed and sliding) starting from the most significant bits. However, we can adjust our method to the scenarios considered in [7] as we will now discuss.

The first scenario in their paper is that contiguous bits of the key are revealed. These bits can be the most significant bits, the least significant bits or be somewhere in the middle. So far we considered the first case but our model can be easily adapted to the others:

- If the least significant bits are revealed, then our tree would get inverted. Searching section III would then require a slight adaptation of the algorithms used on it. Searching it with for instance Pollard kangaroo would require searching in equivalence classes instead of an interval. This adaptation means the probability of finding our solution 'accidentally' in a neighboring interval becomes zero. Creating tables in the Bernstein/Lange precomputation is still possible; we would shift each instance of the DLP to the same equivalence class.
- If bits somewhere in the middle are revealed the model would become more complicated. We would get a bow-shaped model with 2 sections II and III. There are 5 sections; the third contains known bits, on the second and fourth we have partial information and we have no information on the remaining sections. Enumerating through the sections III would become more complicated, though not impossible.

The second scenario [7] poses is that the information is not on any specific bits, but on the square-and-multiply chain. This is actually no problem for our model. If we suppose the most reliable information on the chain is on the most significant bits, then the enumeration tree of figure 1 would become a binary tree. Searching the sections is the same as before. If the most reliable information on the square and multiply chain is on the middle part of the key or the least significant part the adaptations are similar to the contiguous bits scenario.

We now present an application that is not mentioned in [7] but is realistic for an ECC scenario. A common speed up for scalar multiplication using the base point P is to include $P' = 2^m P$ in the system parameters, where the group order n is of length $2m$, and compute kP as $(k_0 + 2^m k_1)P = k_0 P + k_1 P'$. This halves the number of doublings required to compute kP (see Straus [14]) and reduces the overhead of dummy instructions introduced to perform one addition per doubling. When such an implementation is attacked, we will know the MSBs of k_0 and k_1 with much higher probability than their lower bits. This results in an enumeration tree of six sections: sections I and IV contain known bits, for sections II and V we have partial information, and we have little to no information on sections III and VI. Enumeration in such a structure is not straightforward with the methods we presented so far. If section III is small enough, we can brute-force it and use ϵ-enumeration in section VI, but realistically sections III and VI have equal size. To compute the key we have to adapt the kangaroo algorithms to simultaneously hop intervals and equivalence classes. This is achieved by algorithms for multidimensional DLPs which have been studied by Gaudry and Schost in [6]. The running time is $O(\sqrt{\ell_1 \ell_2})$ if the section III and VI are intervals of length ℓ_1 and ℓ_2. An improved version of this algorithm was presented by Galbraith and Ruprai in [5]. We have not devised an error function for these algorithms, but expect results similar to Theorems 1 and 2.

5.2 Conclusion

In summary, we showed that kangaroos can be very useful in making SCA on ECC more efficient:

- Once section III is below 80 bits and section II not too wide there is no point in letting the lab guy do further measurements since a standard PC can casually do the 2^{40} group operations to break the DLP.
- In cases where measurements cannot be pushed further by physical limitations (restricted number of measurements, limits on what templates can be measured) our improvements allow retrieving the key in some situations in which previous methods could not.
- Theoretical kangaroos can be used to estimate the rank of the key in white-box scenarios to determine whether a sufficiently motivated attacker could mount the attack to break the system and we present error functions to use in ϵ-enumeration.

References

1. Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT*, volume 7668 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2012.
2. Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. On the correct use of the negation map in the pollard rho method. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 128–146. Springer, 2011.
3. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
4. Steven D. Galbraith, John M. Pollard, and Raminder S. Ruprai. Computing discrete logarithms in an interval. *Math. Comput.*, 82(282), 2013.
5. Steven D. Galbraith and Raminder S. Ruprai. An improvement to the Gaudry-Schost algorithm for multidimensional discrete logarithm problems. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 368–382. Springer, 2009.

6. Pierrick Gaudry and Éric Schost. A low-memory parallel version of Matsuo, Chao, and Tsujii's algorithm. In Duncan A. Buell, editor, *ANTS*, volume 3076 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2004.
7. K. Gopalakrishnan, Nicolas Thériault, and Chui Zhi Yao. Solving discrete logarithms from partial knowledge of the key. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 224–237. Springer, 2007.
8. Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
9. Jing Pan, Jasper G. J. van Woudenberg, Jerry den Hartog, and Marc F. Witteman. Improving DPA by peak distribution analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 241–261. Springer, 2010.
10. John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.
11. John M. Pollard. Kangaroos, monopoly and discrete logarithms. *J. Cryptology*, 13(4):437–447, 2000.
12. Daniel Shanks. Class number, a theory of factorization, and genera. In Donald J. Lewis, editor, *1969 Number Theory Institute*, volume 20 of *Proceedings of Symposia in Pure Mathematics*, pages 415–440, Providence, Rhode Island, 1971. American Mathematical Society.
13. Andreas Stein and Edlyn Teske. The parallelized Pollard kangaroo method in real quadratic function fields. *Math. Comput.*, 71(238):793–814, 2002.
14. Ernst G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70:806–808, 1964. URL: `http://cr.yp.to/bib/entries.html#1964/straus`.
15. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
16. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.
17. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2013.