Computing Low-Weight Discrete Logarithms

Bailey Kacsmar[†], Sarah Plosker[‡], and Ryan Henry[§]

[†] University of Waterloo, Waterloo ON, Canada (bkacsmar@uwaterloo.ca)
[‡] Brandon University, Brandon MB, Canada (ploskers@brandonu.ca)
[§] Indiana University, Bloomington IN, USA (henry@indiana.edu)

Abstract. We propose some new baby-step giant-step algorithms for computing "low-weight" discrete logarithms; that is, for computing discrete logarithms in which the radix-b representation of the exponent is known to have only a small number of nonzero digits. Prior to this work, such algorithms had been proposed for the case where the exponent is known to have low Hamming weight (i.e., the radix-2 case). Our new algorithms (i) improve the best-known deterministic complexity for the radix-2 case, and then (ii) generalize from radix-2 to arbitrary radixes $b > 1$. We also discuss how our new algorithms can be used to attack several recent Verifier-based Password Authenticated Key Exchange (VPAKE) protocols from the cryptographic literature with the conclusion that the new algorithms render those constructions completely insecure in practice.

Keywords: Discrete logarithms; baby-step giant-step; meet-in-the-middle; cryptanalysis; verifier-based password authenticated key exchange (VPAKE)

1 Introduction

In this paper, we deal with the problem of computing discrete logarithms when the radix-b representation of the exponent sought is known to have low weight (i.e., only a small number of nonzero digits). We propose several new baby-step giant-step algorithms for solving such discrete logarithms in time depending mostly on the radix-b weight (and length) of the exponent.

Briefly, the *discrete logarithm (DL) problem* in a multiplicative group $\mathbb{G}$ of order q is the following: Given as input a pair $(g, h) \in \mathbb{G} \times \mathbb{G}$, output an exponent $x \in \mathbb{Z}_q$ such that $h = g^x$, provided one exists. The exponent x is called a *discrete logarithm* of h with respect to the base g and is denoted, using an adaptation of the familiar notation for logarithms, by $x \equiv \log_g h \bmod q$. A longstanding conjecture, commonly referred to as the *DL assumption*, posits that the DL problem is "generically hard"; that is, that there exist infinite families of groups in which no (non-uniform, probabilistic) polynomial-time (in $\lg q$) algorithm can solve uniform random instances of the DL problem with inverse polynomial (again, in $\lg q$) probability.

Our results do not refute (or even pose a serious challenge to) the DL assumption. Indeed, although our algorithms are generic,[1] they do not apply to uniform random DL instances nor do they generally run in polynomial time. Rather, we demonstrate that, for certain *non-uniform* instance distributions, one

[1] In other words, our algorithms only require black-box oracle access to the group operation, its inverse, and an equality test; they can, therefore, be run over *any* finite group.

can solve the DL problem in time that depends mostly on a parameter strictly smaller than $\lg q$. Specifically, to solve a DL problem instance in which the radix-b representation of the exponent has length m and weight t, our fastest deterministic algorithm evaluates $(t + o(1))\binom{m/2}{t/2}(b-1)^{t/2}$ group operations and stores $2\binom{m/2}{t/2}(b-1)^{t/2}$ group elements in the worst case; for the same problem, our randomized (Las Vegas) algorithm evaluates fewer than $\sqrt{\frac{16t}{\pi}}\binom{m/2}{t/2}(b-1)^{t/2} + O(1)$ group operations and stores $\binom{m/2}{t/2}(b-1)^{t/2}$ group elements, on average. For the special case of radix-2, our fastest deterministic algorithm improves on the previous result (due to Stinson [30; §4.1]) by a factor $c\sqrt{t}\lg m$ for some constant c, reducing the number of group operations used from $\Theta(t^{3/2}\lg m\binom{m/2}{t/2})$ to $(t + o(1))\binom{m/2}{t/2}$. While a far cry from challenging established cryptographic best practices, we do observe that our new algorithms are not without practical ramifications. Specifically, we demonstrate a practical attack against several recent Verifier-based Password Authenticated Key Exchange (VPAKE) protocols from the literature [12–15, 35].

Organization. The remainder of this paper is organized as follows. In Section 2, we recall mathematical preliminaries necessary to frame our main results. In Section 3, we review and improve on several variants of an algorithm for solving the "low-Hamming-weight DL problem" and then, in Section 4, we present our new generalizations to arbitrary radixes $b > 1$. In Section 5, we review existing work that addresses some related "low-weight" DL variants. In Section 6, we showcase the cryptanalytic implications of the new algorithms by explaining how they can be used to attack several Verifier-based Password Authenticated Key Exchange (VPAKE) protocols from the literature.

2 Mathematical preliminaries

Throughout this paper, $\mathbb{G}$ denotes a fixed cyclic group with order q, which we express using multiplicative notation, and g denotes a fixed generator of $\mathbb{G}$. We are interested in computing the DLs of elements $h \in \mathbb{G}$ to the base g. We assume that the group order q is known, though our techniques work much the same when q is not known.

Radix-b representations. Let $b > 1$ be a positive integer (the "radix"). For every positive integer x, there exists a unique positive integer m and an m-tuple $(x_m, \ldots, x_1) \in \{0, 1, \ldots, b-1\}^m$ with $x_m \neq 0$ such that

$$x = \sum_{i=1}^{m} x_i \cdot b^{i-1}, \tag{1}$$

called the *radix-b representation* of x. Here the component x_i is called the *ith radix-b digit*, and $m = \lceil \log_b x \rceil$ the *radix-b length*, of x. The number of nonzero digits in the radix-b representation of x is called its *radix-b weight* (or simply its *weight* when the radix is clear from context). When $b = 2$, the radix-b weight of x is its *Hamming weight* and the radix-b length of x is its *bit length*.

Decomposing radix-b representations. Let m and t be positive integers. We write $[m]$ as shorthand for the set $\{1, 2, \ldots, m\}$ of positive integers less than or equal to m and, given a finite set A, we define $\binom{A}{t}$ as the set of all size-t subsets of A. We are especially interested in $\binom{[m]}{t}$, a collection of $\binom{m}{t}$ subsets equipped with a natural bijective mapping to the set of all (at-most)-m-bit positive integers with Hamming weight t. The mapping from $\binom{[m]}{t}$ to the set of such integers is given by the function $\mathrm{val}_{m,t} : \binom{[m]}{t} \to \mathbb{N}$ that maps each size-t subset $Y \in \binom{[m]}{t}$ to the integer $\mathrm{val}_{m,t}(Y) \coloneqq \sum_{i \in Y} 2^{i-1}$.

The above $\mathrm{val}_{m,t}$ function naturally generalizes to a family of two-operand functions parametrized by a radix $b > 1$. Specifically, for any integer $b > 1$, the function $\mathrm{val}_{b,m,t} : [b-1]^t \times \binom{[m]}{t} \to \mathbb{N}$ maps each t-tuple $X = (x_t, \ldots, x_1) \in [b-1]^t$ and size-t subset $Y \in \binom{[m]}{t}$ to the integer $\mathrm{val}_{b,m,t}(X, Y) \coloneqq \sum_{i=1}^{t} x_i \cdot b^{Y[i]-1}$. In the preceding notation, $Y[i]$ denotes the ith smallest integer in the set Y. Note that the function $\mathrm{val}_{b,m,t}$ is injective: the $(b-1)^t \binom{m}{t}$ possible inputs to $\mathrm{val}_{b,m,t}$ map to *pairwise distinct* positive integers, each having radix-b weight t and radix-b length at most m. Also note that when $b = 2$, the all-ones tuple $(1, 1, \ldots, 1)$ is the only element in $[b-1]^t$; thus, $\mathrm{val}_{2,m,t}$ is functionally equivalent to the $\mathrm{val}_{m,t}$ function introduced in the preceding paragraph. Going forward, we omit the subscripts m, t from the preceding notations, noting that m and t can always be inferred from context when needed.

Stinson [30] describes three algorithms to compute low-Hamming-weight DLs. Lemmas 1 and 2 generalize Lemmas 1.1 and 1.2 from Stinson's paper to the above-generalized family of radix-b val functions. Proofs of these simple lemmas are in our extended technical report [11; §§A.1 and A.2].

Lemma 1. *Fix a radix $b > 1$, let m be a positive integer, and let t be an even integer in $[m]$. If $g^{\mathrm{val}_b(X_1,Y_1)} = h \cdot \left(g^{\mathrm{val}_b(X_2,Y_2)}\right)^{-1}$ for $X_1, X_2 \in [b-1]^{t/2}$ and $Y_1, Y_2 \in \binom{[m]}{t/2}$, then $\log_g h \equiv (\mathrm{val}_b(X_1, Y_1) + \mathrm{val}_b(X_2, Y_2)) \bmod q$.*

Note that $h \cdot \left(g^{\mathrm{val}_b(X_2,Y_2)}\right)^{-1} = h \cdot \left(g^{-1}\right)^{\mathrm{val}_b(X_2,Y_2)}$. The algorithms we present in the next two sections use the right-hand side of this expression instead of the left-hand side, as doing so allows us to invert g once and for all, rather than inverting $g^{\mathrm{val}_b(X_2,Y_2)}$ once for each new choice of (X_2, Y_2).

Lemma 2. *Fix a radix $b > 1$, let m be an arbitrary positive integer, and let t be an even integer in $[m]$. If there is an $x \equiv \log_g h \bmod q$ with radix-b weight t and radix-b length at most m, then there exist two* disjoint *subsets $Y_1, Y_2 \in \binom{[m]}{t/2}$ and corresponding $X_1, X_2 \in [b-1]^{t/2}$ such that $g^{\mathrm{val}_b(X_1,Y_1)} = h \cdot \left(g^{\mathrm{val}_b(X_2,Y_2)}\right)^{-1}$.*

Lemmas 1 and 2 assume that t is even so that $t/2$ is an integer. We make this simplifying assumption purely for notational and expositional convenience; indeed, both lemmas still hold if, for example, we let $(X_1, Y_1) \in [b-1]^{\lfloor t/2 \rfloor} \times \binom{[m]}{\lfloor t/2 \rfloor}$

and $(X_2, Y_2) \in [b-1]^{\lceil t/2 \rceil} \times \binom{[m]}{\lceil t/2 \rceil}$. The algorithms that follow in Sections 3 and 4 make the same simplifying assumption (in fact, the algorithms in Section 3.3 also assume that m is even); however, we stress that each algorithm is likewise trivial to adapt for t and m with arbitrary parities (as discussed by Stinson [30; §5]).

3 Computing DLs with low Hamming weight

In this section, we describe and improve upon two variants of the celebrated "baby-step giant-step" algorithm [29] for computing DLs. These algorithm variants have been specially adapted for cases in which the exponent is known to have low Hamming weight. The most basic form of each algorithm is described and analyzed in a paper by Stinson [30], who credits the first to Heiman [8] and Odlyzko [25] and the second to Coppersmith (by way of unpublished correspondence with Vanstone [4]).[2] In both cases, our improvements yield modest-yet-notable performance improvements—both concretely and asymptotically—over the more basic forms of the algorithms; indeed, our improvements to the second algorithm yield a worst-case computation complexity superior to that of any known algorithm for the low-Hamming-weight DL problem. In Section 4, we propose and analyze a simple transformation that generalizes each low-Hamming-weight DL algorithm in this paper to a corresponding low-radix-b-weight DL algorithm, where the radix $b > 1$ can be arbitrary.

Algorithm 3.1 LowHammingDL$(m, t; g, h)$ // Attempts to compute $x = \log_g h \bmod q$ (assumes $\text{len}_2(x) <= m$, $\text{wt}_2(x) = t$, and t is even)

1: Initialize a hash table H
2: /* "Giant step": Populate lookup table */
3: **for each** $(Y_1 \in \binom{[m]}{t/2})$ **do** // loop runs (m choose t/2) times
4: $\quad y_1 \leftarrow g^{\text{val}(Y_1)}$
5: $\quad H.\text{put}(y_1, Y_1)$ // $y_1 = g^{\text{val}(Y_1)}$ is key; Y_1 is value
6: **end for**
7: /* "Baby step": Search for a collision */
8: **for each** $(Y_2 \in \binom{[m]}{t/2})$ **do** // loop runs <= (m choose t/2) times
9: $\quad y_2 \leftarrow h \cdot (g^{-1})^{\text{val}(Y_2)}$ // cf. Lemma 2
10: $\quad$ **if** $(H.\text{containsKey}(y_2))$ **then**
11: $\quad\quad Y_1 \leftarrow H.\text{get}(y_2)$
12: $\quad\quad$ **return** $(\text{val}(Y_1) + \text{val}(Y_2)) \bmod q$ // cf. Lemma 1
13: $\quad$ **end if**
14: **end for**
15: **return** $\perp$ // $\log_g h$ = undefined, $\text{len}_2(\log_g h) > m$, or $\text{wt}_2(\log_g h) \neq t$

[2] In addition to the basic algorithms described herein, Stinson's paper introduces a generalization of the second algorithm based on a combinatorial structure he coins *splitting systems* [30; §4], as well as a *randomized* variant (also credited to Coppersmith) [30; §2.2], both of which we describe in our extended technical report [11; §§C and D].

3.1 The basic algorithm

Algorithm 3.1 gives pseudocode for the most basic form of the algorithm, which is due to Heiman and Odlyzko [8, 25].

Theorem 3. *Algorithm 3.1 is* correct: *If there is an m-bit integer x with Hamming weight t such that $g^x = h$, then the algorithm returns a DL of h to the base g.*

Proof (sketch). This follows directly from Lemmas 1 and 2. Specifically, Lemma 1 ensures that any value returned on Line 12 of Algorithm 3.1 satisfies $g^x = h$, while Lemma 2 ensures that the baby-step loop (Lines 8–14) will indeed find the requisite pair (Y_1, Y_2) if such a pair exists. □

Remark. When the order q is unknown, one can set m to be any *upper bound* on $\lceil \lg q \rceil$, and then omit the modular reduction on Line 12 of Algorithm 3.1. Indeed, one may even set $m > \lceil \lg q \rceil$ when q is known if, for example, the canonical representation of the desired DL has large Hamming weight but is known to be congruent (modulo q) to an m-bit integer with low Hamming weight.

The next theorem follows easily by inspection of Algorithm 3.1.

Theorem 4. *The storage cost and (both average- and worst-case) computation cost of Algorithm 3.1, counted respectively in group elements and group exponentiations, each scale as $\Theta(\binom{m}{t/2})$.*

Remark. Each exponentiation counted in Algorithm 3.1 is to a power with Hamming weight $t/2$. By pre-computing $g^{\mathrm{val}(\{i\})}$ for $i \in [m]$, one can evaluate these exponentiations using just $t/2 - 1$ group operations a piece. The (both average- and worst-case) computation complexity becomes $\Theta(t\binom{m}{t/2})$ group operations. Going a step further, one can pre-compute $g^{\mathrm{val}(\{i\})-\mathrm{val}(\{j\})}$ for each $i \neq j$, and then iterate through $\binom{[m]}{t/2}$ following a "minimal-change ordering" [19; §2.3.3] wherein each successive pair of subsets differ by exactly two elements [31]. Then all but the first iteration of the baby-step (respectively, giant-step) loop uses a single group operation to "update" the y_1 (respectively, y_2) from the previous iteration. The worst-case computation cost becomes $2\binom{m}{t/2} + t - 3$ group operations (plus one inversion and m^2 group operations for pre-computation).

3.2 Improved complexity via interleaving

Next, we propose and analyze an alternative way to implement the basic algorithm (i.e., Algorithm 3.1), which interleaves the baby-step and giant-step calculations in a manner reminiscent of Pollard's interleaved variant of the classic baby-step giant-step algorithm [28; §3]. Although such interleaving is a well-known technique for achieving constant-factor average-case speedups in baby-step giant-step algorithms, it had not previously been applied in the context

of low-Hamming-weight DLs. Our analysis reveals that interleaving can, in fact, yield a surprisingly large (super-constant) speedup in this context.

The interleaved variant comprises a single loop and two lookup tables, H_1 and H_2. The loop iterates simultaneously over the subsets $Y_1 \in \binom{[m]}{t/2}$ and $Y_2 \in \binom{[m]}{t/2}$ in respectively increasing and decreasing order. (To keep the following analysis simple, we assume the order is lexicographic; however, we note that one can obtain a factor t speedup by utilizing some pre-computation and a minimal-change ordering, exactly as we suggested in the above remarks following the non-interleaved algorithm.) In each iteration, the algorithm computes both $y_1 \coloneqq g^{\mathrm{val}(Y_1)}$ and $y_2 \coloneqq h \cdot (g^{-1})^{\mathrm{val}(Y_2)}$, storing (y_1, Y_1) in H_1 and (y_2, Y_2) in H_2, and also checking if y_1 collides with a key in H_2 or y_2 with a key in H_1. Upon discovering a collision, it computes and outputs $x \equiv \log_g h \bmod q$ using Lemma 1 (cf. Line 12 of Algorithm 3.1) and then halts. A pseudocode description of our interleaved algorithm is included in our extended technical report [11; §B.1].)

Despite its simplicity, this modification appears to be novel and has a surprisingly large impact on the average-case complexity. Indeed, if we assume that the interleaved loop iterates through $\binom{[m]}{t/2}$ in increasing and decreasing lexicographic order (for the giant-step and baby-step calculations, respectively), then the worst possible costs arise when the t one bits in the binary representation of x occur consecutively in either the t highest-order or the t lowest-order bit positions (i.e., when $x = 1^t 0^{m-t}$ or $x = 0^{m-t} 1^t$). In this case, the algorithm produces a collision and halts after $\binom{m-t/2}{t/2}$ iterations of the loop. For $t \in \Theta(\sqrt{m})$, this gives a worst-case constant factor speedup compared to the non-interleaved algorithm;[3] for $t \in \omega(\sqrt{m})$, the worst-case speedup is asymptotic (alas, we are unable to derive a precise characterization of the speedup in terms of m and t). The average-case speedup can be much more dramatic, depending on the distribution of the targeted $x \equiv \log_g h \bmod q$. For a uniform distribution (among the set of all m-bit exponents with Hamming weight t) on x, we heuristically expect the one bits in x to be distributed evenly throughout its binary representation; that is, we expect to find the $(t/2)$th and $(t/2+1)$th one bits in x in or around bit positions $\frac{t}{2}\frac{m}{t+1} < \frac{m}{2}$ and $\frac{t+2}{2}\frac{m}{t+1} > \frac{m}{2}$, respectively. Therefore, we expect the interleaved algorithm to produce a collision and halt after at most around $\binom{m/2}{t/2}$ loop iterations. (Contrast this with the original average-case $\Theta(\binom{m}{t/2})$ complexity of the non-interleaved algorithm.) We summarize our above analysis in Theorem 5.

[3] More precisely, when $t = 2c\sqrt{m}$, we find that $\lim_{m\to\infty} \binom{m-t/2}{t/2} / \binom{m}{t/2} = e^{-c^2}$; that is, as m grows large, the worst-case computation cost of the interleaved algorithm approaches a factor e^{-c^2} that of the non-interleaved algorithm; moreover, this limiting factor is a *lower bound* that underestimates the true worst-case speedup for small values of m. As a case in point, $m = 256$ and $t = 64$ (so that $c = 2$) yields a speedup by a factor 97.2, which is about 78% better than the predicted speedup factor of e^4.

Theorem 5. *The worst-case storage and computation costs of the interleaved algorithm described above, counted respectively in group elements and group operations, each scale as* $\Theta(\binom{m-t/2}{t/2})$. *If x is uniform among m-bit exponents with Hamming weight t, then the average-case storage and computation complexities scale as* $\Theta(\binom{m/2}{t/2})$.

3.3 The Coppersmith algorithms

Algorithm 3.1 and our interleaved variant are "direct" algorithmic instantiations of Lemmas 1 and 2 with a fixed radix $b = 2$. Such direct instantiations perform poorly in the worst case because Lemma 2 guarantees only *existence*—but *not uniqueness*—of the subsets Y_1 and Y_2 and, as a result, the collections of subsets over which these direct instantiations ultimately iterate are only guaranteed to be *sufficient*—but *not necessary*—to compute the desired logarithm. Indeed, given $Y \in \binom{[m]}{t}$ such that $\log_g h \equiv \mathrm{val}(Y) \bmod q$, there exist $\binom{t}{t/2}$ distinct ways to partition Y into $Y_1 \in \binom{Y}{t/2}$ and $Y_2 = Y \setminus Y_1$ to satisfy the congruence $\log_g h \equiv (\mathrm{val}(Y_1) + \mathrm{val}(Y_2)) \bmod q$ arising in Lemma 2. Stirling's approximation implies that $\binom{t}{t/2}$ approaches $2^t/\sqrt{\pi t/2}$ as t grows large so that the number of "redundant" values these basic algorithms may end up computing (and storing) grows *exponentially* with t. We now describe a more efficient variant of this algorithm, originally proposed by Coppersmith [4], that improves on the complexity of the basic algorithms by taking special care to iterate over significantly fewer redundant subsets. (Actually, Coppersmith proposed two related algorithms—one *deterministic* and the other *randomized*; however, due to space constraints, we discuss only the deterministic algorithm in this section, relegating our discussion of the randomized algorithm to our extended technical report [11; §D].)

Coppersmith's deterministic algorithm. The first variant of Algorithm 3.1 proposed by Coppersmith is based on the following observation.

Observation 6 (Coppersmith and Seroussi [5]). *Let t and m be even positive integers with $t \leq m$ and, for each $i = 1, \ldots, m/2 + 1$, define $B_i = \{i, i+1, \ldots, i + m/2 - 1\}$ and $\bar{B}_i = [m] \setminus B_i$. For any $Y \in \binom{[m]}{t}$, there exists some $i \in [m/2]$ and (disjoint) subsets $Y_1 \in \binom{B_i}{t/2}$ and $Y_2 \in \binom{\bar{B}_i}{t/2}$ such that $Y = Y_1 \cup Y_2$.*

A proof of Observation 6 is included in our extended technical report [11; §A.4]. The following analog of Lemma 2 is an immediate corollary to Observation 6.

Corollary 7. *Let t and m be even positive integers with $t \leq m$ and, for each $i = 1, \ldots, m/2 + 1$, define $B_i = \{i, i+1, \ldots, i + m/2 - 1\}$ and $\bar{B}_i = [m] \setminus B_i$. If there is an $x \equiv \log_g h \bmod q$ with Hamming weight t and bit length at most m, then there exists some $i \in [m/2]$ and (disjoint) subsets $Y_1 \in \binom{B_i}{t/2}$ and $Y_2 \in \binom{\bar{B}_i}{t/2}$ such that $g^{\mathrm{val}(Y_1)} = h \cdot g^{-\mathrm{val}(Y_2)}$.*

Using Corollary 7 to improve on the worst-case complexity of the basic algorithm is straightforward. The giant-step and baby-step loops (i.e., Lines 3–6 and 8–14) from Algorithm 3.1 are respectively modified to iterate over only the subsets $Y_1 \in \binom{B_i}{t/2}$ and $Y_2 \in \binom{\bar{B}_i}{t/2}$ for each $i = 1, \ldots m/2$ in turn. In particular, the algorithm populates a lookup table H in the giant-step loop using only the $Y_1 \in \binom{B_1}{t/2}$, and then it searches for a collision within H in the baby-step loop using only the $Y_2 \in \binom{\bar{B}_1}{t/2}$; if the baby-step loop for $i = 1$ generates no collisions, then the algorithm clears the lookup table and repeats the process for $i = 2$, and so on up to $i = m/2$. Observation 6 guarantees that the algorithm finds a collision and halts at some point prior to completing the baby-step loop for $i = m/2$, provided a DL with the specified Hamming weight and bit length exists. Pseudocode for the above-described algorithm is included in our extended technical report [11; §B.2]

The next theorem follows easily from Corollary 7 and by inspection.

Theorem 8. *Coppersmith's deterministic algorithm is* correct*; moreover, its storage cost scales as* $\Theta(\binom{m/2}{t/2})$ *group elements and its (worst-case) computation cost as* $\mathrm{O}(m\binom{m/2}{t/2})$ *group exponentiations.*[4]

Remark. The average-case complexity requires a delicate analysis, owing to the fact that there may be several indices i for which $|Y \cap B_i| = |Y \cap \bar{B}_i| = t/2$ and the algorithm will always halt upon encountering the *first* such index. Interested readers can find a detailed analysis of the average-case complexity in Stinson's paper [30; §3]. Stinson's paper also proposes a generalization of Coppersmith's deterministic algorithm utilizing a family of combinatorial set systems called *splitting systems* [30; §2.1] (of which the Coppersmith–Seroussi set system defined in Observation 6 and Corollary 7 is an example). A discussion of splitting systems and Stinson's improvements to the above algorithm is included in our extended technical report [11; §C].

3.4 Improved complexity via Pascal's Lemma

A methodical analysis of the Coppersmith–Seroussi set system suggests an optimization to Coppersmith's deterministic algorithm that yields an asymptotically lower computation complexity than that indicated by Theorem 8. Indeed, the resulting optimized algorithm has a worst-case computation complexity of just $\Theta(t\binom{m/2}{t/2})$ group operation, which is asymptotically lower than that of any low-Hamming-weight DL algorithm in the literature. Moreover, the hidden constant in the optimized algorithm (i.e., $1 + \mathrm{o}(1)$) seems to be about as low as one could

[4] Stinson states [30; §2.1] that the storage cost is $\binom{m}{t/2}$ group elements; however, this is clearly not possible, as the computation cost is not large enough to even *produce*, let alone necessitate storing, so many group elements. Given that $\binom{m}{t/2}$ group elements is the correct storage cost for the basic algorithm, and that $\binom{m}{t/2}$ differs from $\binom{m/2}{t/2}$ in just two characters, we attribute the discrepancy to a simple copy-paste error or typo.

realistically hope for. Our improvements follow from Observation 9, an immediate consequence of Pascal's Lemma for binomial coefficients, which states that $\binom{m/2}{t/2} = \binom{m/2-1}{t/2-1} + \binom{m/2-1}{t/2}$.

Observation 9. *Let $\{B_1, \ldots, B_{m/2}\}$ be the Coppersmith–Seroussi set system, as defined in Observation 6 and Corollary 7. For each $i = 1, \ldots, m/2 - 1$, we have that $\left|\binom{B_i}{t/2} \cap \binom{B_{i+1}}{t/2}\right| = \binom{m/2-1}{t/2}$.*

A simple corollary to Observation 9 is that the baby-step and giant-step loops for $i = 2, \ldots, m/2$ in a naïve implementation of Coppersmith's deterministic algorithm each recompute $\binom{m/2-1}{t/2}$ values that were also computed *in the immediately preceding invocation*, or, equivalently, that these loops each produce just $\binom{m/2}{t/2} - \binom{m/2-1}{t/2} = \binom{m/2-1}{t/2-1}$ new values. Carefully avoiding these redundant computations can therefore reduce the per-iteration computation cost of all but the first iteration of the outer loop to $2\binom{m/2-1}{t/2-1}$ group operations. The first (i.e., $i = 1$) iteration of the outer loop must, of course, still produce $2\binom{m/2}{t/2}$ values; thus, in the worst case, the algorithm must produce $2\left(\binom{m/2}{t/2} + (\frac{m}{2} - 1)\binom{m/2-1}{t/2-1}\right)$ distinct group elements. Note that in order to avoid all redundant computations in subsequent iterations, it is necessary to provide both the giant-step and baby-step loops with access to the (y_1, Y_1) and (y_2, Y_2) pairs, respectively, that arose in the immediately preceding invocation. Coppersmith's deterministic algorithm already stores each (y_1, Y_1) pair arising in the giant-step loop, but it does not store the (y_2, Y_2) pairs arising in the baby-step loop; hence, fully exploiting Observation 9 doubles the storage cost of the algorithm (in a similar vein to interleaving the loops). The upshot of this increased storage cost is a notable asymptotic improvement to the worst-case computation cost, which we characterize in Lemma 10 and Corollary 11. A proof of Lemma 10 is located in Appendix A.1.

Lemma 10. *Let $\{B_1, \ldots, B_{m/2}\}$ be the Coppersmith–Seroussi set system, as defined in Observation 6 and Corollary 7. We have*

$$\frac{\left|\bigcup_{i=1}^{m/2} B_i\right|}{\sum_{i=1}^{m/2} |B_i|} = \frac{t}{m} + \mathrm{o}(1).$$

To realize the speedup promised by Lemma 10, the optimized algorithm must do some additional bookkeeping; specifically, in each iteration $i = 2, \ldots, m/2$, it must have an efficient way to determine which of the $Y_1 \in \binom{B_i}{t/2}$ and $Y_2 \in \binom{\bar{B}_i}{t/2}$—as well as the associated $y_1 = g^{\mathrm{val}(Y_1)}$ and $y_2 = h \cdot g^{-\mathrm{val}(Y_2)}$—arose in the $(i-1)$th iteration, and which of them arise will for the first time in the ith iteration. To this end, the algorithm keeps two *sequences* of hash tables, say $H_1, \ldots, H_m$ and $I_1, \ldots, I_m$, one for the giant-step pairs and another for the baby-step pairs. Into which hash table a given (Y_1, y_1) pair gets stored is determined by the *smallest* integer in Y_1: a (Y_1, y_1) pair that arose in the $(i-1)$th iteration of the outer loop

will also arise in the ith iteration if and only if the smallest element in Y_1 is not $i-1$; thus, all values from the $(i-1)$th iteration not in the hash table H_{i-1} can be reused in the next iteration. Moreover, each (Y_1, y_1) pair that will arise for the first time in the ith iteration has a corresponding (Y_1', y_1') pair that is guaranteed to reside in H_{i-1} at the end of the $(i-1)$th iteration. Indeed, one can efficiently "update" each such (Y_1', y_1') in H_{i-1} to a required (Y_1, y_1) pair by setting $Y_1 = (Y_1' \setminus \{i-1\}) \cup \{i+m/2\}$ and $y_1 = y_1' \cdot g^{-(i-1)} \cdot g^{i+m/2}$. Note that because Y_1 no longer contains $i-1$, the hash table in which the updated (Y_1, y_1) pair should be stored changes from H_{i-1} to H_j for some $j \geq i$. An analogous method is used for keeping track of and "updating" the (Y_2, y_2) pairs arising in the baby-step loop. Pseudocode for the above-described algorithm is included as Algorithm B.1 in Appendix B. The following corollary is an immediate consequence of Lemma 10.

Corollary 11. *Algorithm B.1 is* correct*; moreover, its storage cost scales as* $\Theta(\binom{m/2}{t/2})$ *group elements and its worst-case computation cost as* $\mathrm{O}(t\binom{m/2}{t/2})$ *group exponentiations.*

Note that the worst-case complexity obtained in Corollary 11 improves on a naïve implementation of Coppersmith's algorithm by a factor $\frac{m}{t}$ (and it improves on the previously best-known lower bound, due to Stinson [30; Theorem 4.1], by a factor $\sqrt{t}\lg m$). As with the basic algorithm, one can leverage pre-computation and a minimal-change ordering to replace all but two of the exponentiations counted by Corollary 11 with a single group operation each; hence, the worst-case computation complexity is in fact just $\Theta(t\binom{m/2}{t/2})$ group operations.

4 From low Hamming weight to low radix-b weight

In this section, we introduce and analyze a simple transformation that allows us to generalize each of the low-Hamming-weight DL algorithms from the preceding section to a low-radix-b-weight DL algorithm, where the radix $b > 1$ can be arbitrary. The transformation is deceptively simple: essentially, it entails modifying the low-Hamming-weight algorithm to iterate over all possible inputs to a val_b function, rather than over all possible inputs to an "unqualified" val function (or, equivalently, to a val_2 function). Algorithm 4.1 provides pseudocode for the simplest possible form of our radix-b algorithm; that is, for the transformation applied to Algorithm 3.1. We illustrate the transformation as it applies to this most basic form of the low-Hamming-weight DL algorithm purely for ease of exposition; indeed, we *do not* recommend implementing this particular variant in practice—rather, we recommend applying the transformation to Algorithm B.1 or to the randomized algorithm (see our extended technical report [11; §D]) as outlined below.

Algorithm 4.1 LowRadixDL$(m, t, b; g, h)$ // Attempts to compute $x = \log_g h \bmod q$ (assumes $\text{len}_b(x) <= m$, $\text{wt}_b(x) = t$, and t is even)

1: Initialize a hash table H
2: /* "Giant step": Populate lookup table */
3: **for each** $(Y_1 \in \binom{[m]}{t/2})$ **do** // outer loop runs (m choose t/2) times
4: **for each** $(X_1 \in [b-1]^{t/2})$ **do** // inner loop runs $(b-1)^{t/2}$ times
5: $y_1 \leftarrow g^{\text{val}_b(X_1, Y_1)}$
6: $H.\text{put}(y_1, (X_1, Y_1))$ // $y_1 = g^{\text{val}_b(X_1, Y_1)}$ is key; (X_1, Y_1) is value
7: **end for**
8: **end for**
9: /* "Baby step": Search for a collision */
10: **for each** $(Y_2 \in \binom{[m]}{t/2})$ **do** // outer loop runs <= (m choose t/2) times
11: **for each** $(X_2 \in [b-1]^{t/2})$ **do** // inner loop runs <= $(b-1)^{t/2}$ times
12: $y_2 \leftarrow h \cdot (g^{-1})^{\text{val}_b(X_2, Y_2)}$ // cf. Lemma 2
13: **if** $(H.\text{containsKey}(y_2))$ **then**
14: $(X_1, Y_1) \leftarrow H.\text{get}(y_2)$
15: $x \leftarrow (\text{val}_b(X_1, Y_1) + \text{val}_b(X_2, Y_2)) \bmod q$
16: **return** x // cf. Lemma 1
17: **end if**
18: **end for**
19: **end for**
20: **return** $\perp$ // $\log_g h$ = undefined, $\text{len}_b(\log_g h) > m$, or $\text{wt}_b(\log_g h) \neq t$

Theorem 12. *Algorithm 4.1 is* correct: *If there exists an integer x with radix-b length m and radix-b weight t such that $g^x = h$, then the algorithm returns a DL of h to the base g.*

Remark. When the radix is $b = 2$, the inner giant-step and baby-step loops (i.e., Lines 4–7 and 11–18) execute only once and Algorithm 4.1 reduces to Algorithm 3.1, an observation which bares out in the following theorem. If the radix is $b > 2$ yet all digits are bounded above by some $c < b$, then the inner loops need only iterate over the $(c-1)^{t/2}$ tuples in $[c-1]^{t/2}$, thus reducing the cost by a factor $\left(\frac{c-1}{b-1}\right)^{t/2}$.

Theorem 13. *The storage cost and (both average- and worst-case) computation cost of the above algorithm, counted respectively in group elements and group exponentiations, each scale as $\Theta\left((b-1)^{t/2}\binom{m}{t/2}\right)$.*

Remark 14. As with the low-Hamming-weight algorithms, it is possible to reduce each of the exponentiations counted by Theorem 13 to a single group operation, in this case by using a minimal-change ordering for the outer loop and a Gray code [19; §2.2.2] for the inner loop.

More efficient radix-b variants. Every one of the algorithm variants we described in Section 3 generalizes similarly to an algorithm for radix b, by simply

including an inner loop over each $X \in [b-1]^{t/2}$ within the giant-step and baby-step loops. In each case, the expressions for storage and worst-case computation complexity pick up an additional factor $(b-1)^{t/2}$; however, the reader should bear in mind that this newfound exponential factor is at least partially offset by a corresponding decrease in the radix-b length and (presumably) weight that appear in the binomial term. In particular, an exponent $x \equiv \log_g h \bmod q$ with bit length m_2 has a radix-b length of $m_b \approx m_2/\log_2 b$. Specifically, applying the transformation to Algorithm B.1 yields a radix-b algorithm with worst-case running time of $(t + o(1))\binom{m/2}{t/2}(b-1)^{t/2}$, where m and t respectively denote the radix-b length and radix-b weight of the DL sought.

In Theorem 15, we (partially) characterize one condition under which it is beneficial to switch from a baby-step giant-step algorithm for radix b to the corresponding baby-step giant-step algorithm for some larger radix. In this theorem, the *radix-b density* of x refers to the ratio of its radix-b weight to its radix-b length. For example, if m and t respectively denote the radix-b length of x and the radix-b weight of x, then its radix-b density is $t/m \in [0, 1]$.

Theorem 15. *Fix a radix $b > 1$ and an exponent x with radix-b density d. There exists a constant $k_0 \in \mathbb{R}$ (with $k_0 > 1$) such that, for all $k > k_0$, if the radix-b^k density of x is less than or equal to d, then a radix-b^k algorithm has lower cost than the corresponding radix-b algorithm.*

Theorem 15 implies that, for a fixed algorithm variant, switching to using a higher radix is beneficial (essentially) whenever the change to the radix does not increase the density of the DL being computed. We emphasize that the exponent k in the theorem need not be an integer;[5] thus, the theorem addresses cases like that of switching from a radix-2 representation to a radix-3 representation ($k = \lg 3$) or from a radix-4 representation to a radix-8 representation ($k = 3/2$). For example, the (decimal) number 20871 has radix-4 representation 11012013 and density 0.75, whereas it has radix-8 representation 50607 and density 0.6.

A proof sketch for Theorem 15 is included in Appendix A.2. We only sketch the main idea behind the proof, and only for the "basic" radix-b algorithm (i.e., for Algorithm 4.1). The reason for this is twofold: first, the details of the relevant calculations are both unenlightening and rather messy (owing to the fact that $(b-1)^{t/2} < (b^k-1)^{t/2k}$, which can make our relevant inequalities go the wrong way for small values of k); and, second, nearly identical calculations illustrate why the theorem holds for the more efficient algorithm variants.

[5] Indeed, if $k > 1$ is an integer, then the radix-b^k density of x cannot be lower—and is usually higher—than the radix-b density of x.

5 Related work

The problem of solving "constrained" variants of the DL problem has received considerable attention in the cryptographic literature, with the most well-known and widely studied such variant being that of computing DLs that are "small" [33] or known to reside in a "short" interval [23, 32, 34]. Existing algorithms can compute such DLs using an expected $O(\sqrt{B - A})$ group operations when the exponent is known to reside in the interval $[A\,..\,B]$.

In addition to the basic Heiman–Odlyzko [8] and Coppersmith–Stinson [30] low-Hamming-weight algorithms discussed earlier, a handful of papers have considered the problem of computing DLs that have low Hamming weight [20, 24] or those expressible as a product whose multiplicands each have low Hamming weight [3, 16, 17]. Efficient algorithms for these computations have applications in attacking encryption schemes that leverage such low-weight [1, 18, 22] and product-of-low-weight-multiplicand [7, 9] exponents as a means to reduce the cost of public-key operations. They have also been adapted to attack so-called *secure human identification protocols* [10], which leverage low-Hamming-weight secrets to improve memorability for unassisted humans.

Specifically, Cheon and Kim [3] proposed a baby-step giant-step algorithm to compute DLs expressible as a product of three low-Hamming-weight multiplicands in groups with known order. The use of such "product-of-three" exponents was proposed by Hoffstein and Silverman [9] to allow for low-cost exponentiations in groups that permit fast endomorphism squaring (which includes the Galois fields $\mathbf{GF}(2^n)$ and the so-called "Koblitz" elliptic curves) while seeking to resist meet-in-the-middle attacks. Subsequently, Kim and Cheon [16, 17][6] improved on those results using a "parametrized" variant of splitting systems, while Coron, Lefranc, and Poupard [6] proposed a related algorithm that works in groups with *unknown composite order* (e.g., in the multiplicative group of units modulo $n = pq$, where p and q are large primes and the factorization of n into p and q is not provided to the algorithm).

Meanwhile, Muir and Stinson [24] studied generalizations of Coppersmith's deterministic algorithm to compute DLs known to have a non-adjacent form (NAF) representation with low weight. (In the latter context, "low weight" means a small numbers of ± 1 digits in the NAF representation.) More recently, May and Ozerov [20] revisited the low-Hamming-weight DL problem in groups of composite order (where a factorization of the order is known), proposing an algorithm that combines aspects of the Silver–Pohlig–Hellman [27] algorithm with any of the basic low Hamming weight algorithms to obtain lower complexity than either approach in isolation.

[6] Incidentally, the Cheon who authored [3] and [16, 17] is one and the same, but the Kim who authored [3] is not the Kim who authored [16, 17].

The algorithms we have presented in this work (i) offer improved complexity relative to existing low-Hamming-weight algorithms, and (ii) generalized to the low-radix-b-weight case for arbitrary $b \geq 2$. This is a (mathematically) natural generalization of the low-Hamming-weight DL problem that has not been explicitly considered in prior work. We suspect that our modifications will "play nice" with some or all of the above-mentioned low-weight DL algorithm variants, and we slate a formal investigation of this interplay for future work.

6 Cryptanalytic applications

We now turn our attention to the cryptanalytic applications of our new algorithms. Specifically, we demonstrate how to use a low-radix-b-weight DL algorithm to attack any one of several *verifier-based password-authenticated key exchange* (VPAKE) protocols from the cryptographic literature. Briefly, a *password-authenticated key exchange* (PAKE) protocol is an interactive protocol enabling a client to simultaneously authenticate itself to, and establish a shared cryptographic key with, a remote server by demonstrating knowledge of a password. The security definitions for PAKE require that the interaction between the client and server reveals at-most a negligible quantity of information about the client's password (and the shared key): a man-in-the-middle who observes (and possibly interferes with) any polynomial number of PAKE sessions between a given client and server should gain at most a negligible advantage in either hijacking an authenticating session or impersonating the client (e.g., by guessing her password). VPAKE protocols extend PAKE with additional protections against the server, ensuring that an attacker who compromises the server cannot leverage its privileged position to infer the client's password using less work than would be required to launch a brute-force attack against the password database (even after engaging in any polynomial number of PAKE sessions with the client).

In recent work [12], Kiefer and Manulis proposed a VPAKE protocol with the novel property of allowing the client to register its password *without ever revealing that password to the server.* Their idea, at a high level, is to have the client compute a "fingerprint" of the password and then prove in zero-knowledge that the fingerprint was computed correctly; subsequent authentications involve a proof of knowledge of the password encoded in a given fingerprint. To make the zero-knowledge proofs practical, the password fingerprints are computed using a structure-preserving map. Benhamouda and Pointcheval [2; §1.2] note that the Kiefer–Manulis VPAKE construction, as originally presented, falls easily to a short-interval variant Pollard's Kangaroo algorithm [23]. In response to this observation, Kiefer and Manulis released an updated version of their paper (as a technical report [13]) that attempts to thwart the sort of short-interval

attacks pointed out by Benhamouda and Pointcheval. A handful of subsequent papers [14, 15, 35] have built on their algorithm, sharing the same basic framework (and, hence, similarly susceptible to the attack described below).

The Kiefer–Manulis Protocol

Before presenting our attack, we briefly summarize the relevant portions of Kiefer and Manulis' VPAKE construction. Passwords in their construction consist of any number of printable ASCII characters (of which there are 94 distinct possibilities that are each assigned a label in $[0\,..\,93]$) up to some maximum length, which we will denote by m; thus, there is a natural mapping between valid passwords and the set of radix-94 integers with length at most m. This yields $\sum_{i=1}^{m} 94^i$ possible passwords (although the authors incorrectly give the number as just 94^m).

The client maps her password pw to $\mathbb{Z}$ via the structure-preserving map

$$\texttt{PWDtoINT}(b; pw) \coloneqq \sum_{i=1}^{|pw|} b^{i-1} pw_i,$$

where $pw_i \in [0\,..\,93]$ is the numeric label assigned to the ith character in pw. Here $b \geq 94$ is an integer parameter, which the authors refer to as the "*shift base*".

The client computes a fingerprint of her password pw by selecting two random values, $\tilde{g} \in_R \mathbb{G}$ and $s \in_R \mathbb{Z}_q^*$ (the so-called "pre-hash" and "post-hash" salts) and using them to produce a Pedersen-like commitment[7]

$$C \coloneqq \tilde{g}^{\texttt{PWDtoINT}(b;\, pw)} h^s$$

and then outputting the tuple $(s, \tilde{g}, C)$. As the post-hash salt s in this construction is output as part of the fingerprint, it does not serve a clear purpose; indeed, any party (including the attacker) can trivially compute $\tilde{g}^{\texttt{PWDtoINT}(b;\, pw)} = C \cdot h^{-s}$. Thus, recovering the client's password $\texttt{PWDtoINT}(b; pw)$ (at least, modulo q) from a fingerprint is equivalent to solving for $x \equiv \log_{\tilde{g}} C \cdot h^{-s} \bmod q$.

The Benhamouda–Pointcheval attack. The original protocol used $b = 94$, which yields $\texttt{PWDtoINT}(b; pw) \leq 94^m - 1$; hence, as noted by Benhamouda and Pointcheval [2; §1.2], an attacker can recover $\texttt{PWDtoINT}(b; pw) \bmod q$ from $(s, \tilde{g}, C)$ in around $\sqrt{94^m} \lesssim 10^m$ steps using the Kangaroo algorithm. (Note that m here is a password length, and not a cryptographic security parameter.) This is a mere square root of the time required to launch a brute-force attack, which falls far short of satisfying the no-better-than-brute-force requirement for a VPAKE protocol.

---

[7] A *Pedersen commitment* [26] to x is a value $C = g^x h^r$ where r is uniform random and $\log_g h$ is secret; it is *perfectly hiding* because for every possible x there exists a unique r that would make the resulting commitment look like C and it is *computationally binding* because finding two distinct (x, r), (x', r') pairs that yield the same commitment is equivalent to computing $\log_g h$.

Kiefer and Manulis' defense. To protect against Kangaroo attacks, Kiefer and Manulis suggested to increase the shift base. Specifically, as exponents $\texttt{PWDtoINT}(b; pw)$ in their scheme have the form $\sum_{i=1}^{|pw|} b^{i-1} pw_i$ with each $pw_i \in [0\,..\,93]$, they solve for the smallest choice of b that causes the "largest" possible password of length $|pw|$ to induce an exponent that satisfies the inequality $94^{2m} < 93 \sum_{i=1}^{m} b^{i-1}$. Doing so means that exponents are distributed throughout the range $[0\,..\,94^{2m}]$, which is (ignoring constants) necessary and sufficient to ensure that a straightforward application of Pollard's Kangaroo algorithm will fail to solve the DL in fewer steps than are required to brute-force the password, on average. If one supposes that the Kangaroo algorithm is the best possible DL-based attack possible, the defense seems reasonable. Kiefer and Manulis suggest $b = 10^5$, which they state "should be a safe choice".

Revised attack from the deterministic low-radix-10^5-weight DL algorithm. Using our optimized form of Coppersmith's algorithm (together with the remarks following Theorem 12), one can solve for any password up to, say $m = 12$ characters long, using fewer than

$$\sum_{t=0}^{12} t \binom{m/2}{t/2} 93^{t/2} \approx 2^{38.2}$$

group operations, as compared with

$$\sum_{m=0}^{12} 94^m \approx 2^{78.7}$$

guesses for a brute-force attack, thus rendering Kiefer and Manulis' defense completely ineffective.

7 Conclusion

The DL problem is a cornerstone of modern cryptography. Several prior works have studied "constrained" variants of the DL problem in which the desired exponent is known either to have low Hamming weight or to be expressible as a product whose multiplicands each have low Hamming weight. In this work, we have focused on the related problem of computing DLs that have low radix-b weight for arbitrary $b \geq 2$. This is a (mathematically) natural generalization of the low-Hamming-weight DL problem that has not been explicitly considered in prior work. We emphasize that a significant part of our contribution was to minimize the hidden constants in the low-Hamming-weight algorithms (improving the best-known complexity for the radix-2 case) and, by extension, in their radix-b generalizations. We expect that our modifications will "play nice" with prior efforts to solve other low-Hamming-weight and product-of-low-weight-multiplicand DL problem variants, and we slate a formal investigation of this interplay for future work. To showcase the cryptanalytic applications of our new algorithms, we demonstrated an attack against several Verifier-Based Password Authenticated Key Exchange (VPAKE) protocols from the cryptographic literature.

Acknowledgements. We thank Doug Stinson, Itai Dinur, and the anonymous referees for their valuable feedback. Sarah Plosker is supported by the Natural Sciences and Engineering Research Council of Canada, the Canada Foundation for Innovation, and the Canada Research Chairs Program. Ryan Henry is supported by the National Science Foundation under Grant No. 1565375.

References

1. Gordon B. Agnew, Ronald C. Mullin, M. Onyszchuk I. and Scott A. Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3(2):63–79 (January 1991).
2. Fabrice Benhamouda and David Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. *IACR Cryptology ePrint Archive*, Report 2013/833 (October 2013).
3. Jung Hee Cheon and HongTae Kim. Analysis of low Hamming weight products. *Discrete Applied Mathematics*, 156(12):2264–2269 (June 2008).
4. Don Coppersmith. Personal communication to Scott Vanstone (July 1997). See [21; Page 128 in Chapter 3].
5. Don Coppersmith and Gadiel Seroussi. On the minimum distance of some quadratic residue codes. *IEEE Transactions on Information Theory*, 30(2):407–411 (March 1984).
6. Jean-Sébastien Coron, David Lefranc, and Guillaume Poupard. A new baby-step giant-step algorithm and some applications to cryptanalysis. In *Proceedings of CHES 2005*, volume 3659 of *LNCS*, pages 47–60, Edinburgh, UK (August 2005).
7. Marc Girault and David Lefranc. Public key authentication with one (online) single addition. In *Proceedings of CHES 2004*, volume 3156 of *LNCS*, pages 413–427, Cambridge, MA, USA (August 2004).
8. Rafi Heiman. A note on discrete logorithms with special structure. In *Advances in Cryptology: Proceedings of EUROCRYPT 1992*, volume 658 of *LNCS*, pages 454–457, Balatonfüred, Hungary (May 1992).
9. Jeffrey Hoffstein and Joseph H. Silverman. Random small Hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49 (August 2003).
10. Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In *Advances in Cryptology: Proceedings of ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 52–66, Gold Coast, Australia (December 2001).
11. Bailey Kacsmar, Sarah Plosker, and Ryan Henry. Computing low-weight discrete logarithms. *IACR Cryptology ePrint Archive*, Report 2017/720 (July 2017).
12. Franziskus Kiefer and Mark Manulis. Zero-knowledge password policy checks and verifier-based PAKE. In *Proceedings of ESORICS 2014 (Part II)*, volume 8713 of *LNCS*, pages 295–312, Wrocław, Poland (September 2014).
13. Franziskus Kiefer and Mark Manulis. Zero-knowledge password policy checks and verifier-based PAKE. *IACR Cryptology ePrint Archive*, Report 2014/242 (April 2014).
14. Franziskus Kiefer and Mark Manulis. Blind password registration for two-server password authenticated key exchange and secret sharing protocols. In *Proceedings of ISC 2016*, volume 9866 of *LNCS*, pages 95–114, Honolulu, HI, USA (September 2016).
15. Franziskus Kiefer and Mark Manulis. Blind password registration for verifier-based PAKE. In *Proceedings of AsiaPKC@AsiaCCS 2016*, pages 39–48, Xi'an, China (May 2016).
16. Sungwook Kim and Jung Hee Cheon. A parameterized splitting system and its application to the discrete logarithm problem with low Hamming weight product exponents. In *Proceedings of PKC 2008*, volume 4939 of *LNCS*, pages 328–343, Barcelona, Spain (March 2008).
17. Sungwook Kim and Jung Hee Cheon. Parameterized splitting systems for the discrete logarithm. *IEEE Transactions on Information Theory*, 56(5):2528–2535 (May 2010).

18. Neal Koblitz. CM-curves with good cryptographic properties. In *Advances in Cryptology: Proceedings of CRYPTO 1991*, volume 576 of *LNCS*, pages 279–287, Santa Barbara, CA, USA (August 1991).
19. Donald L. Kreher and Douglas R. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, New York, NY, USA (December 1998).
20. Alexander May and Ilya Ozerov. A generic algorithm for small weight discrete logarithms in composite groups. In *Proceedings of SAC 2014*, volume 8781 of *LNCS*, pages 278–289, Montreal, QC, Canada (August 2014).
21. Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, New York, NY, USA (October 1996). [Fifth printing; August 2001].
22. Alfred Menezes and Scott A. Vanstone. The implementation of elliptic curve cryptosystems. In *Advances in Cryptology: Proceedings of AUSCRYPT 1990*, volume 453 of *LNCS*, pages 2–13, Sydney, Australia (January 1990).
23. Ravi Montenegro and Prasad Tetali. How long does it take to catch a wild kangaroo? In *Proceedings of STOC 2009*, pages 553–560, Bethesda, MD, USA (May–June 2009).
24. James A. Muir and Douglas R. Stinson. On the low Hamming weight discrete logarithm problem for nonadjacent representations. *Applicable Algebra in Engineering, Communication, and Computing (AAECC)*, 16(6):461–472 (January 2006).
25. Andrew Odlyzko. Personal communication to Rafi Heiman (July 1992). See [8; Page 1 and Reference [Odl]].
26. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology: Proceedings of CRYPTO 1991*, volume 576 of *LNCS*, pages 129–140, Santa Barbara, CA, USA (August 1991).
27. Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110 (January 1978).
28. John M. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13(4):437–447 (September 2000).
29. Daniel Shanks. Class number, a theory of factorization, and genera. In *Proceedings of Symposium of Pure Mathematics*, volume 20, pages 415–440, Providence, RI, USA (July–August 1969).
30. Douglas R. Stinson. Some baby-step giant-step algorithms for the low Hamming weight discrete logarithm problem. *Mathematics of Computation*, 71(237):379–391 (January 2002).
31. Edlyn Teske. Square-root algorithms for the discrete logarithm problem (a survey). In *Proceedings of the International Conference on Public Key Cryptography and Computational Number Theory*, De Gruyter Proceedings in Mathematics, pages 283–301, Warsaw, Poland (September 2000).
32. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *Proceedings of CCS 1994*, pages 210–218, Fairfax, VA, USA (November 1994).
33. Paul C. van Oorschot and Michael J. Wiener. On Diffie-Hellman key agreement with short exponents. In *Advances in Cryptology: Proceedings of EUROCRYPT 1996*, volume 1070 of *LNCS*, pages 332–343, Saragossa, Spain (May 1996).
34. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28 (January 1999).
35. Xiaoyan Yang, Han Jiang, Qiuliang Xu, Mengbo Hou, Xiaochao Wei, Minghao Zhao, and Kim-Kwang Raymond Choo. A provably-secure and efficient verifier-based anonymous password-authenticated key exchange protocol. In *Proceedings of TrustCom/BigDataSE/ISPA 2016*, pages 670–677, Tianjin, China (August 2016).

A Proofs of basic results

This appendix presents proofs (and proof sketches) for some basic results that appear in the main body of the paper.

A.1 Proof of Lemma 10

This subappendix presents a proof for Lemma 10, which was stated in Section 3.4.

Lemma 10 (Restatement). *Let $\{B_1, \ldots, B_{m/2}\}$ be the Coppersmith–Seroussi set system, as defined in Observation 6 and Corollary 7. We have*

$$\frac{\left|\bigcup_{i=1}^{m/2} B_i\right|}{\sum_{i=1}^{m/2}|B_i|} = \frac{t}{m} + \mathrm{o}(1).$$

Proof. From Observation 9, we are interested in the ratio

$$\left(\tbinom{m/2}{t/2} + (\tfrac{m}{2} - 1)\tbinom{m/2-1}{t/2-1}\right) / \left(\tfrac{m}{2}\tbinom{m/2}{t/2}\right).$$

Using Pascal's Lemma to rewrite the numerator, this expression becomes

$$\left(\tbinom{m/2-1}{t/2} + \tfrac{m}{2}\tbinom{m/2-1}{t/2-1}\right) / \left(\tfrac{m}{2}\tbinom{m/2}{t/2}\right).$$

Simplifying, the first term in this expression becomes

$$\tbinom{m/2-1}{t/2} / \left(\tfrac{m}{2}\tbinom{m/2}{t/2}\right) = \frac{2(m-t)}{m^2} \in \mathrm{o}(1),$$

while the second term becomes

$$\left(\tfrac{m}{2}\tbinom{m/2-1}{t/2-1}\right) / \left(\tfrac{m}{2}\tbinom{m/2}{t/2}\right) = \frac{t}{m}.$$

Hence, it follows that $\left(\binom{m/2-1}{t/2} + \frac{m}{2}\binom{m/2-1}{t/2-1}\right) / \left(\frac{m}{2}\binom{m/2}{t/2}\right) = \frac{t}{m} + \mathrm{o}(1)$, as desired. □

A.2 Proof (sketch) of Theorem 15

This subappendix presents a proof sketch for Theorem 15, which was stated in Section 4.

Theorem 15 (Restatement). *Fix a radix $b > 1$ and an exponent x with radix-b density d. There exists a constant $k_0 \in \mathbb{R}$ (with $k_0 > 1$) such that, for all $k > k_0$, if the radix-b^k density of x is less than or equal to d, then a radix-b^k algorithm has lower cost than the corresponding radix-b algorithm.*

Proof of Theorem 15 (sketch). Let $m = \lceil \log_b x \rceil$ and t respectively denote the radix-b length and radix-b weight of x. Then the radix-b^k length of x is $\lceil \log_{b^k} m \rceil \approx \lceil m/k \rceil$ and we are interested in cases where the radix-b^k weight of x is (approximately) less than or equal to $\lceil t/k \rceil$. In such cases, the cost of the basic radix-b^k algorithm is about

$$2\tbinom{\lceil m/k \rceil}{\lceil t/2k \rceil}(b^k - 1)^{\lceil t/2k \rceil} \approx 2\tbinom{\lceil m/k \rceil}{\lceil t/2k \rceil} b^{t/2},$$

and this approximation tightens as k grows large. The right-hand side of the expression is bounded below by $2b^{t/2}$, with equality holding if and only if $k \geq m/2$. By contrast, the cost of the radix-b algorithm is about $2\binom{m}{t/2}(b-1)^{t/2} \approx 2\binom{m}{t/2} b^{t/2}$, which is strictly larger than $2b^{t/2}$. □

B Pseudocode

This appendix provides pseudocode for our optimized variant of Coppersmith's deterministic algorithm, as described in Section 3.4.

Algorithm B.1 $\text{LowHammingDL}_{\text{Pascal}}(m, t; g, h)$ // Attempts to compute $x = \log_g h \bmod q$ (assumes $\text{len}_2(x) <= m$, $\text{wt}_2(x) = t$, and m, t are even)

1: Initialize hash tables $H_1, I_1, \ldots, H_m, I_m$ // two tables per exponent bit
2: /* Initial "giant step" */
3: **for each** $(Y_1 \in \binom{\lfloor m/2 \rfloor}{t/2})$ **do** // runs (m/2 choose t/2) times
4: $y_1 \leftarrow g^{\text{val}(Y_1)}$
5: $j \leftarrow Y_1[1]$ // j is smallest integer in Y_1
6: $H_j.\text{put}(y_1, Y_1)$ // $y_1 = g^{\text{val}(Y_1)}$ is key; Y_1 is value
7: **end for**
8: /* Initial "baby step" */
9: **for each** $(Y_2 \in \binom{[m] \setminus \lfloor m/2 \rfloor}{t/2})$ **do** // runs <= (m/2 choose t/2) times
10: $y_2 \leftarrow h \cdot g^{-\text{val}(Y_2)}$ // cf. Corollary 7
11: **for** $(i = 1$ **to** $m/2)$ **do** // search for collision in each H_i
12: **if** $(H_i.\text{contains}(y_2))$ **then**
13: $Y_1 \leftarrow H_i.\text{get}(y_2)$
14: **return** $(\text{val}(Y_1) + \text{val}(Y_2)) \bmod q$ // cf. Lemma 1
15: **end if**
16: **end for**
17: $j \leftarrow Y_2[1]$ // j is smallest integer in Y_2
18: $I_j.\text{put}(y_2, Y_2)$ // $y_2 = g^{\text{val}(Y_2)}$ is key; Y_2 is value
19: **end for**
20: /* Interleaved "Pascal steps" */
21: **for** $(i = 1$ **to** $m/2 - 2)$ **do** // runs <= (m/2 − 1) times
22: **for each** $((y_1, Y_1) \in H_i)$ **do** // runs <= (m/2 − 1 choose t/2 − 1) times
23: $Y_1' \leftarrow (Y_1 \setminus \{i\}) \cup \{m/2 + i\}$ // "update" Y_1
24: $y_1' \leftarrow y_1 \cdot g^{-2^i} \cdot g^{2^{m/2+i}}$ // "update" y_1; $y_1' == g^{\text{val}(Y_1')}$
25: **for** $(j = m/2 + i$ **to** $m + i)$ **do** // search for collision in each I_j
26: $j' \leftarrow j \pmod m$
27: **if** $(I_{j'}.\text{contains}(y_1'))$ **then**
28: $Y_2 \leftarrow I_{j'}.\text{get}(y_1')$
29: **return** $(\text{val}(Y_1') + \text{val}(Y_2)) \bmod q$ // cf. Lemma 1
30: **end if**
31: **end for**
32: $j \leftarrow Y_1'[1]$ // j is smallest integer in Y_1'
33: $H_j.\text{put}(y_1', Y_1')$ // $y_1' = g^{\text{val}(Y_1')}$ is key; Y_1' is value
34: **end for**
35: $H_i.\text{clear}()$
36: **for each** $((y_2, Y_2) \in I_{m/2+i})$ **do** // runs <= (m/2 − 1 choose t/2 − 1) times
37: $Y_2' \leftarrow (Y_2 \setminus \{m/2 + i\}) \cup \{i\}$ // "update" Y_2
38: $y_2' \leftarrow y_2 \cdot g^{-2^i} \cdot g^{2^{m/2+i}}$ // "update" y_2; $y_2' == h * g^{-\text{val}(Y_2')}$
39: **for** $(j = i$ **to** $m/2 + i)$ **do** // search for collision in each H_j
40: **if** $(H_j.\text{contains}(y_2'))$ **then**
41: $Y_1 \leftarrow H_j.\text{get}(y_2')$
42: **return** $(\text{val}(Y_1) + \text{val}(Y_2')) \bmod q$ // cf. Lemma 1
43: **end if**
44: **end for**
45: $j \leftarrow Y_2'[1]$ // j is smallest integer in Y_2'
46: $I_j.\text{put}(y_2', Y_2')$ // $y_2' = h * g^{-\text{val}(Y_2')}$ is key; Y_2' is value
47: **end for**
48: $H_i.\text{clear}()$
49: **end for**
50: **return** $\perp$ // $\log_g h$ = undefined, $\text{len}_2(\log_g h) > m$, or $\text{wt}_2(\log_g h) \neq t$