Fast Exponentiation Using Split Exponents

Jung Hee Cheon, Stanislaw Jarecki, Taekyoung Kwon, and Mun-Kyu Lee

***Abstract*—We propose a new method to speed up discrete logarithm (DL)-based cryptosystems by considering a new variant of the DL problem, where the exponents are formed as $e_1 + \alpha e_2$ for some fixed α and two integers e_1, e_2 with a low weight representation. We call this class of exponents *split exponents*, and we show that with certain choice of parameters the DL problem on split exponents is essentially as secure as the standard DL problem, while the exponentiation operation using exponents of this class is significantly faster than best exponentiation algorithms given for standard exponents. For example, the speed of scalar multiplication on the standard Koblitz curve K163 is estimated to be accelerated by up to 51.5% and 23.5% at the cost of memory for one precomputed point, compared to the TNAF and window TNAF methods, respectively. As for security, we show that the provable security of the DL problem using split exponents is only by a small constant, e.g. $1/4$, worse than the security of the standard DL problem. Split exponents can be adopted to speed up various DL-based cryptosystems. We exemplify this on the recent CCA-secure public key encryption of Bellare, Kohno, and Shoup.**

***Index Terms*—Exponentiation, Koblitz Curves, Low Hamming Weight, Discrete Logarithm**

I. Introduction

Exponentiation g^e on an abelian group (including modular exponentiation in a finite field and scalar multiplication on an elliptic curve) is the most common primitive operation in public-key cryptography. Since exponentiation consists of repeated field multiplications and squarings, the research on speeding up exponentiation strives to reduce the total number of these component operations as well as the complexity of individual operations. However, while the number of squarings necessary for an exponentiation can be reduced by various (off-line) precomputation methods, and the cost of squarings is even more dramatically reduced in binary fields using normal basis or in Koblitz elliptic curves using Frobenius map [1], it remains a challenge to reduce the number of multiplications.

The most well-known methods for reducing the number of multiplications are the sliding window method [2] and the fixed-base comb method [3]. Though from the details of these algorithms they seem to be quite different, they share the property that repeated patterns are computed only once and reused many times. In other words, they can be reformulated in a unified framework as follows: First, convert the exponent e to a linear combination $e = \sum_i e_i\alpha_i$ of some fixed exponents $\alpha_1 = 1, \alpha_2, \ldots, \alpha_t$. Then compute $g^e = \prod_i g_i^{e_i}$ using the precomputed values $g_i = g^{\alpha_i}$ for each i. In the sliding window method with window size w, the set of α_i's consists of all the additive combinations of small powers of 2, i.e. $2^0, 2^1, \ldots, 2^{w-1}$, whereas the fixed-base comb method uses as the set of α_i's all the additive combinations of elements $2^0, 2^{n/w}, \ldots, 2^{(w-1)n/w}$, where n is the bit-length of the group order p. (Observe that the fixed-base comb method can be used only with *off-line* precomputation since many operations are required to compute g^{α_i}.) In both cases, the number of multiplications required for an exponentiation is *bounded below by* $\mathrm{wt}(e)/w - 1$ for relatively small w where $\mathrm{wt}(e)$ is the Hamming weight of e assuming that g^{α_i}'s are given as precomputed values. (Given g^{α_i}'s, both methods require $\sum_{i=1}^{2^w} \mathrm{wt}(e_i) - 1$ multiplications to compute g^e for $e = \sum_{i=1}^{2^w} e_i\alpha_i$. Since $\mathrm{wt}(e) \le \sum_{i=1}^{2^w} \mathrm{wt}(\alpha_i)\mathrm{wt}(e_i) \le w\sum_{i=1}^{2^w} \mathrm{wt}(e_i)$, the lower bound is obtained.)

We observe that this lower bound comes from the fact that α_i's have a regular form and they have quite small Hamming weights in both methods. Hence, it is an interesting question to look for an exponentiation method to improve this lower bound by using a new form of α_i's.

1) Our Contribution: As the first step in this direction of research, we consider the case with $\alpha_1 = 1$ and $\alpha_2 = \alpha$ where α has a large Hamming weight. To be more precise, we consider the set of exponents

$$S_1 + \alpha S_2 := \{e_1 + \alpha e_2 | e_1 \in S_1, e_2 \in S_2\},$$

where $\alpha \in \mathbb{Z}_p$ and S_1 and S_2 are arbitrary subsets of Z_p. Let us call an element of $S_1 + \alpha S_2$ a *split exponent*. Surprisingly, it turns out that if the product of the cardinalities of S_1 and S_2 is greater than p, the average cardinality of $S_1 + \alpha S_2$ over all $\alpha \in \mathbb{Z}_p$ is at least $p/2$. Furthermore, we show an algorithm for picking a specific *good* α such that $S_1 + \alpha S_2$ covers a significant portion of $\mathbb{Z}_p$, and we show that the provable security of the discrete logarithm problem using split exponents is only by a small constant, e.g. $1/4$, worse than the security of the standard discrete logarithm problem. Our algorithm for picking a good α is efficient only when the sizes of S_1 and S_2 are unbalanced. We pose an open problem to develop a similar algorithm for the case that S_1 and S_2 are of same size.

To reduce the number of multiplications, S_1 and S_2 should be composed of the elements having small Hamming weights. As a specific instance, we take S_1 and S_2 to be the sets

Manuscript received XX; revised XX. Current version published XX. J.H. Cheon and T. Kwon were supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korean government (KRF-2007-314-D00254). J.H. Cheon was supported in part by the NRF grant funded by the Korean government (MEST) (No. 2010-0000218). S. Jarecki was partially supported by the NSF CAREER award #0747541. M.K. Lee was supported in part by the NRF grant funded by the Korean government (MEST) (No. 2010-0016787).

J.H. Cheon is with Department of Mathematical Sciences, Seoul National University, Seoul 151-742, Korea (Email: jhcheon@snu.ac.kr). S. Jarecki is with Department of Computer Science, University of California, Irvine, CA 92697, USA (Email: stasio@ics.uci.edu). T. Kwon is with Department of Computer Engineering, Sejong University, Seoul 143-747, Korea (Email: tkwon@sejong.ac.kr). M.K. Lee is with School of Computer and Information Engineering, Inha University, Incheon 402-751, Korea (Email: mklee@inha.ac.kr). Corresponding Author: Mun-Kyu Lee.

of w-NAFs with small Hamming weight, where w-NAF is a generalization of non-adjacent form (NAF) [4]. Then the number of multiplications required for the exponentiation g^e for $e \in S_1 + \alpha S_2$ given g and g^α is substantially reduced. The advantage of using this new class of exponents is most pronounced in groups where multiplication is much more expensive than squaring, e.g. on binary fields with normal basis representation or on Koblitz curves where we interpret the point addition as a multiplication and the Frobenius map as a squaring, respectively. For example, a scalar multiplication on the standard Koblitz curve K163 [5] requires only 26 point additions, which is a speed-up by 51.5% and 23.5% compared to the TNAF and window TNAF methods [6], [7], respectively. The only overhead to obtain this speedup is memory space for one precomputed point. We remark that to achieve the same speed using precomputation methods (fixed-base window TNAF method), more than seven points should be precomputed and stored.[1]

Finally, we show how split exponents can be used for more efficient implementation of some existing cryptographic schemes based on the discrete logarithm (DL) problem. Namely, we apply the split exponent exponentiation method to the recent encryption scheme of Bellare, Kohno, and Shoup [8] which is the most efficient CCA-secure encryption whose security is provably related to a DL-like problem (the Gap DH problem). Similar adaptations should be possible for long-term secrets in many other DL-based cryptosystems. Also we discuss how to use small Hamming weight exponents to speed up the verification of Schnorr signature.

2) Remark on Applicability: Fast exponentiation is the most required for real-world applications of public key cryptography and such a trend is not changing even in the future ubiquitous computing environments. The notion of split exponent is general in our method, which means that it can be applied to an exponentiation on any abelian group used by public key cryptographic techniques. With regard to usefulness, however, it is more desirable that the group may have fast squaring operation or fast endomorphism for the best performance. Fortunately, there are a number of real-world applications implementing such groups as finite fields of small characteristics and elliptic curves with complex multiplications. They may include not only general applications of public key encryption, digital signature, and authenticated key exchange, but also resource-constrained implementation of them.

It should be noted that our method is the fastest exponentiation algorithm, compared to existing schemes, since we have achieved the fastest exponentiation (scalar multiplication) on Koblitz curves as explained in Section III-B. Our claim is justified easily due to the well-known fact that in general the scalar multiplication on Koblitz curves is always much faster than on other curves. In addition, the memory overhead is only a single elliptic curve point αP, where P is the primitive element of the point group. As a result, there should be a great number of promising applications requiring fast exponentiation through our scheme, e.g., an implementation of public key cryptographic techniques in resource-constrained devices such as embedded devices and tiny sensor nodes [9].

3) Related Work: There are many well known general methods for exponentiation including the binary method, the M-ary method and the sliding window method [2]. Furthermore, if the base g is fixed, precomputation methods such as BGMW [10], de Rooij [11], and the fixed-base comb method [3] can be used to speed up the computation at the cost of memory.

There are works which consider uniformly generated exponents, but represented in a form which makes exponentiation faster. For example, using the property that a point addition and a point subtraction have almost the same complexity on an elliptic curve, one can speed up scalar multiplication with scalars in a signed-digit form such as NAF [4]. Also, for a special class of curves with fast endomorphism, a scalar can be expanded using a complex radix τ, to reduce the number of point doublings [1], [4], [12], [13]. If we are dealing with a fixed point, we can use precomputation methods such as the fixed-base window TNAF method [6], [7].

Recently, Dimitrov, Imbert and Mishra proposed a double-base expansion method for fast scalar multiplication [14], and there are also various extensions to their work [15], [16], [17], [18], [19], [20], [21]. We are more interested in their Koblitz curve variants [22], [23], [24], [25], because these methods concentrate on reduction in the number of point additions rather than point doublings as our method does. However, the fastest algorithm among them [23] requires 31.09 additions on average for a scalar multiplication over K163, which is slower than our algorithm by 16.4%. While there is also another interesting approach combining τ-adic expansion and point halving [26], [27], it is slower than the algorithm in [23] as well as ours.

Of a special interest to us are the methods which reduce the number of multiplications (or point additions) by considering exponents of special form. Studies of small Hamming weight exponents [28], [29] and the more recent study of an exponent class of products of random small Hamming weight factors [30] belong to this category. However, none of these schemes is provably as hard as the standard discrete logarithm problem.

4) Organization: In Section 2 we introduce split exponents and show that the discrete logarithm problem on split exponents can be provably almost equivalent to the standard discrete logarithm problem. In Section 3, we give a practical instance of split exponents, analyze the performance of the exponentiations by split exponents and compare it with that of previous algorithms. In Section 4 we show how split exponents can be used to speed up the Bellare-Kohno-Shoup CCA-secure public key encryption and other DL-based schemes. Section 5 devotes to the Schnorr signature scheme with faster signature verification. We conclude in Section 6.

[1]For groups in which squaring requires non-negligible cost, existing algorithms may outperform our algorithm. For example, by precomputing a group element $g^{2^{n/2}}$, the fixed-base comb method [3] can perform an exponentiation faster than our algorithm. Note that our approach does not try to reduce the number of squarings (or point doublings), but it tries to reduce the number of multiplications (or point additions). On the other hand, the most efficient existing precomputation method for Koblitz curves is not the method of [3], but it is the fixed-base window TNAF method [6], [7]. Our method requires significantly smaller amount of memory than the fixed-base window TNAF method to achieve the same speed.

II. Discrete Logarithm with Split Exponents

In this section, we introduce a new class of exponents and show the hardness of the discrete logarithm problem with such exponents.

Let G be an additive abelian group of prime order p. All addition and multiplication operations, except when noted otherwise, are in $\mathbb{Z}_p$. If $S_1, S_2 \subseteq \mathbb{Z}_p$ and $\alpha \in \mathbb{Z}_p$, we denote as $S_1 + \alpha S_2$ a set of elements $x \in \mathbb{Z}_p$ such that there exists $(x_1, x_2) \in S_1 \times S_2$ s.t. $x = x_1 + \alpha x_2$. We call an element $x \in S_1 + \alpha S_2$, which can be represented as $(x_1, x_2) \in S_1 \times S_2$ s.t. $x = x_1 + \alpha x_2$, a "split exponent". Let $x \stackrel{r}{\leftarrow} X$ denote assignment of a random uniformly chosen value in set X to variable x. The standard Discrete Logarithm (DL) problem is thus the problem of computing x given P, xP where $x \stackrel{r}{\leftarrow} \mathbb{Z}_p$ and P is a generator of G. We define the *split exponent discrete logarithm* problem as the problem of computing x given P, xP for $x \stackrel{r}{\leftarrow} S_1 + \alpha S_2$, where the adversary is additionally given αP:

Definition 1: Let $S_1, S_2 \subseteq \mathbb{Z}_p$. Let P be a generator of G. Let α be an element in $\mathbb{Z}_p$. Let $\mathcal{A}$ be any probabilistic algorithm. We define $\mathcal{A}$'s success probability in solving the *Split Exponent Discrete Logarithm [SEDL]* problem on (S_1, S_2, α, G) as

$$\mathbf{Adv}^{\mathrm{sedl}}_{\mathcal{A},S_1,S_2,\alpha,G} \stackrel{\mathrm{def}}{=} \Pr[\mathcal{A}(P, xP) = x \mid x \stackrel{r}{\leftarrow} S_1 + \alpha S_2].$$

We say that an algorithm $\mathcal{A}$ (t, ϵ)-breaks the SEDL on (S_1, S_2, α, G) if $\mathcal{A}$ runs in time at most t, and $\mathbf{Adv}^{\mathrm{sedl}}_{\mathcal{A},S_1,S_2,\alpha,G}$ is at least ϵ. The (t, ϵ)-SEDL assumption on (S_1, S_2, α, G) is that no adversary (t, ϵ)-breaks the SEDL on (S_1, S_2, α, G).

A. Average Hardness of the Split Exponent DL Problem

The main factor in deciding the hardness of the SEDL problem is the size of set $S_1 + \alpha S_2$. In the following lemma, we give a lower bound on the *average* size of this set for a random α in $\mathbb{Z}_p$, defined as:

$$C_{S_1,S_2} = \frac{1}{p} \sum_{\alpha \in \mathbb{Z}_p} |S_1 + \alpha S_2|.$$

Lemma 1: For any $S_1, S_2 \subseteq \mathbb{Z}_p$ we have:

$$C_{S_1,S_2} \geq \left(\frac{|S_1||S_2|}{|S_1||S_2| + p - |S_1|} \right) \times p.$$

In particular, if $|S_1||S_2| \geq p$ then $C_{S_1,S_2} \geq p/2$.

In other words, for $|S_1||S_2| \geq p$ the *average* hardness of the SEDL problem is just factor of $1/2$ away from the hardness of the DL problem. Perhaps this fact can be directly used in some DL-based cryptosystem. We use it in section II-B to lower-bound the probability of finding some α for which the set $S_1 + \alpha S_2$ is *guaranteed* to be large enough so that the hardness of SEDL *for this particular* α is only a small constant away from the hardness of the DL problem.

Proof: For any $\alpha \in \mathbb{Z}_p$, we define

$$W_\alpha = \{(x, x', y, y') \in S_1^2 \times S_2^2 \mid x + y\alpha = x' + y'\alpha,\ y \neq y'\}.$$

For any 4-tuple $v = (x, y, x', y') \in S_1^2 \times S_2^2$ with $y \neq y'$, we have $v \in W_\alpha$ for $\alpha = (x' - x)/(y - y')$. Therefore, if $W_\alpha \cap W_{\alpha'}$ is nonempty then $\alpha = \alpha'$. Consequently, sets W_α for $\alpha \in \mathbb{Z}_p$ form a partition of all 4-tuples $(x, y, x', y') \in S_1^2 \times S_2^2$ with $y \neq y'$. In particular we have:

$$\sum_{\alpha \in \mathbb{Z}_p} |W_\alpha| = |S_1|^2|S_2|^2 - |S_1|^2|S_2|. \tag{1}$$

For each $\alpha \in \mathbb{Z}_p$, we define a relation $\sim_\alpha$ on the set $S_1 \times S_2$:

$$(x, y) \sim_\alpha (x', y') \Leftrightarrow x + y\alpha = x' + y'\alpha.$$

In other words, $(x, y) \sim_\alpha (x', y')$ if $(x, x', y, y') \in W_\alpha$ or $(x, y) = (x', y')$.

It is easy to see that $\sim_\alpha$ is an equivalence relation on $S_1 \times S_2$ and the number of its equivalence classes is $N_\alpha := |S_1 + \alpha S_2|$. Let $V_{\alpha,1}, \ldots, V_{\alpha,N_\alpha}$ denote the distinct equivalence classes of this relation. Then we have:

$$\sum_{i=1}^{N_\alpha} |V_{\alpha,i}| = |S_1||S_2|, \quad \sum_{i=1}^{N_\alpha} \binom{|V_{\alpha,i}|}{2} 2! = |W_\alpha|. \tag{2}$$

The second equality comes from the fact that an ordered pair of two distinct elements in the same equivalent class corresponds to one element of W_α. In other words, we have:

$$\sum_{i=1}^{N_\alpha} |V_{\alpha,i}|^2 = |W_\alpha| + |S_1||S_2|.$$

By applying Cauchy-Schwarz inequality, we have:

$$\begin{aligned} |S_1|^2|S_2|^2 &= \left(\sum_{i=1}^{N_\alpha} |V_{\alpha,i}| \right)^2 \\ &\leq \left(\sum_{i=1}^{N_\alpha} 1^2 \right) \left(\sum_{i=1}^{N_\alpha} |V_{\alpha,i}|^2 \right) \\ &= N_\alpha(|W_\alpha| + |S_1||S_2|). \end{aligned} \tag{3}$$

We use Cauchy-Schwarz inequality again to obtain:

$$\begin{aligned} \left(\sum_{\alpha \in \mathbb{Z}_p} 1 \right)^2 &\leq \left(\sum_{\alpha \in \mathbb{Z}_p} (|W_\alpha| + |S_1||S_2|) \right) \\ &\quad \times \left(\sum_{\alpha \in \mathbb{Z}_p} \frac{1}{|W_\alpha| + |S_1||S_2|} \right). \end{aligned} \tag{4}$$

Applying equation (1) we have:

$$\sum_{\alpha \in \mathbb{Z}_p} (|W_\alpha| + |S_1||S_2|) = |S_1|^2|S_2|^2 + (p - |S_1|)|S_1||S_2|. \tag{5}$$

Finally, applying inequalities (3) and (4) we derive:

$$\begin{aligned} \frac{1}{p} \sum_{\alpha \in \mathbb{Z}_p} N_\alpha &\geq \frac{|S_1|^2|S_2|^2}{p} \sum_{\alpha \in \mathbb{Z}_p} \frac{1}{|W_\alpha| + |S_1||S_2|} \\ &\geq \frac{p|S_1||S_2|}{|S_1||S_2| + p - |S_1|}. \end{aligned}$$

∎

Corollary 1: Let S_1, S_2 be two subsets of $\mathbb{Z}_p$.

$$\frac{1}{p} \sum_{\alpha \in \mathbb{Z}_p} (|S_1||S_2| - |S_1 + \alpha S_2|) \leq \frac{|S_1|^2|S_2|^2 - |S_1|^2|S_2|}{|S_1||S_2| + p - |S_1|}. \tag{6}$$

In particular, if $|S_1||S_2| \leq \sqrt{p}$, the upper bound is less than 1, and hence the *average* size of set $S_1 + \alpha S_2$ is between $|S_1||S_2| - 1$ and $|S_1||S_2|$.

B. Hardness of the Split Exponent DL Problem for good α's

In the previous subsection we established that for $|S_1||S_2| \geq p$ the average size of set $S_1+\alpha S_2$ is at least $1/2$ the size of $\mathbb{Z}_p$ over all $\alpha \in \mathbb{Z}_p$. Trivially, if $S_1+\alpha S_2 = \mathbb{Z}_p$ for some S_1, S_2, α then SEDL on S_1, S_2, α is at least as hard as the ordinary DL problem. Moreover, if the size of the set $S_1 + \alpha S_2$ is at least a fraction c of the size of the standard DL domain $\mathbb{Z}_p$ then the SEDL problem is related by factor c to the standard DL problem:

Definition 2: Let $S_1, S_2 \subseteq \mathbb{Z}_p$. We call $\alpha \in \mathbb{Z}_p$ "c-good" for S_1, S_2 if $|S_1 + \alpha S_2| \geq cp$.

Theorem 1: Let $S_1, S_2 \subseteq \mathbb{Z}_p$. If α is c-good for S_1, S_2 then the SEDL problem on S_1, S_2, α, G is at least $(t, \epsilon/c)$-hard if the DL problem on G is (t, ϵ)-hard.

Proof: The proof is immediate: If algorithm $\mathcal{A}$ breaks SEDL on S_1, S_2, α, G in time t with probability ϵ/c then same $\mathcal{A}$ breaks DL with probability ϵ because if $|S_1 + \alpha S_2| \geq cp$ then a random $x \in \mathbb{Z}_p$ is in $S_1 + \alpha S_2$ with probability c. ■

Therefore, security of any DL-based cryptosystem degrades only by a factor of c if we replace standard exponents with split exponents for a c-good α. Two issues remain in order to enable us to utilize this fact in a cryptosystem. We need an efficient procedure to pick c-good α's for a good enough constant c, e.g. $c = 1/4$, and we need an efficient way to sample the split exponents given S_1, S_2, α. (Note that choosing $(x_1, x_2) \xleftarrow{r} S_1 \times S_2$ and setting $x = x_1+\alpha x_2$ is *not* equivalent to choosing $x \xleftarrow{r} S_1 + \alpha S_2$!) These two tasks are handled, respectively, by Algorithms 2 and 1 below. Both algorithms are geared to sets S_1, S_2 where S_2 is small and $|S_1+\alpha S_2| \approx p$ as their running times are linear in $|S_2|$ and $p/|S_1 + \alpha S_2|$.

Algorithm 1 Choosing a Random Split Exponent

On input S_1, S_2, α:
1. Randomly select $z \xleftarrow{r} \mathbb{Z}_p$.
2. For all $y \in S_2$, check if $x = z - y\alpha \in S_1$. Output (x, y) and stop if it is.
3. If no $(x, y) \in S_1 \times S_2$ is found, go to State 1.

Algorithm 1 outputs a uniformly distributed element z of $S_1 + \alpha S_2$ (represented as pair $(x, y) \in S_1 \times S_2$), with the expected number of $1/c$ iterations provided that α is c-good. Hence the expected running time of this algorithm is at most $|S_2|/c$ modular multiplications and checks of membership in S_1.

Algorithm 2 Finding a c-good Element α

On input $S_1, S_2 \subseteq \mathbb{Z}_p$ and $\tau \in \mathbb{Z}$. Let $\hat{c} = 2c$.
1. Randomly select $\alpha \xleftarrow{r} \mathbb{Z}_p$.
2. Randomly select $x_1, x_2, \ldots, x_\tau \in \mathbb{Z}_p$ and test if each of them belongs to $S_1 + \alpha S_2$ by Step 2 of Algorithm 1.
3. If the number of x_i's in $S_1 + \alpha S_2$ is at least $\hat{c}\tau$, output α. Otherwise go to State 1.

We first estimate the probability $\mathcal{P}_{fail}$ that some inner loop of Algorithm 2 outputs α which is *not* c-good. Suppose that the outer loop picks α which is not c-good, i.e. $|S_1 + \alpha S_2| = c'p < cp$. Let $X_1, \ldots, X_\tau$ be random variables s.t. $X_i = 1$ if $x_i \in S_1 + \alpha S_2$, and 0 otherwise. Let $X = X_1 + \cdots + X_\tau$. Using this notation, $\mathcal{P}_{fail} = Pr[X > \hat{c}\tau]$ for $\hat{c} = 2c$. If we let $X' = \frac{c}{c'}X$ then the expected value of X' is $\mu = c\tau$. Let $\delta = \hat{c}/c - 1 = 1$. By a Chernoff bound we have:

$$\begin{aligned}\mathcal{P}_{fail} &= Pr[X' > (c/c')\hat{c}\tau] \leq Pr[X' > \hat{c}\tau = (1+\delta)\mu] \\ &< \left(\frac{e^\delta}{(\delta+1)^{\delta+1}}\right)^{c\tau} < 2^{-0.557c\tau}.\end{aligned}$$

In other words, we can set the probability that Algorithm 2 outputs an incorrect α arbitrarily low by setting a large enough τ.

Now we can estimate the expected running time of Algorithm 2. Let $\theta = |S_1||S_2|/p \geq 1$. Lemma 1 says that $E(X_\alpha) \leq \frac{p}{\theta+1}$ for a random variable $X_\alpha := p - |S_1 + \alpha S_2|$. Applying Markov inequality for X_α, we obtain:

$$\Pr[X_\alpha \geq (1-\hat{c})p] \leq \frac{E(X_\alpha)}{(1-\hat{c})p} \leq \frac{1/(\theta+1)}{1-\hat{c}},$$

which implies that a random α in $\mathbb{Z}_p$ is $\hat{c}$-good with probability

$$\mathcal{P}_1 \geq \frac{\theta/(\theta+1) - \hat{c}}{1-\hat{c}}.$$

If α is $\hat{c}$-good, at least $\hat{c}\tau$ elements from randomly selected τ elements from $\mathbb{Z}_p$ belong to $S_1 + \alpha S_2$ with probability

$$\mathcal{P}_2 = \sum_{i \geq \hat{c}\tau} \binom{\tau}{i} \hat{c}^i (1-\hat{c})^{\tau-i} \approx 1/2.$$

If $|S_1||S_2| \geq p$ and we use Algorithm 1 to test if x_i's belong to $S_1 + \alpha S_2$, the expected running time of Algorithm 2 is at most

$$T = (\mathcal{P}_1 \mathcal{P}_2)^{-1} \times \tau \times |S_2|.$$

Example: If $\theta \geq 1$, we may take $c = 1/4$ and $\tau = 576$. Then $\mathcal{P}_{fail} \leq 2^{-80}$. We also have $\mathcal{P}_1 \geq (\theta-1)/(\theta+1)$ and $\mathcal{P}_2 \approx 1/2$, which gives the running time of Algorithm 2:

$$T = 1152|S_2| \cdot (\theta+1)/(\theta-1),$$

which is less than $2^{12}|S_2|$ for $\theta \geq 2$.

III. Implementation of Split Exponents

In this section, we propose to use small Hamming weight w-NAFs as a possible instance of the split sets S_1 and S_2. We show that split exponents implemented in this form substantially accelerate scalar multiplication on Koblitz curves, where a w-NAF is interpreted as a τ-adic representation for a complex number τ. We also show that w-NAF based split exponents give similar speed-up for exponentiation on binary fields represented in normal bases.

A. Low Hamming Weight Exponents

We define w-*NAF*s following the definitions given in [4], [31], [32]:

Definition 3: Let w be an integer ≥ 2 and D a subset of $\mathbb{Z}$ with $0 \notin D$. A w-*NAF* with the digit set D is a sequence of digits satisfying the following two conditions:

1) Each non-zero digit belongs to D.
2) Among any w consecutive digits, at most one is non-zero.

We denote a w-NAF as a string $a = (a_{m-1} \cdots a_1 a_0)$, where m is the *length* of a. The length of a is defined to be the smallest i such that $a_{i-1} \neq 0$. The *(Hamming) weight* of a is defined to be the number of non-zero digits in its w-NAF representation. In this paper, we are especially interested in w-NAFs of *small Hamming weight*.

We will consider the set of split exponents $S_1 + \alpha S_2$ for $\alpha \in \mathbb{Z}_p$ where S_1 and S_2 are the sets of w-NAFs of fixed Hamming weight. The number of w-NAFs of length $\leq m$ and weight t with a coefficient set D is given [31] by

$$\binom{m-(w-1)(t-1)}{t}|D|^t.$$

Given a base k, a w-NAF form can be naturally mapped into the k-adic representation of an integer via $(a_{m-1} \cdots a_1 a_0) \mapsto \sum_{i=0}^{m-1} a_i k^i$. Therefore we can identity a w-NAF with its conversion to an integer, and we can use a w-NAF as an exponent of modular exponentiations in finite fields and scalar multiplications in elliptic curves.

B. Scalar Multiplication on Koblitz Curves by Split Exponents

In this subsection, we introduce a fast scalar multiplication algorithm by split exponents based on small Hamming weight w-NAFs and analyze its performance.

Consider an ordinary elliptic curve E defined over $\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q + 1 - t$ and $\gcd(q, t) = 1$. The Frobenius map τ is defined as follows:

$$\tau : E(\overline{\mathbb{F}}_q) \to E(\overline{\mathbb{F}}_q); (x, y) \mapsto (x^q, y^q),$$

where $\overline{\mathbb{F}}_q$ is the algebraic closure of $\mathbb{F}_q$. The Frobenius map τ is a root of the characteristic equation $\chi_E(T) = T^2 - tT + q$ in $End(E)$. We denote $E(\mathbb{F}_{q^n})$ by the subgroup of $E(\overline{\mathbb{F}}_q)$ consisting of $\mathbb{F}_{q^n}$-rational points. Let G be the subgroup of $E(\mathbb{F}_{q^n})$ generated by P with a prime order ℓ satisfying $\ell^2 \nmid \#E(\mathbb{F}_{q^n})$ and $\ell \nmid \#E(\mathbb{F}_q)$.

We may consider a w-NAF as a τ-adic representation of an integer with the nonzero digit set $D = \{\pm 1, \ldots, \pm(2^{w-1} - 1)\}$. It is called a τ-adic w-NAF and denoted by $a = (a_{m-1} \cdots a_1 a_0)_\tau$ or $a = \sum_{i=0}^{m-1} a_i \tau^i$. Note that given a τ-adic w-NAF $a = (a_{m-1} \cdots a_1 a_0)_\tau$ and a point $Q \in G$, aQ is defined as $aQ := \sum_{i=0}^{m-1} a_i \tau^i(Q)$. The following theorem [31], [33] shows that different τ-adic w-NAFs act as different scalars for scalar multiplication if their lengths are bounded.

Theorem 2: [31], [33] $a = (a_{m-1}, \ldots, a_0)_\tau$ and $b = (b_{m'-1}, \ldots, b_0)_\tau$ be two τ-adic w-NAFs. Then $aQ = bQ$ for some nonzero $Q \in G$ implies that $m = m'$ and $a_i = b_i$ for all i if both of m and m' are equal to or less than

$$M_{q,\ell,w} = \log_q \left(\ell/(q^{w/2} + 1)^2 \right) - (w - 1). \qquad (7)$$

A scalar multiplication by a τ-adic w-NAF can be done similarly to the window TNAF method [6], [7]. The only difference is to use $P_i = iP$ instead of $P_i = (i \mod \tau^w)P$. (Refer to Algorithm 4 in Appendix A.) If a scalar is of the form $k = k_1 + \alpha k_2$ and $Q = \alpha P$ is given together with P, then kP can be computed as $k_1 P + k_2 Q$ using simultaneous scalar multiplication sharing the τ operations. It requires $(2^{w-1} - 2) + (\mathrm{wt}(k_1) + \mathrm{wt}(k_2) - 1)$ point additions, two point doublings and $(m - 1)$ τ operations. (For $w = 2$, no doublings are required.)

We can further reduce the number of point additions by sharing the point additions in the table construction stage. More precisely, given $k_1 = \sum_{j=0}^{m-1} k_{1,j}\tau^j$ and $k_2 = \sum_{j=0}^{m-1} k_{2,j}\tau^j$, we first compute $R_i = \sum_{k_{1,j}=\pm i} \mathrm{sign}(k_{1,j})\tau^j(P) + \sum_{k_{2,j}=\pm i} \mathrm{sign}(k_{2,j})\tau^j(Q)$ for each i and then compute

$$kP = k_1 P + k_2 Q = R_1 + 3R_3 + \cdots + (2^{w-1} - 1)R_{2^{w-1}-1}$$

using the BGMW technique [10]. The detailed procedure is given in Algorithm 3. It requires $\mathrm{wt}(k_1) + \mathrm{wt}(k_2) + 2^{w-2} - 2$ point additions, one point doubling, and $2(m-1)$ τ operations. (For $w = 2$, no doubling is required, since Step 3 is not executed.) Thus it significantly reduces the number of point additions at the cost of additional $(m - 1)$ τ operations.

Algorithm 3 Scalar multiplication by a split scalar

1. Input P, Q, k_1 and k_2.
2. Scanning stage:
   2.1 Set $R_i \leftarrow O$ for $i = 1, 3, 5, \ldots, 2^{w-1} - 1$.
   2.2 For $j = 0$ upto $m - 1$
      2.2.1 Set $R_{|k_{1,j}|} \leftarrow R_{|k_{1,j}|} + sign(k_{1,j})\tau^j(P)$.
      2.2.2 Set $R_{|k_{2,j}|} \leftarrow R_{|k_{2,j}|} + sign(k_{2,j})\tau^j(Q)$.
3. Computation stage for $k_1 P + k_2 Q = R_1 + 3R_3 + \ldots + (2^{w-1} - 1)R_{2^{w-1}-1}$:
   3.1 Set $S \leftarrow R_{2^{w-1}-1}$; Set $T \leftarrow R_{2^{w-1}-1}$.
   3.2 For $i = 2^{w-1} - 3, 2^{w-1} - 5, \ldots, 5, 3$
      3.2.1 Set $S \leftarrow S + R_i$.
      3.2.2 Set $T \leftarrow T + S$.
   3.3 Set $T \leftarrow 2T$.
   3.4 Set $T \leftarrow T + S + R_1$.
4. Output T.

Now we compare the performances of these algorithms with those of existing scalar multiplication algorithms. We consider the standard methods such as the TNAF and window TNAF methods [6], [7] as well as more recently proposed methods using double bases [23], [25], since to our knowledge, they are the fastest algorithms for scalar multiplication with a non-fixed point over a Koblitz curve.

Table I shows appropriate parameters m and t_1, t_2 for various w's. m is chosen to guarantee the uniqueness of τ-adic w-NAFs according to Theorem 2 and t_1, t_2 are chosen so that $|S_2| \approx 2^{40}$ and $|S_1| \times |S_2| \approx 2^{162}$ where S_i $(i = 1, 2)$ is the set of τ-adic w-NAFs of length m and weight t_i. The last three columns show the numbers of point operations required for the window TNAF algorithm and two proposed algorithms over the Koblitz curve K163. The number of point operations for the window TNAF algorithm is $2^{w-2} - 1 + \frac{162}{w+1}$ additions and $162 + \kappa$ applications of τ where κ comes from the table construction [6].

If we use a normal basis to represent the underlying field elements of an elliptic curve, then the computation of a Frobenius map is almost free, and we can ignore the 'T' terms in the table. In this case, the optimal value of w for the window TNAF method is $w = 5$, and its cost is 34A. On the other hand, for the split scalars, the optimal choice is Algorithm 3 with $w = 4$, which requires 25A+1D. Thus assuming the cost for a doubling is approximately the same as that of an addition, the expected speed-ups over TNAF (the special case with $w = 2$) and window TNAF are 51.5% and 23.5%, respectively. Note that the methods in [23] and [25] require 31.09 and 36.37 point additions over K163, respectively, which implies that our method is faster than them by 16.4% and 28.5%.

On the other hand, if a polynomial basis is used, the performance analysis becomes a tedious task because the cost for squaring operations cannot be completely ignored. According to our precise analysis given in Appendix B, the speed-ups over the TNAF and window TNAF methods are expected to be 36–40% and 10–15%, respectively.

C. Binary Fields with Split Exponents

In binary fields, we consider the set of split exponents $S_1 + \alpha S_2 \subseteq \mathbb{Z}_p$ where S_1 and S_2 are the sets of w-NAFs of fixed Hamming weights with the nonzero digit set $D = \{1, 3, 5, \ldots, 2^w - 1\}$. They are called *unsigned* w-NAFs. If we use normal basis representation and replace by a squaring the endomorphism τ in Koblitz curve, we can obtain a similar speed-up to the Koblitz curve case.

Given a split exponent $x = x_1 + \alpha x_2$ for unsigned w-NAFs x_1 and x_2 with length m, each of which has weight t_1 and t_2 respectively, we can compute g^x for a finite field element g in two ways as in the previous subsection. The first method is to individually compute g^{x_1} and $(g^\alpha)^{x_2}$ and multiply them. (For an individual exponentiation by an unsigned w-NAF, refer to Algorithm 5 in Appendix C.) It requires $t_1 + t_2 + 2^w - 3$ multiplications.

The second algorithm is to reduce the number of multiplications using the BGMW technique [10]. Its complexity is $t_1 + t_2 + 2^{w-1} - 2$. (For the detailed procedure, refer to Algorithm 6 in Appendix C.)

Now we need to consider how to generate unsigned w-NAFs uniformly. One can easily show that every positive integer has exactly one unsigned w-NAF [29]. Moreover, all unsigned w-NAFs of length $\leq \lfloor \log p \rfloor - w + 1$ are distinct.

Table II compares the performance when m is the largest integer less than or equal to $\lfloor \log p \rfloor - w + 1$. Note that typical cryptographic applications use short exponents of 160 bits over finite fields of order 2^{1024} to guarantee the 2^{80} security. Thus we assume that the exponentiation algorithms use 160-bit exponents and $|S_1||S_2| \approx 2^{160}$. According to this table, the expected speed-up is 23.7%, which is similar to the case of Koblitz curves.

We remark that as m gets larger, the performance gain is expected to be better since we can reduce t_1 and t_2. But when m is larger than $\lfloor \log p \rfloor - w + 1$, distinct unsigned w-NAFs of length m can be congruent modulo p. It would be an interesting problem to devise an algorithm to pick w-NAFs almost uniformly from $\mathbb{Z}_p$ for larger m.

IV. Applications of Exponentiation by Split Exponents

In this section, we show how split exponents can be adopted in cryptographic schemes based on the discrete logarithm problem, and we show the efficiency gains resulting from these modifications.

A. Public Key Encryption

Bellare, Kohno, and Shoup [8] proposed a CCA-secure version of ElGamal encryption that achieves the fastest encryption and decryption among ElGamal-based schemes by re-using the ephemeral ElGamal key. We show that usage of split exponents can speed up both the encryption and the decryption operations even further at the cost of doubling the size of public keys and increasing the setup cost.

Setup: Let G be a prime-order subgroup of elliptic curve points from a Koblitz curve defined over $\mathbb{F}_{2^{163}}$, where p is its order and P is a generator. Let S_1 and S_2 be two subsets of $\mathbb{Z}_p$ s.t. $|S_1||S_2| \geq 2^{160}$, $|S_1| \approx 2^{120}$, and $|S_2| \approx 2^{40}$. A $1/4$-good element $\alpha \in \mathbb{Z}_p$ for S_1, S_2 is chosen using Algorithm 2, and the public parameters are $Param = \langle G, S_1, S_2, P, P_1 := \alpha P, P_2 := \alpha^2 P, \mathcal{E}, H \rangle$, where $\mathcal{E}$ is a CCA-secure symmetric key encryption scheme and H is a collision resistant hash function from bit strings onto group G, modeled as a random oracle in the security analysis.

Key Generation: Each user U_i chooses a secret key $x_i + \alpha y_i$ uniformly from $S_1 + \alpha S_2$, and publishes a public key $\langle X_i, Y_i \rangle$ such that $X_i = x_i P + y_i P_1$ and $Y_i = x_i P_1 + y_i P_2$. Similarly (s)he chooses an ephemeral secret key $u_i + \alpha v_i$ uniformly from $S_1 + \alpha S_2$, and computes $\langle R_i, S_i \rangle$ such that $R_i = u_i P + v_i P_1$ and $S_i = u_i P_1 + v_i P_2$.

Encryption: The encryption of a message m of the user U_j to the user U_i is $\langle R_j, S_j, C \rangle$ where $C = \mathcal{E}_K(m)$ and $K = H(R_j, S_j, X_i, Y_i, u_j X_i + v_j Y_i)$.

Decryption: Given $\langle R_j, S_j, C \rangle$, compute $K = H(R_j, S_j, X_i, Y_i, x_i R_j + y_i S_j)$ and decrypt the ciphertext $C = \mathcal{E}_K(m)$ using K.

A corresponding argument to Theorem 1 shows that if α is $1/4$-good then the hardness of the Gap Diffie Hellman Problem on group G on split exponents is related by factor $1/16$ to the hardness of the Gap Diffie Hellman Problem (see e.g. [8]). Therefore the "chosen sender and receiver" security of the BKS encryption scheme (where the adversary attacks a fixed sender/receiver pair of players) is provably related by factor $1/16$ to the Gap Diffie Hellman Problem. The reduction that shows this follows the reduction in [8]. Since the reduction is for "chosen sender chosen receiver" adversary, the reduction does not have to sample the split exponent keys of other players in the network. Also, note that the encryption and

TABLE I
PERFORMANCE OF VARIOUS SCALAR MULTIPLICATION ALGORITHMS ON K163 (A: ADDITION, D: DOUBLING, T: COMPUTATION OF THE τ MAP)

Parameters				$\lvert S_1\rvert$	$\lvert S_2\rvert$	Number of point operations		
w	m	t_1	t_2			Win.TNAF [6]	2×Alg. 4	Alg. 3
2	157	28	6	1.1×2^{122}	1.8×2^{39}	54A+162T	33A+156T	33A+312T
3	156	23	5	1.5×2^{124}	1.0×2^{39}	42A+163T	29A+2D+155T	28A+1D+310T
4	154	18	5	1.5×2^{119}	1.7×2^{43}	36A+165T	28A+2D+153T	25A+1D+306T
5	152	17	4	1.1×2^{127}	1.8×2^{39}	34A+168T	34A+2D+151T	27A+1D+302T
6	150	14	4	1.7×2^{121}	1.6×2^{44}	38A+174T	47A+2D+149T	32A+1D+298T

TABLE II
PERFORMANCE OF VARIOUS EXPONENTIATION ALGORITHMS FOR 160-BIT EXPONENT

Parameters				$\lvert S_1\rvert$	$\lvert S_2\rvert$	Number of multiplications		
w	m	t_1	t_2			Alg. 5	2×Alg. 5	Alg. 6
2	159	27	6	1.2×2^{120}	1.0×2^{40}	53	34	33
3	158	22	5	1.8×2^{121}	1.1×2^{39}	42	**32**	**29**
4	157	19	4	1.7×2^{124}	1.1×2^{36}	**38**	36	**29**
5	156	16	4	1.1×2^{123}	1.0×2^{40}	41	49	34
6	155	13	4	1.5×2^{116}	1.8×2^{43}	53	78	47

decryption procedures can avoid having to verify whether the receiver's or sender's keys are elements in group G because points on this Koblitz curve form a group G' of size $2p$, and existence of the DDH oracle on G' is therefore implied by the existence of the DDH oracle on G. (The first oracle can be implemented with a single call to the second one together and a single test of membership in G.)

Compared with the original scheme, we could see from Table I that both encryption and decryption are accelerated by 23.5% with normal basis (and 10–15% with polynomial basis) by virtue of split exponents. Though the size of public key is doubled and the setup cost is increased to select a $1/4$-good element α, the computational cost of both encryption and decryption are significantly reduced by using split exponents.

B. Other Diffie-Hellman and ElGamal Variants

The same speed-up can be achieved in any Diffie-Hellman system which uses fixed exponents, where the relatively high cost of uniform sampling of set $S_1 + \alpha S_2$ can be amortized. For example, the Diffie-Hellman Key Agreement with fixed exponents can be made more efficient at the cost of doubling the public key size. If the public key is $\langle X_i, Y_i\rangle$ where $X_i = (x_i + \alpha y_i)P$ and $Y_i = \alpha X_i$, we could speed up the shared key computation by 23.5% again, since $x_j X_i + y_j Y_i = (x_i + \alpha y_i)(x_j + \alpha y_j)P = x_i X_j + y_i Y_j$.

In versions of ElGamal encryption where the ephemeral key is refreshed for each encryption the split exponents can only be used for the public keys, i.e. U_i's public key is a pair X_i, Y_i but the ephemeral key is just $R_i = uP$ for $u \xleftarrow{r} \mathbb{Z}_p$. This would slow down encryption process, but it would accelerate the decryption operation by 23.5%.

V. SCHNORR SIGNATURES WITH w-NAF

Note that the advantage of split exponents comes from the condition that each of S_i consists of $\sqrt{p}$ elements of small Hamming weight chosen from p elements. Thus if the set of challenges of the Schnorr signature consists of $\sqrt{p}$ elements, we can enjoy the advantage of small Hamming weight exponents without using split exponents.

In Asiacrypt 2000, Schnorr and Jacobson analyzed [34] the security of Schnorr signatures in the generic group model and the random oracle model for the hash function, and showed that the scheme has 2^{80} security level in the model as long as the challenge c is chosen uniformly in *any* set with 2^{80} elements.

We describe a variant of the Schnorr signature scheme on Koblitz curves where the challenge c in the NIZK for the discrete logarithm on which the Schnorr signature scheme is based is not a random element in $\mathbb{Z}_p$ but a value c uniformly chosen from the set S of w-NAFs of fixed weight. Since there is an efficient algorithm to pick up a w-NAF integer of fixed weight t [31], we can easily obtain an efficient full-domain hash function to S by the standard technique.

Setup: Let G be a prime-order subgroup of elliptic curve points from a Koblitz curve defined over $\mathbb{F}_{2^{163}}$, where p is its order and P is a generator. Let S be a subset of $\mathbb{Z}_p$. The public parameters are $Param = \langle G, S, P, H\rangle$, where H is a collision resistant full domain hash function $H : \{0,1\}^* \to S$.

Key Generation: Each user U_i chooses a secret key $x_i \xleftarrow{r} \mathbb{Z}_p$ and publishes a public key $X_i = x_i P$.

Signing: A signature on message m is a tuple $\langle m, c, s\rangle$ where $R = -kP$, $c = H(m, R)$ and $s = k + cx_i$, for $k \xleftarrow{r} \mathbb{Z}_p$.

Verification: Signature $\langle c, s\rangle$ on m is accepted if

$$c = H(m, -sP + cX_i).$$

The verification of Schnorr signature mainly consists of two scalar multiplications, that is, $-sP$ and cX_i. The above variant improves the speed of cX_i computation. If we use $w = 4$ and $t = 11$, Algorithm 4 requires only 14 elliptic curve additions, so the speed-ups using a normal basis is $(19 - 14)/19 = 26.3\%$ over the window TNAF. For the number of elliptic curve additions required to compute cX_i for other parameters, refer to Table III. Note that the first scalar multiplication is a fixed point multiplication, so it can be easily sped up using various fixed-base precomputation methods.

We may consider split exponents for challenges of Schnorr

TABLE III
COMPUTATIONAL REQUIREMENTS FOR cX_i (A: ADDITION D: DOUBLING T: COMPUTATION OF THE τ MAP)

Parameters			$\|S\|$	Number of point operations	
w	m	t		Win.TNAF [6], [7]	Alg. 4
2	157	15	1.0×2^{81}	27A+79T	14A+156T
3	156	13	1.1×2^{84}	21A+80T	13A+1D+155T
4	154	11	1.5×2^{83}	19A+82T	13A+1D+153T
5	152	9	1.2×2^{79}	21A+85T	15A+1D+151T
6	150	8	1.1×2^{79}	27A+91T	22A+1D+149T

signatures. That is, the challenge value $c = (c_1, c_2)$ is chosen uniformly from $S_1 \times S_2$ where each of S_i is a set of w-NAF of fixed Hamming weight. Then we need to publish one more public parameter αP for $\alpha \in \mathbb{Z}_p$ and one more public key $Y_i = \alpha X_i$ for each user U_i. Then the signature is a tuple $\langle m, c_1, c_2, s\rangle$ for a message m where $R = -kP$, $(c_1, c_2) = H(m, R)$, and $s = k + (c_1 + \alpha c_2)x_i$ for $k \xleftarrow{r} \mathbb{Z}_p$. The signature is verified by checking $(c_1, c_2) = H(m, -sP + c_iX_i + c_2Y_i)$.

By Corollary 1, if $|S_1| = |S_2| = 2^{40}$ then the average size of $|S_1 + \alpha S_2|$, for random α, is at least $2^{80} - 1$. Hence we may assume that for an overwhelming fraction of α's the size of $|S_1 + \alpha S_2|$ is indeed very close to 2^{80}, which would imply almost the same $\sqrt{p}$ lower bound on the complexity of forging a signature as the argument in [34]. If we use 3-NAF of weight 5 for (c_1, c_2), the $c_iX_i + c_2Y_i$ takes only 11 elliptic curve additions which is an improvement of 42.1% over the window TNAF in normal basis representation. For a specific α, however, we need to investigate more the distribution of $|S_1 + \alpha S_2|$.

VI. CONCLUSION AND FURTHER STUDY

In this paper, we proposed a new variant of the discrete logarithm problem, called SEDL, which is the discrete logarithm on a class of exponents we call *split exponents*, and we showed how to adopt these exponents to speed up encryption and decryption operations in a BKS encryption scheme [8].

An interesting open problem is to show hardness of the SEDL problem on all but negligible fraction of α's. In this paper, we showed that the SEDL problem is hard for a random α, and hence we also showed that the hardness of the SEDL problem is a small constant away from the hardness of the standard DL problem for certain "good" α's. However, we showed an efficient generation of such α's only if S_1 and S_2 are unbalanced, e.g. $|S_1| = 2^{120}$ and $|S_2| = 2^{40}$. Furthermore, we showed an efficient sampling algorithm from the set of split exponents for such a fixed good α, also only in the case when set S_2 is much smaller than S_1. Finally, both algorithms, the one for finding good α's and the sampling algorithm, are time consuming, which limits their applicability. If the SEDL problem was hard for all but negligible α's then split exponents could be used to speed up a much larger class of DL-based cryptosystems, because they could then be applied to ephemeral values in these cryptosystems, and not just to the long-term private keys, as in our variant of the BKS encryption scheme. Similarly, extending our results to sets S_1 and S_2 of the same size would result in further speeding up of exponentiation operations.

In this paper, we compared the performance of split exponent-based exponentiation with those of existing exponentiation algorithms only in the 80-bit security setting. But it will also be interesting to analyze the significance of split exponents from an asymptotic point of view as the security parameter grows. In addition, incorporating split exponents in the double-base setting such as [22], [23], [25] will be a promising research direction.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments.

REFERENCES

[1] N. Koblitz, "CM-curves with good cryptographic properties," in *CRYPTO 91*, ser. LNCS, vol. 547. Springer, 1992, pp. 279–287.
[2] D. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 3rd ed. Addison-Wesley, 1998.
[3] C. Lim and P. Lee, "More flexible exponentiation with precomputation." in *CRYPTO 94*, ser. LNCS, vol. 839. Springer, 1994, pp. 95–107.
[4] J. Solinas, "Efficient arithmetic on Koblitz curves," *Designs, Codes and Cryptography*, vol. 19, pp. 195–249, 2000.
[5] NIST, "Recommended elliptic curves for federal government use," 1999.
[6] D. Hankerson, J. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *CHES 2000*, ser. LNCS, vol. 1965. Springer, 2000, pp. 1–24.
[7] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer, 2004.
[8] M. Bellare, T. Kohno, and V. Shoup, "Stateful public-key cryptosystems: How to encrypt with one 160-bit exponentiation," in *ACM CCS 2006*, 2006, pp. 380–389.
[9] S. Seo, D. Han, H. Kim, and S. Hong, "TinyECCK: Efficient elliptic curve cryptography implementation over $GF(2^m)$ on 8-bit Micaz mote," *IEICE Trans. Inf. Sys.*, vol. 91-D, no. 5, pp. 1338–1347, 2008.
[10] E. Brickell, D. Gordon, K. McCurley, and D. Wilson, "Fast exponentiation with precomputation," in *EUROCRYPT 92*, ser. LNCS, vol. 658. Springer, 1993, pp. 200–207.
[11] P. de Rooij, "Efficient exponentiation using precomputation and vector addition chain," in *EUROCRYPT 94*, ser. LNCS, vol. 950. Springer, 1994, pp. 389–399.
[12] N. P. Smart, "Elliptic curve cryptosystems over small fields of odd characteristic," *Journal of Cryptology*, vol. 12, no. 2, pp. 141–151, 1999.
[13] R. Gallant, R. Lambert, and S. Vanstone, "Faster point multiplication on elliptic curves with efficient endomorphisms," in *CRYPTO 2001*, ser. LNCS, vol. 2139. Springer, 2001, pp. 190–200.
[14] V. Dimitrov, L. Imbert, and P. Mishra, "Efficient and secure elliptic curve point multiplication using double-base chains," in *ASIACRYPT 2005*, ser. LNCS, vol. 3788. Springer, 2005, pp. 59–78.
[15] M. Ciet and F. Sica, "An analysis of double base number systems and a sublinear scalar multiplication algorithm," in *MYCRYPT 2005*, ser. LNCS, vol. 3715. Springer, 2005, pp. 171–182.
[16] C. Doche and L. Imbert, "Extended double-base number system with applications to elliptic curve cryptography," in *INDOCRYPT 2006*, ser. LNCS, vol. 4329. Springer, 2006, pp. 335–348.
[17] P. Mishra and V. Dimitrov, "Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation," in *ISC 2007*, ser. LNCS, vol. 4779. Springer, 2007, pp. 390–406.
[18] D. Bernstein, P. Birkner, T. Lange, and C. Peters, "Optimizing double-base elliptic-curve single-scalar multiplication," in *INDOCRYPT 2007*, ser. LNCS, vol. 4859. Springer, 2007, pp. 167–182.
[19] P. Mishra and V. Dimitrov, "A graph theoretic analysis of double base number systems," in *INDOCRYPT 2007*, ser. LNCS, vol. 4859. Springer, 2007, pp. 152–166.
[20] C. Doche and L. Habsieger, "A tree-based approach for computing double-base chains," in *ACISP 2008*, ser. LNCS, vol. 5107. Springer, 2008, pp. 433–446.
[21] C. Doche, D. Kohel, and F. Sica, "Double-base number system for multi-scalar multiplications," in *EUROCRYPT 2009*, ser. LNCS, vol. 5479. Springer, 2009, pp. 502–517.

[22] R. Avanzi and F. Sica, “Scalar multiplication on Koblitz curves using double bases,” in *VIETCRYPT 2006*, ser. LNCS, vol. 4341. Springer, 2006, pp. 131–146.

[23] R. Avanzi, V. Dimitrov, C. Doche, and F. Sica, “Extending scalar multiplication using double bases,” in *ASIACRYPT 2006*, ser. LNCS, vol. 4284. Springer, 2006, pp. 130–144.

[24] V. Dimitrov, K. Järvinen, M. Jacobson, W. Chan, and Z. Huang, “FPGA implementation of point multiplication on Koblitz curves using Kleinian integers,” in *CHES 2006*, ser. LNCS, vol. 4249. Springer, 2006, pp. 445–459.

[25] ——, “Provably sublinear point multiplication on Koblitz curves and its hardware implementation,” *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1469–1481, 2008.

[26] R. Avanzi, M. Ciet, and F. Sica, “Faster scalar multiplication on Koblitz curves combining point halving with the Frobenius endomorphism,” in *PKC 2004*, ser. LNCS, vol. 2947. Springer, 2004, pp. 28–40.

[27] R. Avanzi, C. Heuberger, and H. Prodinger, “Minimality of the Hamming weight of the τ-NAF for Koblitz curves and improved combination with point halving,” in *SAC 2005*, ser. LNCS, vol. 3897. Springer, 2005, pp. 332–344.

[28] D. Coppersmith and G. Seroussi, “On the minimum distance of some quadratic residue codes.” *IEEE Transactions on Information Theory*, vol. 30, no. 2, pp. 407–410, 1984.

[29] D. Stinson, “Some baby step giant step algorithms for the low hamming weight discrete logarithm problem,” *Mathematics of Computation*, vol. 71, no. 237, pp. 379–391, 2002.

[30] J. Hoffstein and J. H. Silverman, “Random small hamming weight products with applications to cryptography,” *Discrete Applied Mathematics*, vol. 130, no. 1, pp. 37–49, 2003.

[31] J. Cheon and J. Yi, “Fast batch verification of multiple signatures,” in *PKC 2007*, ser. LNCS, vol. 4450. Springer, 2007, pp. 442–457.

[32] J. Muir and D. Stinson, “Minimality and other properties of the width-w non-adjacent form,” *Mathematics of Computation*, vol. 75, pp. 369–384, 2006.

[33] T. Lange and I. Shparlinski, “Collisions in fast generation of ideal classes and points on hyperelliptic and elliptic curves.” *Applicable Algebra in Engineering, Communication and Computing*, vol. 15, no. 5, pp. 329–337, 2005.

[34] C. Schnorr and M. Jakobsson, “Security of signed ElGamal encryption,” in *ASIACRYPT 2000*, ser. LNCS, vol. 1976, 2000, pp. 73–89.

[35] J. López and R. Dahab, “Improved algorithms for elliptic curve arithmetic in $GF(2^n)$,” in *SAC 98*, ser. LNCS, vol. 1556. Springer, 1999, pp. 201–212.

[36] G. Agnew, R. Mullin, and S. Vanstone, “Fast exponentiation in $GF(2^n)$,” in *EUROCRYPT 88*, ser. LNCS, vol. 330. Springer, 1988, pp. 251–255.

Appendix A
Scalar Multiplication Algorithm for τ-adic w-NAFs

We introduce a simple scalar multiplication algorithm using τ-adic w-NAF, Algorithm 4. Note that the window TNAF method, i.e., τ-adic NAF window method, given in [6], [7], is similar to Algorithm 4, but it uses $P_i = \alpha_i P$ instead of $P_i = iP$, where $\alpha_i = i \bmod \tau^w$. We can easily see that Algorithm 4 requires $(2^{w-2}-1)+(\mathrm{wt}(k)-1)$ point additions, one point doubling and $(m-1)$ τ operations, where $\mathrm{wt}(k)$ is the weight of k. It also requires temporary memory to store $2^{w-2}-1$ points, i.e., $3P, 5P, \ldots, (2^{w-1}-1)P$. For $w = 2$, we need no doubling, since the table construction stage is not required.

Algorithm 4 Scalar Multiplication by a τ-adic w-NAF

1. Input P and $k = (k_{m-1}k_{m-2}\ldots k_0)_\tau$, where k is given as a τ-adic w-NAF.
2. Table construction stage:
   2.1 Set $P_0 \leftarrow O$, $P_1 \leftarrow P$ and $P_2 \leftarrow 2P$.
   2.2 For $j = 1$ upto $2^{w-2}-1$, set $P_{2j+1} \leftarrow P_{2j-1} + P_2$.
3. Scalar multiplication stage:
   3.1 Find the largest i s.t. $k_i \neq 0$. Set $Q \leftarrow sign(k_i)P_{|k_i|}$.
   3.2 For $j = i-1$ downto 0
      3.2.1 Set $Q \leftarrow \tau Q$.
      3.2.2 if $k_j \neq 0$ then set $Q \leftarrow Q + sign(k_j)P_{|k_j|}$.
4. Output Q.

Appendix B
Comparison of the Window TNAF Algorithm, Algorithm 4 and Algorithm 3 in Polynomial Basis Representation

In this section, we precisely compare the performances of the TNAF algorithm, the window TNAF algorithm, 2×Algorithm 4 and Algorithm 3 when the underlying binary field is represented as a polynomial basis. First, we revise Table I to discriminate the computational requirements for each part of the algorithms, and construct Table IV. That is, we denoted the cost for table construction or scanning and the cost for other parts as separate terms in the new table.

If the point operations are done in affine coordinates, we can obtain the relations $1A = 1I+2M+1S$, $1D = 1I+2M+2S$, and $1T = 2S$, where I, M, S represent the computational costs for a field inversion, a field multiplication, and a field squaring, respectively. On the other hand, if we consider projective coordinates, it is desirable to use a mixed coordinate system where a doubling is performed using two points in López-Dahab (LD) projective coordinates and an addition is performed with one point represented in LD projective coordinates and the other in affine coordinates [35]. In this case, the costs for a doubling and an addition are $3M + 5S$ and $8M + 5S$, respectively [7], [35]. However, in order to use this coordinate system, some part of the scalar multiplication should still be done in affine coordinates. For example, for the window TNAF method, table construction is done in affine coordinates and the remaining part is done in mixed coordinates. The final result of a scalar multiplication is obtained by converting the result of the last point operation into affine coordinates, which requires $1I+2M+1S$ since an LD projective point $(X : Y : Z), Z \neq 0$ corresponds to the affine point $(X/Z, Y/Z^2)$.

For 2×Algorithm 4, we have two options to use mixed coordinates. The first choice is to apply a similar approach to the window TNAF method. That is, table construction is done in affine coordinates and the remaining part is computed using mixed-coordinate additions and LD Frobenius maps. Another choice is to use mixed coordinates in table construction and LD projective additions and LD Frobenius maps in the remaining computation. Note that the costs of an LD projective addition and an LD Frobenius map are $13M + 6S$ and $3S$, respectively. According to our analysis, the first option is preferred in most cases. Therefore we consider only the first option.

We also have two options for Algorithm 3. The first choice is to use affine coordinates in Step 2. Then in Step 3, the computation of S's is done in mixed coordinates, and T's are computed using LD projective additions. The second choice is to use affine Frobenius maps and mixed additions in Step

TABLE IV
PERFORMANCE OF VARIOUS SCALAR MULTIPLICATION ALGORITHMS ON K163 WITH $|S_1||S_2| \approx 2^{162}$ AND $|S_2| \approx 2^{40}$ (A: ADDITION, D: DOUBLING, T: COMPUTATION OF THE τ MAP)

Parameters				$\|S_1\|$	$\|S_2\|$	Number of point operations		
w	m	t_1	t_2			Win.TNAF [6]	2×Alg. 4	Alg. 3†
2	157	28	6	1.1×2^{122}	1.8×2^{39}	(0A+0T)+(54A+162T)	(0A+0D)+(33A+156T)	(33A+312T)+(0A+0D)
3	156	23	5	1.5×2^{124}	1.0×2^{39}	(1A+1T)+(41A+162T)	(2A+2D)+(27A+155T)	(26A+310T)+(2A+1D)
4	154	18	5	1.5×2^{119}	1.7×2^{43}	(3A+3T)+(33A+162T)	(6A+2D)+(22A+153T)	(19A+306T)+(6A+1D)
5	152	17	4	1.1×2^{127}	1.8×2^{39}	(7A+6T)+(27A+162T)	(14A+2D)+(20A+151T)	(13A+302T)+(14A+1D)
6	150	14	4	1.7×2^{121}	1.6×2^{44}	(15A+12T)+(23A+162T)	(30A+2D)+(17A+149T)	(2A+298T)+(30A+1D)

†We assume that every possible coefficient appears at least once.

2, and LD projective additions in Step 3. Let us denote these two options as *m-1* and *m-2*.

TABLE VI
TOTAL COST REPRESENTED IN THE NUMBER OF SQUARINGS USING THE ESTIMATION $I/M = 5, M/S = 7$ [7]

	Win. TNAF [6]		2×Alg. 4		Alg. 3		
w	affine	mixed	affine	mixed	affine	m-1	m-2
2	3024	3830	1962	2531	2274	2324	2687
3	2426	3089	1862	2364	2071	2154	2476
4	2130	2705	**1808**	**2253**	**1913**	**2112**	**2429**
5	**2036**	**2545**	2104	2525	2005	2436	2831
6	2248	2713	2750	3136	2247	3142	3704

TABLE VII
TOTAL COST REPRESENTED IN THE NUMBER OF SQUARINGS USING THE ESTIMATION $I/M = 8, M/S = 7$ [7]

	Win. TNAF [6]		2×Alg. 4		Alg. 3		
w	affine	mixed	affine	mixed	affine	m-1	m-2
2	4158	3851	2655	2552	2967	3038	2708
3	3308	3131	2513	2469	2680	2721	2497
4	2886	2789	**2438**	**2442**	**2459**	**2532**	**2450**
5	**2750**	**2713**	2860	2882	2593	2730	2852
6	3046	3049	3779	3829	2940	3205	3725

TABLE VIII
TOTAL COST REPRESENTED IN THE NUMBER OF SQUARINGS USING THE ESTIMATION $I/M = 10, M/S = 10$ [6]

	Win. TNAF [6]		2×Alg. 4		Alg. 3		
w	affine	mixed	affine	mixed	affine	m-1	m-2
2	6858	5197	4305	3394	4617	4738	3550
3	5408	4215	4063	**3367**	4130	4143	3258
4	4686	3781	**3938**	3420	**3759**	**3730**	**3199**
5	**4450**	**3761**	4660	4212	3993	3880	3769
6	4946	4401	6229	5887	4590	4309	5002

Tables V shows the number of field operations for the three scalar multiplication algorithms using the above coordinate systems. Tables VI, VII and VIII present the total costs represented in the number of squarings using typical estimations for the ratios I/M and M/S according to the data given in [6], [7]. The minimum values over possible choices of w are highlighted. We can summarize the three tables as follows:

- If $I/M = 5$, then affine coordinates are preferable. In this case, the parallel execution of Algorithm 4 with window size $w = 4$ is the best choice for the computation of $k_1P + k_2(\alpha P)$, and it is faster than the TNAF algorithm ($w = 2$) and the window TNAF algorithm for kP by 40.21% and 11.20%, respectively.
- If $I/M = 8$, then mixed coordinates are better than affine coordinates for the window TNAF algorithm. But these two coordinates provide similar performances for the other two algorithms. Consequently, the parallel execution of Algorithm 4 with window size $w = 4$ is the best choice, and the speed-ups over the TNAF and window TNAF methods are 36.69% and 10.14%, respectively.
- If $I/M = 10$, mixed coordinates are preferable in most cases. Algorithm 3 ($w = 4$) with m-2 coordinates is the best choice, and it is faster than the TNAF and the window TNAF algorithms by 38.45% and 14.94%, respectively.

APPENDIX C
EXPONENTIATION USING SPARSE UNSIGNED w-NAFS

In this section, we present exponentiation algorithms using split sparse exponents. First, we start with a simple algorithm (Algorithm 5) that uses free squaring over a normal basis, which can be seen as an improved version of [36]. It requires $(2^{w-1}-1)+(\mathrm{wt}(x)-1) = 2^{w-1}+\mathrm{wt}(x)-2$ multiplications, where $\mathrm{wt}(x)$ denotes the weight of x.

Algorithm 5 Exponentiation g^x using normal basis representation

1. Input g and $x = (x_{m-1} \dots x_0)_2$, where g is represented by a normal basis and x is given as an unsigned w-NAF.
2. Table construction stage:
   2.1 Set $g_0 \leftarrow 1$, $g_1 \leftarrow g$ and $g_2 \leftarrow g^2$.
   2.2 For $j = 1$ upto $2^{w-1} - 1$, set $g_{2j+1} \leftarrow g_{2j-1} \times g_2$.
3. Exponentiation stage:
   3.1 Find the largest i such that $x_i \neq 0$. Then set $y \leftarrow g_{x_i}$.
   3.2 For $j = i - 1$ downto 0
      3.2.1 Set $y \leftarrow y^2$.
      3.2.2 if $x_j \neq 0$ then set $y \leftarrow y \times g_{x_j}$.
4. Output y.

If an exponent is of the form $x = x_1 + \alpha x_2$ and $h = g^\alpha$ is given as input together with g, then g^x can be computed as $g^{x_1}h^{x_2}$. A naive algorithm to compute $g^{x_1}h^{x_2}$ is to use Algorithm 5 twice, and it requires $2(2^{w-1}-1)+(\mathrm{wt}(x_1)-1)+(\mathrm{wt}(x_2)-1)+1 = 2^w+\mathrm{wt}(x_1)+\mathrm{wt}(x_2)-3$ multiplications. However, a modification of the BGMW method [10] gives a significant speed-up: the number of required multiplications for Algorithm 6 is $\mathrm{wt}(x_1) + \mathrm{wt}(x_2) + 2^{w-1} - 2$.

TABLE V
COMPARISON OF THE NUMBER OF FIELD OPERATIONS FOR THREE SCALAR MULTIPLICATION ALGORITHMS

	Win. TNAF [6]						2×Alg. 4						Alg. 3								
	affine			mixed			affine			mixed			affine			m-1			m-2		
w	I	M	S	I	M	S	I	M	S	I	M	S	I	M	S	I	M	S	I	M	S
2	54	108	378	1	434	757	33	66	345	1	266	634	33	66	657	34	68	658	1	266	790
3	42	84	368	2	332	695	31	62	343	5	226	607	29	58	650	27	78	663	1	239	768
4	36	72	366	4	272	661	30	60	338	9	194	580	26	52	639	20	106	670	1	235	749
5	34	68	370	8	232	641	36	72	340	17	194	572	28	56	633	14	178	700	1	291	759
6	38	76	386	16	216	641	49	98	349	33	202	567	33	66	630	3	324	769	1	411	792

Algorithm 6 Exponentiation by a split exponent

1. Input g, h, x_1 and x_2.
2. Set $s \leftarrow 1$, $t \leftarrow 1$.
3. For $i = 2^w - 1, 2^w - 3, \ldots, 3, 1$
   3.1 For each j such that $x_{1,j} = i$, set $s \leftarrow s \times g^{2^j}$.
   3.2 For each j such that $x_{2,j} = i$, set $s \leftarrow s \times h^{2^j}$.
   3.3 if $i = 1$ then set $t \leftarrow t \times s$; else set $t \leftarrow t \times s^2$.
4. Output t.

PLACE
PHOTO
HERE

Jung Hee Cheon received the B.S. and Ph.D. degrees in Mathematics from KAIST in 1991, and 1997, respectively. He is a professor of Mathematical Sciences at the Seoul National University (SNU). Before joining to SNU, he was in ETRI, Brown University, and ICU. He is on the editorial board of Journal of KIISC and CSI. He received the best paper award in Asiacrypt 2008. His research interests include computational number theory, cryptography, and information security.

PLACE
PHOTO
HERE

Stanislaw Jarecki is an Associate Professor of Computer Science at the University of California, Irvine. He received his Ph.D. in Computer Science from MIT in 2001. His main research interests are cryptography and fault tolerant distributed protocols. His recent and current research projects involve threshold protocols, robustness in cryptographic protocols, multisignatures, general efficient secure computation, secure computation of set intersection, and privacy in authentication schemes.

PLACE
PHOTO
HERE

Taekyoung Kwon received his B.S., M.S., and Ph.D. degrees in Computer Science from Yonsei University, Seoul, Korea, in 1992, 1995, and 1999, respectively. He was a postdoctoral fellow at the University of California, Berkeley, from 1999 to 2000, and developed a cryptographic protocol, which was later standardized by IEEE P1363.2 and ISO/IEC JTC1 SC27 11770-4, respectively. In 2001, he joined Sejong University, Seoul, Korea, where he is now an Associate Professor of Computer Engineering. He visited University of Maryland, College Park, in his sabbatical from 2007 to 2008. He is on the editorial board of Journal of KIISC. He has served many of international and domestic academic activities. His research interest includes information security, applied cryptography, network protocol, computational theory and human-computer interaction.

PLACE
PHOTO
HERE

Mun-Kyu Lee received the B.S. and M.S. degrees in Computer Engineering from Seoul National University in 1996 and 1998, respectively, and the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University in 2003. From 2003 to 2005, he was a senior engineer at Electronics and Telecommunications Research Institute, Korea. He is currently an Assistant Professor in the School of Computer and Information Engineering at Inha University, Korea. His research interests are in the areas of theory of computation, information security and cryptology.