Low-Weight Binary Representations for Pairs of Integers

Jerome A. Solinas

National Security Agency, USA

Abstract. Shamir's method speeds up the computation of the product of powers of two elements of a group, a common object in public-key algorithms. Shamir's method is based on binary expansions and was designed for modular and finite field arithmetic. Elliptic curve arithmetic uses signed binary expansions rather than the ordinary binary expansions of modular arithmetic. This note extends Shamir's method to the elliptic curve setting by specifying an optimal signed binary representation for a pair of positive integers.

1 Shamir Methods

Shamir suggested [4] a simple but powerful trick for speeding up an operation that is common in public-key cryptography.

Let $\mathfrak{G}$ be a subgroup of the multiplicative group of nonzero elements of a finite field $\mathbb{F}_q$.[1] The basic public-key operation in $\mathfrak{G}$ is *exponentiation:* computing g^a for a given element $g \in \mathfrak{G}$ and a positive integer a. This is typically accomplished [6] by the *binary method,* based on the binary expansion $\overline{\mathbf{e}}$ of a. The method requires a squaring for every bit in $\overline{\mathbf{e}}$ and a multiplication by g for every 1 appearing in $\overline{\mathbf{e}}$. If ℓ is the length of $\overline{\mathbf{e}}$, then the binary method costs $\sim \ell$ squarings and $\sim \ell/2$ general multiplications (on average).

More generally, it is commonly needed to evaluate expressions of the form

$$g^a\, h^b \quad . \tag{1}$$

In particular, most common digital signatures (apart from RSA) are verified by evaluating an expression for the form (1).

The naïve way to evaluate the expression (1) is to perform two exponentiations and multiply the results. Using the binary method,

[1] In this section, we use the multiplicative notation, since the public-key setting motivating the results is a finite subgroup of the multiplicative group of a finite field.

this approach costs $\sim 2\ell$ squarings and $\sim \ell$ general multiplications (on average).

Shamir observed that it is unnecessary to compute the two exponentials separately, as only the product is needed to verify the signature. Thus one can combine the two exponentiations into one operation, with an increase in efficiency.

The simplest way to do this is to apply the binary method to the binary expansions of both a and b at the same time. The technique is illustrated below for the computation of $g^{37}\,h^{20}$.

$37 =$	1	0	0	1	0	1
$20 =$	0	1	0	1	0	0
$\square$	1	g^2	$g^4\,h^2$	$g^8\,h^4$	$g^{18}\,h^{10}$	$g^{36}\,h^{20}$
$\times g$	g			$g^9\,h^4$		$g^{37}\,h^{20}$
$\times h$		$g^2\,h$		$g^9\,h^5$		

It is clear that this method, which we call the *simple Shamir method*, costs $\sim \ell$ squarings and $\sim \ell$ general multiplications (on average).

Shamir further observed that one obtains an additional speedup by first precomputing and storing the product gh. Then, when one encounters 1's in the same place of both binary expansions, one multiplies by gh rather than by g and h individually. Since the probability is $1/4$ that two random bits each have the value 1, this saves one fourth of the general multiplications in the simple Shamir method. The technique is illustrated below for the same example as above.

$37 =$	1	0	0	1	0	1
$20 =$	0	1	0	1	0	0
$\square$	1	g^2	$g^4\,h^2$	$g^8\,h^4$	$g^{18}\,h^{10}$	$g^{36}\,h^{20}$
$\times g$	g					$g^{37}\,h^{20}$
$\times h$		$g^2\,h$				
$\times gh$				$g^9\,h^5$		

This technique, which we call the *fast Shamir method*, costs $\sim \ell$ squarings and $\sim 3\ell/4$ general multiplications (on average). The precomputation requires one general multiplication, and thereafter one must store the result.

2 Signed Binary Expansions

Although the binary and Shamir methods work in any group for which we possess a general group law, they are not always the most efficient techniques. The simplest example of this is that of elliptic curves, where the cost of computing the inverse $-P$ of a point P is negligible compared to addition of points. More generally, we shall consider groups in which each element can be inverted very efficiently. Since the motivating example is that of elliptic curves, we shall henceforth use additive rather than multiplicative notation for the group operations. Rather than exponentiation, the operation of interest is *scalar multiplication:* given an element P and an integer a, compute aP, the result of adding a copies of P.

In this situation in which inversion is efficient, one need not restrict the coefficients of a binary expansion to 0 and 1. One can use a *signed binary expansion* (allowing coefficients 0 and ± 1) just as efficiently, and at no extra storage. This can prove advantageous if a signed binary expansion can be found which has a lower Hamming weight (*i.e.* has fewer nonzero terms) than the binary expansion for the same number.

A signed binary expansion of an integer n will be denoted

$$n = \langle u_m,\ \ldots,\ u_1,\ u_0 \rangle\ ;$$

this means that

$$n = \sum_{j=0}^{m} u_j\, 2^j$$

and

$$u_j = 0 \ \text{ or } \ \pm 1 \ \text{ for all } j.$$

Whenever a signed binary expansion appears in this paper, we shall adopt the convention that $u_{i,j} = 0$ for $j > m$.

A given nonzero integer n has infinitely many signed binary expansions, but one can specify a special kind which is optimal for scalar multiplication. Explicitly, the *Nonadjacent Form* (NAF) of an integer n is a signed binary expansion with the property that no two consecutive terms are nonzero. We now list the basic properties of the NAF (see, *e.g.* [7]).

(NAF-1) *The NAF is a convenient, well-defined representation:*

- Every integer has a NAF.
- The NAF of an integer is unique (up to initial zeroes).
- There is a simple and efficient algorithm for computing the NAF of a given integer.

(NAF-2) *The NAF is the most efficient representation:*

- The NAF of a positive integer is at most one bit longer than its binary expansion.
- The NAF of n has the minimal Hamming weight of any signed binary expansion of n.

(NAF-3) *The efficiency of the NAF is easy to calculate:*

- The average Hamming density among NAF's is 1/3.

The above properties constitute a collection of attributes that any "good" representation of integers should have.

3 Joint Signed Binary Expansions

To extend the Shamir methods to the addition-subtraction setting, one uses *joint signed binary expansions:* expressions of the form

$$\begin{array}{l} n_0 = \langle\, u_{0,m}, \quad \ldots, \quad u_{0,1}, \quad u_{0,0} \,\rangle \\ n_1 = \langle\, u_{1,m}, \quad \ldots, \quad u_{1,1}, \quad u_{1,0} \,\rangle \end{array} \tag{2}$$

in which each $u_{i,j}$ is either 0 or ± 1.

To extend the simple Shamir method, one chooses each of two expansions to be the NAF, as in the following example.

Example: the simple Shamir method for computing $11P + 20Q$

$$\begin{array}{rcccccc} 11 = & 1 & 0 & -1 & 0 & -1 \\ 20 = & 1 & 0 & 1 & 0 & 0 \\ \\ \times 2 & \mathcal{O} & 2P+2Q & 4P+4Q & 6P+10Q & 12P+20Q \\ P & P & & 3P+4Q & & 11P+20Q \\ Q & P+Q & & 3P+5Q & & \end{array}$$

This method costs $\sim \ell$ doublings and $\sim 2\ell/3$ general additions (on average). It is clear that using NAFs is the optimal way to perform the simple Shamir method, since it minimizes the number of nonzero terms in the two signed binary expansions.

Adapting the fast Shamir method to the signed binary case is not as straightforward. One obvious approach is to use again the NAF's of the two coefficients; this yields a technique which requires $\sim \ell$ doublings and $\sim 5\ell/9$ general additions (on average). Although efficient, this is not optimal. In other words, there are other joint signed binary expansions for pairs of integers that sometimes work better than the NAF's of the integers.

4 The Joint Sparse Form

The number of general additions required by the fast Shamir method is the *joint Hamming weight* of the joint signed binary expansion, *i.e.* the number of nonzero columns. To optimize the fast Shamir method, one chooses a joint signed binary expansion with minimal joint Hamming weight (and more or less minimal length).

This can be done using what we call a *Joint Sparse Form.* This is a represention (2) with the following properties:

(JS-1) Of any three consecutive positions, at least one is a double zero. In other words, for any i and positive j, we have $u_{i,j+k} = u_{1-i,j+k} = 0$ for $k = 0$ or ± 1.

(JS-2) Adjacent terms do not have opposite signs. In other words, it is never the case that $u_{i,j+1}\, u_{i,j} = -1$.

(JS-3) If $u_{i,j+1}\, u_{i,j} \neq 0$, then $u_{1-i,j+1} = \pm 1$ and $u_{1-i,j} = 0$.

It should be noted that the NAF of an integer n_1 is a special case, namely the Joint Sparse Form with $n_0 = 0$ (since the all-zero expansion is the only signed binary expansion of 0).

The following example illustrates the advantage of the Joint Sparse Form. For $n_0 = 53$ and $n_1 = 102$, the NAF's

$$\begin{aligned} 53 &= \langle\ 0,\ 1,\ \ 0, -1,\ 0,\ 1,\ \ 0,\ 1\ \rangle \\ 102 &= \langle\ 1,\ 0, -1,\ \ 0,\ 1,\ 0, -1,\ 0\ \rangle \end{aligned}$$

have joint Hamming weight 8, whereas the Joint Sparse Form

$$\begin{aligned} 53 &= \langle\ 0,\ 1,\ 0,\ \ 0, -1,\ 0, -1, -1\ \rangle \\ 102 &= \langle\ 1,\ 0,\ 0, -1, -1,\ 0, -1,\ \ 0\ \rangle \end{aligned}$$

has joint Hamming weight 6.

The following example uses a Joint Sparse Form in a signed binary version of the fast Shamir method.

<u>*Example:*</u> *the fast Shamir method for computing* $13P + 10Q$

$13 =$	1	0	0	-1	-1
$10 =$	0	1	0	1	0
$\times 2$	$\mathcal{O}$	$2P$	$4P + 2Q$	$8P + 4Q$	$14P + 10Q$
P	P				$13P + 10Q$
Q		$2P + Q$			
$P + Q$					
$P - Q$				$7P + 5Q$	

This method costs $\sim \ell$ squarings and $\sim w$ general multiplications, where ℓ is the length and w the joint Hamming weight of the expansion.

5 Uniqueness of the Joint Sparse Form

The Joint Sparse Form satisfies a set of properties analogous to (NAF-1), (NAF-2), (NAF-3). The first property to be proved is that of uniqueness. A Joint Sparse Form is *proper* if at least one of its leftmost bits are nonzero.

Theorem 1. *A pair n_0, n_1 of positive integers has at most one proper Joint Sparse Form.*

Proof. Suppose, on the contrary, that there are two distinct proper Joint Sparse Forms

$$\begin{aligned} n_0 &= \langle\, u_{0,m}, \ \ \ldots, \ \ u_{0,1}, \ \ u_{0,0} \,\rangle = \langle\, w_{0,n}, \ \ \ldots, \ \ w_{0,1}, \ w_{0,0} \,\rangle \\ n_1 &= \langle\, u_{1,m}, \ \ \ldots, \ \ u_{1,1}, \ \ u_{1,0} \,\rangle = \langle\, w_{1,n}, \ \ \ldots, \ \ w_{1,1}, \ w_{1,0} \,\rangle \ . \end{aligned}$$

Since these representations are different, then $u_{i,j} \neq w_{i,j}$ for some i and j. Let g be the minimal value of j for which the two forms disagree. For $i = 0, 1$, let

$$r_i = \langle u_{i,g-1}, \ldots, u_{i,1}, u_{i,0} \rangle = \langle w_{i,g-1}, \ldots, w_{i,1}, w_{i,0} \rangle$$

and

$$k_i = (n_i - r_i)/2^g \ .$$

Then

$$\begin{aligned} k_0 &= \langle\, u_{0,m}, \ \ \ldots, \ \ u_{0,g+1}, \ \ u_{0,g} \,\rangle = \langle\, w_{0,n}, \ \ \ldots, \ \ w_{0,g+1}, \ w_{0,g} \,\rangle \\ k_1 &= \langle\, u_{1,m}, \ \ \ldots, \ \ u_{1,g+1}, \ \ u_{1,g} \,\rangle = \langle\, w_{1,n}, \ \ \ldots, \ \ w_{1,g+1}, \ w_{1,g} \,\rangle \ . \end{aligned}$$

Since the expansions disagree at $j = g$, we may assume that

$$u_{0,g} \neq w_{0,g}$$

by exchanging the roles of n_0 and n_1 if necessary. It follows that k_0 is odd, for otherwise we would have

$$u_{0,g} = w_{0,g} = 0 \ .$$

Therefore $u_{0,g}$ and $w_{0,g}$ must have values 1 and -1. Without loss of generality, we assume that $u_{0,g} = 1$ and $w_{0,g} = -1$.

Suppose that $k_0 \equiv 1 \pmod 4$ (the case $k_0 \equiv 3 \pmod 4$ is similar). Since $u_{0,g} = 1$, then $u_{0,g+1} = 0$. Since $w_{0,g} = -1$, then $w_{0,g+1}$ must be odd. By (JS-2), then, we have $w_{0,g+1} = -1$. It follows from (JS-3) that $w_{1,g} = 0$ and $w_{1,g+1} = \pm 1$; thus $k_1 \equiv 2 \pmod 4$. It follows that $u_{1,g} = 0$ and $u_{1,g+1} = \pm 1$.

It now follows that from (JS-1) that both expansions have double zeroes at position $g + 2$. Thus

$$u_{0,g+2} = u_{1,g+2} = w_{0,g+2} = w_{1,g+2} = 0 \ .$$

But this means that $k_0 \equiv 1 \pmod 8$ since

$$\langle u_{0,g+2}, \quad u_{0,g+1}, \quad u_{0,g} \rangle = \langle 0, \quad 0, \quad 1 \rangle \ ,$$

whereas

$$\langle w_{0,g+2}, \quad w_{0,g+1}, \quad w_{0,g} \rangle = \langle 0, \quad -1, \quad -1 \rangle$$

so that $k_0 \equiv 5 \pmod 8$. This contradiction shows that the initial assumption must have been wrong. $\square$

As a result of Thm. 1, we shall speak henceforth of *the* Joint Sparse Form of two integers.

6 Calculating the Joint Sparse Form

The most straightforward way to prove the existence of the Joint Sparse Form for every pair of positive integers n_0 and n_1 is to present an algorithm for producing it.

Algorithm 1 *(Joint Sparse Form)*

Input: Nonnegative integers n_0 and n_1, not both zero

Output: An expression (2) for n_0 and n_1 in Joint Sparse Form

```
Set k_0 ← n_0 and k_1 ← n_1
Set j ← 0
Set d_0 ← 0 and d_1 ← 0
While k_0 + d_0 > 0 or k_1 + d_1 > 0 do
     Set ℓ_0 ← d_0 + k_0
     Set ℓ_1 ← d_1 + k_1
     For i from 0 to 1 do
          If ℓ_i is even
               then u ← 0
               else
                    u ← ℓ_i mods 4
                    If ℓ_i ≡ ±3 (mod 8)
                         and ℓ_{1−i} ≡ 2 (mod 4)
                         then u ← −u
          Set u_{i,j} ← u
     Next i
     For i from 0 to 1 do
          If 2d_i = 1 + u_{i,j}
               then d_i ← 1 − d_i
          Set k_i ← ⌊k_i/2⌋
     Next i
     Set j ← j + 1
EndWhile
```

The notation "mods" indicates that the modular reduction is to return the smallest residue in absolute value.

It should be noted that the only operations performed on the integers k_i and ℓ_i are evaluation modulo 8 (*i.e.* examining the three lowest-order bits) and the operation $\lfloor k_i/2 \rfloor$, which involves chopping off the rightmost bit. Thus, although it is expressed in terms of integer arithmetic, Alg. 1 is in fact performing bit operations on the entries of the binary expansions of n_i. Thus we may regard those expansions, rather than the integers n_i, as the inputs to the algorithm.

In order to prove the desired properties of the Joint Sparse Form, it is necessary to generalize Alg. 1 by allowing as inputs certain signed binary expansions.

We say that the signed binary expansion

$$n = \langle u_m, \ \ldots, \ u_1, u_0 \rangle \tag{3}$$

is *reduced* if $u_{j-1}u_j \neq -1$ for all $j > 0$. A joint signed binary expansion (2) is said to be *reduced* if it consists of two reduced signed binary expansions. (Thus a Joint Sparse Form is reduced by condition (JS-2).)

Algorithm 2 *(Joint Sparse Form)*

Input: A reduced joint signed binary expansion

$$\begin{array}{l} n_0 = \langle\, e_{0,m-1}, \quad \ldots, \quad e_{0,2}, \quad e_{0,1}, \quad e_{0,0} \,\rangle \\ n_1 = \langle\, e_{1,m-1}, \quad \ldots, \quad e_{1,2}, \quad e_{1,1}, \quad e_{1,0} \,\rangle \end{array} \tag{4}$$

Output: An expression (2) for n_0 and n_1 in Joint Sparse Form

`Set` $a_0 \leftarrow e_{0,0}$, $b_0 \leftarrow e_{0,1}$, $c_0 \leftarrow e_{0,2}$, $d_0 \leftarrow 0$
`Set` $a_1 \leftarrow e_{1,0}$, $b_1 \leftarrow e_{1,1}$, $c_1 \leftarrow e_{1,2}$, $d_1 \leftarrow 0$
`For` j `from` 0 `to` m `do`
    `For` i `from` 0 `to` 1 `do`
        `If` $d_i \equiv a_i \pmod 2$
            `then set` $u \leftarrow 0$
            `else`
                `Set` $u \leftarrow (d_i + 2b_i + a_i) \text{ mods } 4$
                `If` $d_i + 4c_i + 2b_i + a_i \equiv \pm 3 \pmod 8$
                    `and` $d_{1-i} + 2b_{1-i} + a_{1-i} \equiv 2 \pmod 4$
                    `then set` $u \leftarrow -u$
        `Set` $u_{i,j} \leftarrow u$
        `Set` $\mathcal{R}_{i,j} \leftarrow (d_i, c_i, b_i, a_i)$
    `Next` i
    `Set` $\mathcal{S}_j \leftarrow (\mathcal{R}_{0,j}, \mathcal{R}_{1,j})$
    `For` i `from` 0 `to` 1 `do`
        `Set` $d_i \leftarrow (d_i + a_i - u_{i,j})/2$
            $a_i \leftarrow b_i$
            $b_i \leftarrow c_i$
            $c_i \leftarrow e_{i,j+3}$
    `Next` i
`Next` j

It should be remembered that, by convention, $e_{i,j} = 0$ for $j \geq m$. The quantities $\mathcal{R}_{i,j}$ and $\mathcal{S}_j$ do not figure in the algorithm; they are included solely to establish notation for the proofs to follow.

It is easy to check that, in the special case in which the $e_{i,j}$'s are "ordinary" unsigned bits, Alg. 2 is equivalent to Alg. 1.

7 Correctness of the Algorithms

Our next task is to prove that Alg. 2 (and so Alg. 1) actually produces the Joint Sparse Form of its input. In order to do this, we examine more closely the operation of the algorithm.

We call the vectors $\mathcal{S}_j$ the *states* of the algorithm. The output vector $(u_{0,j}, u_{1,j})$ is a function of the state $\mathcal{S}_j$. Thus we may describe the action of Alg. 2 as follows: the j^{th} iteration of the `Do` loop inputs the state $\mathcal{S}_{j-1}$, outputs $(u_{0,j-1}, u_{1,j-1})$, and changes the state to $\mathcal{S}_j$.

We next enumerate the possible values for the states. To begin with, there are 51 possible values[2] for each $\mathcal{R}_{i,j} = (d_i, c_i, b_i, a_i)$. For each $\mathcal{R}_{i,j}$, we define

$$k_{i,j} := 4c_i + 2b_i + a_i$$
$$\ell_{i,j} := d_i + k_i \quad .$$

These values agree with those used in Alg. 1 in the cases where that algorithm applies.

We next define a set of eight states for $\mathcal{S}_j$ in terms of $\ell_{i,j}$.

$\mathcal{S}_j$	$\mathcal{R}_{0,j}$	$\mathcal{R}_{1,j}$
C	$\ell_{i,j}$ even	$\ell_{i,j}$ even
D	$\ell_{i,j} \equiv 0 \pmod 4$	$\ell_{i,j}$ odd
E	$\ell_{i,j} \equiv 2 \pmod 4$	$\ell_{i,j} \equiv \pm 1 \pmod 8$
F	$\ell_{i,j} \equiv 2 \pmod 4$	$\ell_{i,j} \equiv \pm 3 \pmod 8$
G	$\ell_{i,j}$ odd	$\ell_{i,j} \equiv 0 \pmod 4$
H	$\ell_{i,j} \equiv \pm 1 \pmod 8$	$\ell_{i,j} \equiv 2 \pmod 4$
J	$\ell_{i,j} \equiv \pm 3 \pmod 8$	$\ell_{i,j} \equiv 2 \pmod 4$
K	$\ell_{i,j}$ odd	$\ell_{i,j}$ odd

[2] The reason there are not $3^4 = 81$ possibilities is that some combinations of a_i, b_i, c_i do not occur since the sequences $e_{i,j}$ are reduced.

We next discuss which states can follow which. It is straightforward to verify the following by checking each of the 51 cases.

- If $\ell_{i,j} \equiv 0 \pmod 4$ and $\ell_{1-i,j}$ is odd, then $\ell_{i,1+j}$ is even.
- If $\ell_{i,j} \equiv 2 \pmod 4$ and $\ell_{1-i,j}$ is odd, then $\ell_{i,1+j}$ is odd.
- If $\ell_{i,j}$ is odd and $\ell_{1-i,j} \not\equiv 2 \pmod 4$, then $\ell_{i,1+j}$ is even.
- If $\ell_{i,j} \equiv 1 \pmod 4$ and $\ell_{1-i,j} \equiv 2 \pmod 4$, then $\ell_{i,1+j} \equiv 0 \pmod 4$.
- If $\ell_{i,j} \equiv 3 \pmod 4$ and $\ell_{1-i,j} \equiv 2 \pmod 4$, then $\ell_{i,1+j}$ is odd.

As a result, we have the following values for $\mathcal{S}_{j+1}$ for each $\mathcal{S}_j$. (The possible outputs for each state are also given for future convenience.)

$\mathcal{S}_j$	$u_{0,j}$	$u_{1,j}$	$\mathcal{S}_{j+1}$
C	0	0	$*$
D	0	± 1	C
E	0	± 1	G
F	0	± 1	K
G	± 1	0	C
H	± 1	0	D
J	± 1	0	K
K	± 1	± 1	C

The entry $*$ indicates that any state is a possible successor to state C. In every other case, the state $\mathcal{S}_{j+1}$ is determined by $\mathcal{S}_j$.

Finally, it will be useful to further combine the above eight states into three, as follows.

$\mathcal{S}_j$		$u_{0,j} = u_{1,j} = 0$?	$\mathcal{S}_{j+1}$
A	E, F, H, or J	No	B
B	D, G, or K	No	C
C	C	Yes	A, B, or C

Theorem 2. *Alg. 1 always outputs the Joint Sparse Form of its inputs.*

Proof. It is straightforward to verify that the expansion (2) produced by the algorithm is in fact a joint signed binary expansion of n_0 and n_1. It remains to prove that this expansion satisfies properties (JS-1), (JS-2), (JS-3).

(JS-1): This condition is equivalent to the assertion that, for every j, at least one of $\mathcal{S}_{j-1}$, $\mathcal{S}_j$, $\mathcal{S}_{j+1}$ is in state C. Suppose that $\mathcal{S}_{j-1}$ is not in state C; then its state is either B (so that $\mathcal{S}_j$ is in state C) or A (in which case $\mathcal{S}_{j+1}$ is in state C).

(JS-2): Suppose that $u_{0,j} \neq 0$ and $u_{0,j+1} \neq 0$. Then it follows from the above table that $\mathcal{S}_j$ is in state J and that $\mathcal{S}_{j+1}$ is in state K. It is straightforward to compute $u_{0,j+1}$ and to verify that it equals $u_{0,j}$ in each of the possible cases for $\mathcal{R}_{0,j}$ that comprise state J. The analogous procedure obviously works for $u_{1,j}$ and $u_{1,j+1}$.

(JS-3): Suppose that $u_{0,j}\, u_{0,j+1} \neq 0$, so that $\mathcal{S}_j$ is in state J and that $\mathcal{S}_{j+1}$ is in state K. Then the above table shows that $u_{1,j} = 0$ and $u_{1,j+1} = \pm 1$. The case $u_{1,j}\, u_{1,j+1} \neq 0$ is similar. □

8 Optimality of the Joint Sparse Form

Our next task is to prove that the Joint Sparse Form has minimal joint Hamming weight among all joint signed binary expansions. Our approach will be to prove that the joint Hamming weight of the output of Alg. 2 is less than or equal to that of the input.

Let (4) be the input to Alg. 2 and let (2) be the output. For each k, define $\mathcal{W}_k$ to be pair of sequences

$$\begin{array}{l} (\ w_{0,m}, \ \ \ldots, \ \ w_{0,1}, \ \ w_{0,0}\ ) \\ (\ w_{1,m}, \ \ \ldots, \ \ w_{1,1}, \ \ w_{1,0}\ ) \end{array}$$

where

$$w_{i,j} := \begin{cases} u_{i,j} & \text{for } j < k \\ e_{i,j} & \text{for } j \geq k. \end{cases}$$

Then $\mathcal{W}_0$ is the input sequence and $\mathcal{W}_{m+1}$ is the output sequence. Step j of Alg. 2 replaces $\mathcal{W}_j$ with $\mathcal{W}_{j+1}$.

Now, if $e_{0,j}$ or $e_{1,j}$ is nonzero and $u_{0,j} = u_{1,j} = 0$, then the joint Hamming weight of $\mathcal{W}_{j+1}$ is one less than that of $\mathcal{W}_j$. Thus we say that there is a *weight loss* at step j. Similarly, if $u_{0,j}$ or $u_{1,j}$ is nonzero and $e_{0,j} = e_{1,j} = 0$, then the joint Hamming weight of $\mathcal{W}_{j+1}$ is one more than that of $\mathcal{W}_j$; this is a *weight gain* at step j. To prove that Alg. 2 always produces an output whose joint Hamming weight is less than or equal to that of its input, we must show that the number of weight losses is at least that of the weight gains.

To prove this, we must devise a set of states that enables us to describe, in terms of the transitions between the states, when weight losses and gains occur. We begin with the following groupings of the 51 cases for $\mathcal{R}_{i,j}$.

(a)	$\ell_{i,j}$ odd, $d = 0$
(b)	$\ell_{i,j}$ odd, $d = \pm 1$, $b = 0$
(c)	$\ell_{i,j}$ odd, $d = \pm 1$, $b = \pm 1$
(d)	$\ell_{i,j}$ even, $d = 0$
(e)	$\ell_{i,j} \equiv 2 \pmod 4$, $d = \pm 1$, $b = 0$
(f)	$\ell_{i,j}$ even, other cases

The groupings can be combined as follows to form a new set of states.

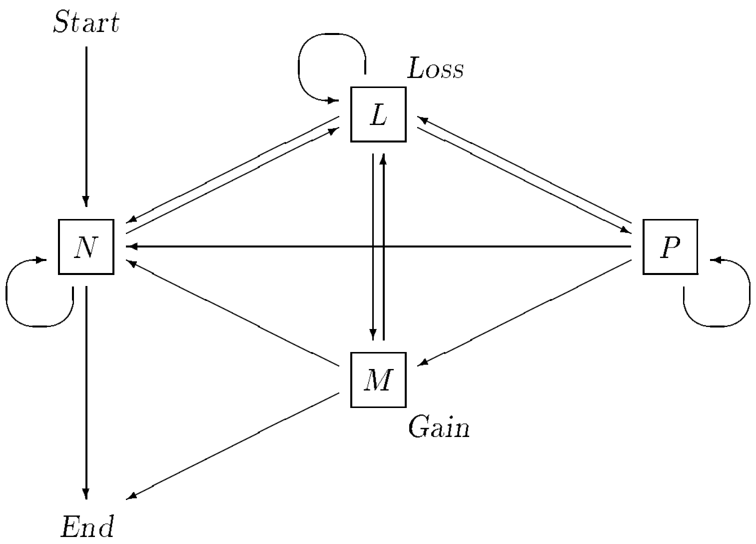
$\mathcal{S}_j$	$\mathcal{R}_{0,j}$ and $\mathcal{R}_{1,j}$	$\mathcal{S}_{j+1}$
L	both (d), (e), or (f); not both (d)	L, M, N, or P
M	both (b), (c), or (d); not both (d)	L or N
N	one (a) and the other (a), (c), (d), or (f); or both (d)	L or N
P	other 12 cases	L, M, N, or P

As before, it can be verified on a case-by-case basis that each case comprising states M and N is followed by a case of states L or N. It can also be verified that state L consists of precisely those cases in which there is a weight loss, and state M of the cases with a weight gain. There is neither in states N and P.

Since each d_i is initially 0 in Alg. 2, each $\mathcal{R}_{i,j}$ begins in groupings (a) or (d). It follows that the first state is always N.

At the final step of the algorithm, each a_i, b_i, and c_i are 0. Thus the final values of $\mathcal{R}_{i,j}$ are in groupings (b) or (d); thus the final state is either M or N.

It follows that the flow of Alg. 2 through the four states is as follows.



Lemma 1. *The joint Hamming weight of the output of Alg. 2 is no greater than that of the input.*

Proof. It is clear from the diagram that every visit to state M must be preceeded by a separate visit to state L. Thus the total number of weight losses is at least the total number of weight gains. □

Theorem 3. *The joint Hamming weight of any joint signed binary expansion of the positive integers n_0 and n_1 is greater than or equal to that of the Joint Sparse Form of n_0 and n_1.*

Proof. Given any signed binary expansion for n, one can *reduce* by replacing adjacent coefficients $\langle 1,\ -1\rangle$ by $\langle 0,\ \ 1\rangle$ and $\langle -1,\ \ 1\rangle$ by $\langle 0,\ -1\rangle$. In this way, one produces a reduced expansion for n whose nonzero positions are a subset of those of the original expansion. Thus any joint signed binary expansion can be reduced to produce a reduced expansion whose joint Hamming weight is less than or equal to that of the original expansion.

Applying Alg. 2 to the reduced expansion, one obtains a Joint Sparse Form. By Lemma 1, the joint Hamming weight of this Joint Sparse Form is no greater than that of the reduced expansion; thus it is no greater than that of the original expansion. □

9 Efficiency of the Fast Shamir Addition-Subtraction Method

To determine the number of elliptic operations required by Shamir's method, it is necessary to compute the length and average joint Hamming weight of the Joint Sparse Form. (The length gives the number of doubles required, and the density then determines the number of general additions.)

If the signed binary expansion (3) is reduced, then it is easy to see that

$$\lfloor \log_2 n \rfloor \le m \le \lceil \log_2 n \rceil \ .$$

It follows that a reduced signed binary expansion is at most one bit longer than the ordinary binary expansion. As a result, the Joint Sparse Form is at most one bit longer than the binary expansion of the larger of the two integers.

It remains to estimate the joint Hamming density of the Joint Sparse Form.

Theorem 4. *The average joint Hamming density among Joint Sparse Form representations is* $1/2$.

Proof. For each $j \ge 0$, define $\mathrm{Prob}_j(A)$ to be the probability of $\mathcal{S}_j$ being in state A, for inputs n_0 and n_1 taken randomly from the nonnegative integers less than 2^m for some $m \ge j+3$. (Since the outputs $u_{0,j}$ and $u_{1,j}$ of Alg. 2 are independent of the inputs $e_{0,k}$ and $e_{1,k}$ for $k \ge j+3$, the quantity $\mathrm{Prob}_j(A)$ does not depend on m.) The probabilities $\mathrm{Prob}_j(B)$ and $\mathrm{Prob}_j(C)$ are defined analogously.

To accomodate the restriction $m \geq j+3$, we must work with the *adjusted* joint Hamming density rather than the true joint Hamming density. The adjusted joint Hamming density is simply the density over all but the three leftmost places. Asymptotically, the omission of three places does not matter.

Since $u_{0,j} = u_{1,j} = 0$ is the output for state C and for neither of the other states, the average adjusted joint Hamming density for inputs less than 2^m is given by

$$\mathcal{L}_m = 1 - \frac{1}{m-2} \sum_{j=0}^{m-3} \mathrm{Prob}_j(C) \ .$$

The overall average joint Hamming density is then

$$\mathcal{L} = \lim_{m \to \infty} \mathcal{L}_m \ .$$

To evaluate the numbers $\mathcal{L}_m$, we develop a recursion. It was seen in §7 that if $\mathcal{S}_{j-1}$ has state A, then $\mathcal{S}_j$ has state B, and that if $\mathcal{S}_{j-1}$ has state B, then $\mathcal{S}_j$ has state C. If $\mathcal{S}_{j-1}$ has state C, then a simple counting argument shows that

$$\begin{array}{ll}
\mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{C}\Big) = 1/4 & \mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{G}\Big) = 1/8 \\
\mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{D}\Big) = 1/8 & \mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{H}\Big) = 1/16 \\
\mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{E}\Big) = 1/16 & \mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{J}\Big) = 1/16 \\
\mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{F}\Big) = 1/16 & \mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{K}\Big) = 1/4,
\end{array}$$

so that

$$\mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{A}\Big) = 1/4 \qquad \mathrm{Prob}\Big(\mathrm{state}(\mathcal{S}_j) = \mathrm{B}\Big) = 1/2.$$

It follows that

$$\begin{aligned}
\mathrm{Prob}_{j+1}(A) &= \frac{1}{4}\,\mathrm{Prob}_j(C) \\
\mathrm{Prob}_{j+1}(B) &= \frac{1}{2}\,\mathrm{Prob}_j(C) + \mathrm{Prob}_j(A) \\
\mathrm{Prob}_{j+1}(C) &= \frac{1}{4}\,\mathrm{Prob}_j(C) + \mathrm{Prob}_j(B) \ .
\end{aligned}$$

We now rewrite the last expression for $\mathrm{Prob}_{j+1}(C)$, using the fact that

$$\mathrm{Prob}_j(A) + \mathrm{Prob}_j(B) + \mathrm{Prob}_j(C) = 1 \ ,$$

and obtain the recursion

$$\mathrm{Prob}_{j+1}(C) = 1 - \frac{3}{4}\,\mathrm{Prob}_j(C) - \frac{1}{4}\,\mathrm{Prob}_{j-1}(C) \ .$$

The initial values for this recursion follow from the values

$$\begin{aligned} \mathrm{Prob}_0(A) &= 1/4 \\ \mathrm{Prob}_0(B) &= 1/2 \\ \mathrm{Prob}_0(C) &= 1/4 \ , \end{aligned}$$

which can be verified by a simple counting. The value $\mathrm{Prob}_1(C) = 9/16$ follows at once.

With these initial values, the general solution to the recursion is given by

$$\mathrm{Prob}_j(C) = \frac{1}{2} + \frac{1}{2^j\sqrt{14}}\,\cos(j\theta - \phi)$$

where

$$\theta = \cos^{-1}\left(-\frac{3}{4}\right) \ , \qquad \phi = \sin^{-1}\left(\frac{1}{\sqrt{8}}\right) \ .$$

It is now a simple matter to verify that

$$\mathcal{L}_m = \frac{1}{2} + O(2^{-m})$$

so that $\mathcal{L} = 1/2$. □

It follows from Thm. 4 that the average cost of the fast Shamir addition-subtraction method is $\sim \ell$ doubles and $\sim \ell/2$ general additions, or the cost of a single scalar multiplication via the binary method. This compares with $\sim 2\ell$ doubles and $\sim 2\ell/3$ additions for the naive method. The precise savings depends on the relative costs of doubles and general additions.

10 Applications to Elliptic Scalar Multiplication

We now specialize to the usual case of interest, namely elliptic curves over finite fields. Aside from the intended application, namely the computation of the quantity $aP + bQ$, there are various situations in which the Shamir method can aid in carrying out a single scalar multiplication.

The simplest approach is this. Suppose that the order r of the private key space (*i.e.* the large prime dividing the number of points on the curve) is less than $2^{2\ell}$. To take a 2ℓ-bit multiple n of a point P, one proceeds as follows.

1. Compute $Q := 2^{\ell} P$.
2. Write $n = a + b \cdot 2^{\ell}$, where a and b are ℓ-bit integers.
3. Compute $nP = aP + bQ$ via the elliptic Shamir method.

This procedure requires $\sim 2\ell$ elliptic doubles and $\sim \ell/2$ elliptic additions, a savings of $\sim \ell/6$ elliptic additions over the addition-subtraction method. This savings will sometimes be too small to justify the extra memory needed to store three extra points. However, a more significant savings occurs if the point P is to be reused.[3] In this case, step 1 above can be carried out one time and Q can be stored off. Each subsequent scalar multiplication of P will then cost only $\sim \ell$ elliptic doubles and $\sim \ell/2$ elliptic additions. In addition to the extra stored point Q, it is necessary to access the two points $P \pm Q$ during the scalar multiplication process. These two points can be precomputed and stored, or computed anew each time, depending on the relative cost of computation and memory.

An important variant of this technique is given in [5]. In this approach, the curve is chosen to have complex multiplication by a ring of small discriminant. This enables one to choose $Q = \alpha P$, where α is a fixed complex ring element such that multiplication by α is inexpensive. Such a Q need not be stored, but rather computed from P whenever needed. There is now an extra step in each scalar multiplication, for the equation $n = a + b \cdot 2^{\ell}$ must be replaced by

[3] This is a common occurrence. Examples include the base point of the curve, public keys of certifying authorities, and public keys of frequently contacted users.

$$a + b\alpha := n \bmod \rho \tag{5}$$

where ρ is a fixed complex integer of norm r. (This step can be omitted when one is generating an ephemeral key pair, since one can generate random a and b directly.) The process is optimized in the case of so-called compact curves (see [1], [2]).

In a few special cases, one can avoid the modular reduction step (5) entirely. The simplest example is the curve E given by

$$y^2 = x^3 - 2$$

over $\mathbb{F}_p$, where p is the prime $2^{390} + 3$. The number of $\mathbb{F}_p$-rational points on this curve is $n = 2^{390} - 2^{195} + 7$; this number has the form $n = 63r$, where r is a prime of approximately 384 bits. (Thus the order-r subgroup $\mathfrak{G}$ of $E(\mathbb{F}_p)$ is similar in cryptographic strength to the curve P-384 presented in [8].) Let $\lambda := (2^{195} - 2)/3$, and define $\omega \in \mathbb{F}_p$ by

$$\omega \equiv 2^{389} + 2^{194} + 1 \pmod p \ ;$$

then $\lambda(x, y) = (\omega x, y)$ for all $(x, y) \in \mathfrak{G}$. It follows that

$$(2^{195} \cdot a + b)(x, y) = (2a + b)(x, y) + 3a \left((2^{389} + 2^{194} + 1)\, x, y \right)$$

for $(x, y) \in \mathfrak{G}$.

It follows that an element of $\mathfrak{G}$ can be multiplied by an arbitrary positive $n < r$ with only about 195 elliptic doublings and 97.5 general additions, using the Joint Sparse Form. Since a general 384-bit scalar multiply requires $\sim$ 384 doublings and $\sim$ 128 additions, the above technique saves about 3/7 the operations.[4] Note that the partition of n into a and b is accomplished without any operation of the sort given in (5). The only additional memory required is that needed[5] to store $2^{389} + 2^{194} + 1$.

[4] It is also worth noting that the arithmetic modulo p can be speeded considerably by using the techniques of [3].

[5] A more space-efficient choice is to store 2^{194} only. From this one can obtain $2^{389} + 2^{194} + 1$ when needed via a single modular squaring and a few additions. Or one could store 2^{97} and perform two modular squarings for each elliptic scalar multiplication; and so on.

11 Further Work

There are many directions in which the elliptic Shamir method might be extended or improved. Several of these are listed below.

1: *A left-to-right algorithm.* The Shamir methods always go left-to-right across the binary expansions of its inputs. Alg. 1, however, is a right-to-left algorithm. This means that the two methods cannot be performed in tandem, since the derivation of the Joint Sparse Form must be completed before the elliptic Shamir method can begin. This suggests that it would be useful to devise a left-to-right algorithm to calculate the Joint Sparse Form.

Unfortunately, it is not hard to see that no algorithm of the type of Alg. 1 can exist. More precisely, Alg. 1 works by sliding a "window" of width 3 across the expansion. This would not work going left to right; indeed, no window of bounded width can work. This can be seen even in the special case of the NAF of one integer: the integers $\lfloor 2^{2m+1}/3 \rfloor$ and $\lceil 2^{2m+1}/3 \rceil$ have NAF's of different lengths, yet their binary expansions differ only in the rightmost bit. There is, however, an alternative to the NAF which has the same Hamming weight and which can be obtained by a left-to-right algorithm. It may be the case that a similar alternative exists to the Joint Sparse Form.

2: *Three or more Terms.*

The most obvious direction for generalization of the Shamir methods is that of evaluating a combination of three or more terms. In the nonelliptic (unsigned) case, the generalization is immediate. The method, however, is not very practical beyond two terms, both because of diminishing returns in efficiency and because of rapidly increasing memory requirements to store all of the combinations of the base terms.

In the elliptic case, the generalization is not at all obvious, since it requires a higher-order analogue of the Joint Sparse Form. Although the memory requirements are even worse in this case (since both sums and differences of base points must be stored), it would still be of theoretical interest to generalize the Joint Sparse construction to three or more terms.

It should be remarked that one can construct "hybrid" higher-order methods that are more practical than a pure higher-order Shamir method. For example, one might write an m-bit scalar multiplication as the sum of four ℓ-bit scalar multiplications, where $\ell = m/4$. One could then transform the first two bit strings into Joint Sparse Form, and do the same with the last two. One could then carry out a simple-Shamir-like combination of the two Joint Sparse expansions. This would yield a method requiring ℓ elliptic doubles and ℓ elliptic additions, provided three auxiliary base points had been precomputed and stored off. In addition to that, four other points would need to be computed and kept through the operation. This method would be useful in situations such as evaluating $aP + bQ$ on a compact curve. There is a large number of hybrid constructions possible, with different numbers of terms and different memory requirements.

3: *Larger coefficients.*
Another obvious generalization is to allow coefficients other than ± 1. In the nonelliptic (unsigned) case, every positive integer has a nonadjacent binary expansion with coefficients 1 and 3; such an expansion has average Hamming density 1/3. It would be interesting to see how two such expansions could be put together to create an analogue of the Joint Sparse Form. In the elliptic case, one can allow ± 1 and ± 3 to obtain an average Hamming density of 1/4. Again, it would be of interest to construct the Joint Sparse analogue.

References

1. E. Brown, B. Myers, and J. Solinas, "Elliptic curves with compact parameters," to appear.
2. E. Brown, B. Myers, and J. Solinas, "Hyperelliptic curves with compact parameters," to appear.
3. R. Crandall, *Method and apparatus for public key exchange in a cryptographic system,* U.S. Patent # 5,159,632, Oct. 27, 1992.
4. T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory* **IT-31** (1985), pp. 469-472.

5. R. Gallant, R. Lambert, and S. Vanstone, *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms,* Proc. CRYPTO 2001, to appear.
6. D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms,* 2nd ed., Addison-Wesley, 1981.
7. F. Morain & J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains," *Inform. Theor. Appl.* **24** (1990), pp. 531–543.
8. National Institute of Standards and Technology, FIPS PUB 186-2, *Digital Signature Standard (DSS),* Appendix 6, "Recommended Elliptic Curves for Federal Government Use,"
`http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf`