Elliptic curve secp256k1
Implementation notes

Todor Milev[*]
todor@fa.biz

April 2018

1 Introdution

Public/private key cryptography is arguably the most important aspect of modern crypto-currency systems. The somewhat slow execution of private/public key cryptography algorithms appears to be one of the main bottlenecks of FAB's Kanban system.

Following Bitcoin, FAB coin uses the standard public/private key cryptography ECDSA over **secp256k1**. Here, ECDSA stands for Elliptic Curve Digital Signature Algorithm and **secp256k1** stands for the elliptic curve:

$$y^2 = x^3 + 7$$

(we specify the base point later), over the finite field:

$$\mathbb{Z}/p\mathbb{Z},$$

where

$$p = 2^{256} - 2^{32} - 977. \tag{1}$$

In this document, we discuss and document technical details of FAB's implementation of ECDSA over **secp256k1**. Our openCL implementation is based on the project [1], which is in turn based on the C project libsecp256k1 [2].

2 Operations in $\mathbb{Z}/p\mathbb{Z}$

Recall from (1) that p is the prime given by

$$p = 2^{256} - 2^{32} - 977.$$

In this section, we describe our implementation of $Z/p\mathbb{Z}$.

[*]FA Enterprise System

2.1 Representations of numbers

A number in x in $\mathbb{Z}/p\mathbb{Z}$ is represented by a large integer X, in turn represented by a sequence of 10 small integers $x_0, \ldots, x_9$ for which $0 \leq x_i < 2^{32}$ and such that

$$X = \sum_{i=0}^{9} x_i \left(2^{26}\right)^i .$$

The representations of x is not unique but becomes so when we request that

$$0 \leq x_i < 2^{26}$$

and

$$0 \leq X < p = 2^{256} - 2^{32} - 977.$$

We say that the unique representation $x_0, \ldots, x_9$ of x above is its *normal form* (and x is *normalized*). Two elements of $\mathbb{Z}/p\mathbb{Z}$ are equal if and only if their normal forms are equal. We will not assume that a number x is represented by its normal form as some of the operations described below do not require that.

In what follows, we shall use the notation

$$a' = a \ \mathbf{mod}\ q$$

to denote remainder $0 \leq a' < q$ of a when dividing a by q. For the rest of this section, we also set

$$d = 2^{26}.$$

In this way the normal form of a large number X is its d-base representation (here d is the "digit" of the base).

2.2 Computing the normal form of x

2.3 Multiplying two elements

Let a be an element represented by $A = \sum a_i d^i$ with $a_0, \ldots, a_8 < 2^{30}$ and $a_9 < 2^{22}$. Likewise, let b be represented by $B = \sum b_j d^j$ with analogous inequalities on the coefficients b_j. In this section, we show how to compute the normal form of $a \cdot b$. This operation is implemented in the function `ECMultiplyFieldElementsInner`.

Compute as follows.

$$\begin{aligned} A \cdot B &= \left(\sum_{i=0}^{9} a_i d^i\right)\left(\sum_{j=0}^{9} b_j d^j\right) \\ &= \sum_{k=0}^{18} \left(\underbrace{\sum_{i=0}^{k} a_i b_{k-i}}_{=\bar{t}_k} \right) d^k \\ &= \sum_{k=0}^{18} \bar{t}^k d^k, \end{aligned}$$

where we have set $\bar{t}_k = \left(\sum_{i=0}^{k} a_i b_{k-i} \right)$. Since $0 \leq a_i, b_i < d$, the $\bar{t}_k$'s are smaller than 2^{64} and can be computed by standard arithmetic using the `uint64_t` integer type of the GPU.

Let

$$A \cdot B = \sum_{k=0}^{19} t_k d^k$$

with $0 \leq t_k < d$ be the unique representation of $A \cdot B$ base d. Then the t_k can be computed from the $\bar{t}_k$'s consecutively via

$$\begin{array}{rcl} t_0 & = & \bar{t}_0 \ \mathbf{mod}\ d \\ c_0 & = & \left\lfloor \dfrac{\bar{t}_0}{d} \right\rfloor \\ \hline t_1 & = & c_0 + \bar{t}_0 \ \mathbf{mod}\ d \\ c_1 & = & \left\lfloor \dfrac{c_0 + \bar{t}_0}{d} \right\rfloor \\ t_2 & = & c_1 + \bar{t}_1 \ \mathbf{mod}\ d \\ c_2 & = & \left\lfloor \dfrac{c_1 + \bar{t}_1}{d} \right\rfloor \\ & \vdots & \\ t_{18} & = & c_{17} + \bar{t}_{17} \ \mathbf{mod}\ d \\ c_{18} & = & \left\lfloor \dfrac{c_{17} + \bar{t}_{17}}{d} \right\rfloor \\ \hline t_{19} & = & c_{18}. \end{array}$$

In the above, all computations except the first and the last are similar, as indicated by the horizontal line, and $\lfloor \bullet \rfloor$ stands as usual for the floor function. Both **mod** d and division by d are carried out by bit-shift operations and are therefore fast.

Let $S = (A \cdot B \ \mathbf{mod}\ p)$; computing the normal form $S = \sum s_i d^i$, $0 \leq s_i < d$, $S < p$ is the final aim of the present discussion. Since $2^{256} \ \mathbf{mod}\ p = 2^{32} + 977$, it follows that

$$\begin{aligned}
t_0 + t_{10}d^{10} \ \mathbf{mod}\ p &= t_0 + t_{10}2^{26\cdot 10} && \mathbf{mod}\ p \\
&= t_0 + t_{10}\cdot 2^4 \cdot 2^{256} && \mathbf{mod}\ p \\
&= t_0 + t_{10}2^4\left(2^{32}+977\right) && \mathbf{mod}\ p \\
&= \left(t_0 + t_{10}2^4\cdot 977\right) + t_{10}2^{10}\cdot d && \mathbf{mod}\ p \\
\text{set } g_0 = t_0 + t_{10}2^4\cdot 977 \ \mathbf{mod}\ d & \\
\text{set } f_0 = \left\lfloor \frac{t_0 + t_{10}2^4\cdot 977}{d} \right\rfloor & \\
&= g_0 + \left(\underbrace{f_0 + t_{10}2^{10}}_{=h_0}\right) d && \mathbf{mod}\ p.
\end{aligned} \tag{2}$$

Set $h_0 = f_0 + t_{10}2^{10}$ as indicated above. The computation above shows how "reduce" the d-digit t_{10} by modifying the two least significant d-digits. Accounting for the "carry-over" digit h_0, we can continue with this process for the next pair of digits $t_1 d + t_{11}d^{11}$, and so on. This is done below (similar steps have been omitted).

$$\begin{aligned}
h_0 d + t_1 d + t_{11}d^{11} \ \mathbf{mod}\ p &= \left(h_0 + t_1 + t_{11}d^{10}\right) d && \mathbf{mod}\ p \\
&= \ldots \text{compute as in (2)} \ldots \\
\text{set } g_1 = h_0 + t_1 + t_{11}2^4\cdot 977 \ \mathbf{mod}\ d & \\
\text{set } f_1 = \left\lfloor \frac{h_0 + t_1 + t_{11}2^4\cdot 977}{d} \right\rfloor & \\
&= g_1 d + \left(\underbrace{f_1 + t_{11}2^{10}}_{=h_1}\right) d^2 && \mathbf{mod}\ p \\
&\ \vdots \\
h_8 d^9 + t_9 d^9 + t_{19}d^{19} \ \mathbf{mod}\ p &= \ldots \\
\text{set } g_9 = h_8 + t_9 + t_{19}2^4\cdot 977 \ \mathbf{mod}\ d & \\
\text{set } f_9 = \left\lfloor \frac{h_8 + t_9 + t_{19}2^4\cdot 977}{d} \right\rfloor & \\
&= g_9 d^9 + \left(\underbrace{f_9 + t_{19}2^{10}}_{=h_9}\right) d^{10} && \mathbf{mod}\ p.
\end{aligned}$$

In the computations above, t_{19} is maximum 26 bits long and so

$$h_9 < f_9 + 2^{26}2^{10} < 2^{37}. \tag{3}$$

In order to obtain the normal form of S, we need to modify the digits g_9 and

h_9 as follows.

$$\begin{array}{rcll}
g_9 d^9 + h_9 d^{10} \ \mathbf{mod}\ p & = & (g_9 + h_9 d) d^9 & \mathbf{mod}\ p \\
\text{set } r = g_9 + h_9 d \ \mathbf{mod}\ 2^{22} & & & \\
\text{set } m = \left\lfloor \dfrac{g_9 + h_9 d}{2^{22}} \right\rfloor & & & \\
 & = & \left(r + m \cdot 2^{22}\right) d^9 & \mathbf{mod}\ p \\
 & = & r d^9 + m 2^{256} & \mathbf{mod}\ p \\
 & = & r d^9 + m \left(2^{32} + 977\right) & \mathbf{mod}\ p.
\end{array}$$

From inequality (3), we get that $h_9 d < 2^{37} 2^{26} = 2^{63}$, and so the computations above fit in `uint64_t` type. Again using (3) we get the following estimates for the sizes of digits.

$$\begin{array}{rcl}
g_9 + h_9 d & < & 2^{64} \\
m = \left\lfloor \dfrac{g_9 + h_9 d}{2^{22}} \right\rfloor & < & \dfrac{2^{64}}{2^{22}} = 2^{42} \\
m \left(2^{32} + 977\right) & < & 2^{33} \cdot 2^{42} = 2^{75} < d^3.
\end{array}$$

Thus the number $m \left(2^{32} + 977\right)$ can be written in the form $z_0 + z_1 d + z_2 d^2$ with $0 \leq z_0, z_1, z_2 < d$. Collecting the information so far, we get

$$A \cdot B \ \mathbf{mod}\ p = (z_0 + g_0) + (z_1 + g_1) d + (z_2 + g_2) d^2 + \sum_{k=3}^{8} g_k d^k + r d^9, \qquad (4)$$

where $z_0 + g_0 < 2d$, $z_1 + g_1 < 2d$, $z_2 + g_2 < 2d$, $g_k < d$, $r < 2^{22}$. Except in the cases when $z_i + g_i \geq d$, this expression is normalized. To ensure our result is normalized, we need to reduce the representation above according to Section 2.2. This does not need to be done immediately as the form (4) satisfies all assumptions of the present discussion.

References

[1] Author: `https://github.com/hhanh00`. `https://github.com/hhanh00/secp256k1-cl` (project secp256k1-cl). 2014.

[2] Pieter Wuille and contributors. libsecp256k1 `https://github.com/sipa/secp256k1`. 2015.