Curve25519 양자회로 구현

https://youtu.be/pLhslhe6mxo

정보컴퓨터공학과 송경주

Curve25519

- Daniel J. Bernstein이 제안한 타원곡선
- ECC에서 사용되며 128비트의 보안(256비트 키 크기) 제공
- Curve 기반으로 설계된 키교환 알고리즘: X25519
- X25519: ECDH 에서 사용될 수 있음
- 곡선: $y^2 = x^3 + 488662x^2 + x$
- Prime field : $2^{255} - 19$

Algorithm 1 Curve25519 algorithm

```
Function Curve25519
PointXZMulSecure(&P1, &P2, k, P)
RecoverY(&P1, &P1, &P2, &P2, P, b)
ProToAff(R, &P1)
```

Curve25519

- PointAdd: Point addition
- PointDbl: Point Doubling
- 주요 연산들이 덧셈,뺄셈,곱셈,제곱으로 이루어짐

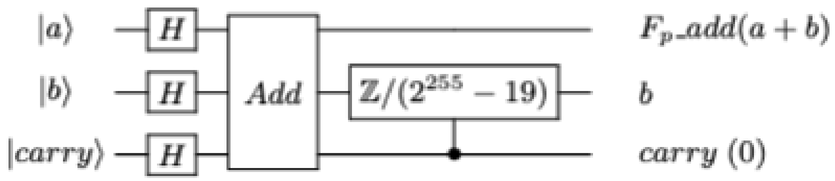
```
Function PointXZMulSecure
Input: Scalar k, Point P, R1, R2
xp ← P.x
Initialize points T[0], T[1]
T[0].x, T[0].y, T[0].z[0] ← xp, 0, 1
T[1] ← PointDbl(T[0])
for (i = 253 to −1) :
    ki ← get_bit(k, i)
    T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
    T[1-ki] ← PointAdd(T[1-ki], T[ki], xp)
    T[ki] ← PointDbl(T[ki])
R1 ← T[0]
R2 ← T[1]
```

```
Function ProToAff
xp, yp, zp ← P.x, P.y, P.z
xr, yr, zr ← R.x, R.y, R.z
t1 ← 1/zp
yr ← yp×t1
xr ← xp×t1
```

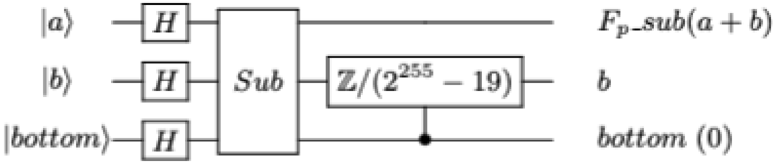
```
Function RecoverY
Input: Points P1, P2 (in projective coordinates), Point P (in affine coordinates),
    Scalar b
x1, z1, x2, z2, x, y ← P1.x, P1.z, P2.x, P2.z, P.x, P.y
xr, yr, zr ← R.x, R.y, R.z
t1 ← x × x1
t1 ← t1 - z1
t2 ← z1 × x
t2 ← x1 - t2
t3 ← z2 × t1
t4 ← x2 × t2
t2 ← 4 × b
t2 ← t2 × y
t2 ← t2 × z2
t2 ← t2 × x2
t2 ← t2 × z1
zr ← t2 × z1
xr ← t2 × x1
t2 ← t3 + t4
t3 ← t3 - t4
yr ← t2 × t3
```

Curve25519 기본연산 양자회로 구현

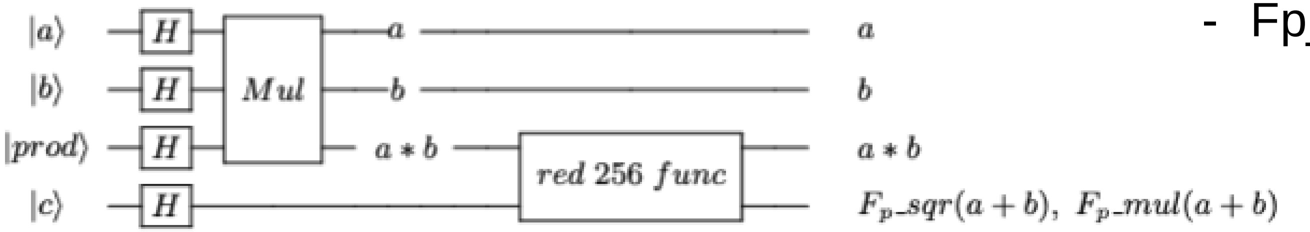
- 해당 prime field 양자회로 구현은 $\mathbb{Z}/(2^{255} - 19)$ 외에도 양자회로에서 단순 상수 변경으로 다양한 prime field 상에서 범용적으로 사용가능



(a) F_p_add



(b) F_p_sub



(c) F_p_sqr, F_p_mul

(a) Fp_add: in-place 연산
- Fp_add(a, b) → a, Fp_add(a+b)

(b) Fp_sub: in-place 연산
- Fp_sub(a, b) → a, Fp_sub(a-b)

(c) Fp_sqr, mul: out-of-place 연산 (동일)
- Fp_sqr(a, a, prod) → a, a, Fp_sqr(a^2)
- Fp_mul(a, b, prod) → a, b, Fp_sqr(a*b)

Curve25519 양자회로 구현

- PointAdd function 양자회로 최적화 구현
- 알고리즘 순서를 재배치하여 기존 알고리즘에서 사용한 t1, t2 사용을 생략할 수 있음
- 따라서 매 라운드 및 반복문에서 t1, t2 큐비트 생략가능
  - Hard work: Sequential한 algorithm을 양자회로에 적합하게 재배치해야 함
    + t1, t2 생략이 필요한 큐비트 값을 변형시키면 안됨

Algorithm 3 The operation sequence of the original PointAdd function

Input: $t_{1,2}$

$t_1 = xp + zp$
$\clubsuit_1\ t_2 = xp - zp$
$xr = xq - zq$
$zr = t_1 \times xr$
$t_1 = xq + zq$
$xr = t_1 \times t_2$
$\clubsuit_2\ t_1 = xr - zr$
$t_2 = xr + zr$
$xr = t_2 \times t_2$
$t2 = t_1 \times t_1$
$zr = xd \times t_2$

Original PointAdd function

Algorithm 4 The operation sequence of the modified PointAdd function

Input: $t_{1,2}$

$t_1 = xp + zp$
$xr = xq - zq$
$zr = t_1 \times xr$
$t_1 = xq + zq$
$\clubsuit_1\ t_2 = xp - zp$
$xr = t_1 \times t_2$
$t_2 = xr + zr$
$xr = t_2 \times t_2$
$\clubsuit_2\ t_1 = xr - zr$
$t2 = t_1 \times t_1$
$zr = xd \times t_2$

Modified PointAdd function

Curve25519 양자회로 구현

- 알고리즘 순서를 변환하여 t1, t2 사용 생략
- 연산 대상이 되는 큐비트들에 대해 target qubit 및 control qubit을 적절히 설정 → 마지막 Inverse operation 수와 연결됨
- line 10, line 11 의 순서를 통합하여 out-of-place 연산에 필요한 큐비트를 줄임(Purple text)
- 즉, line 10, 11을 통합하면 zr=xd*t1*t1 연산이 되므로 z=t1*xd를 먼저 계산한 후, zr=zr*t1을 계산함
  - 기존: (1) xr_sqr_temp = xr*xr_temp (2) zr_temp = xr_temp*xd :: (xr_sqr_temp, xr_temp, xr_temp 필요)
  - 통합: (1) zr_temp = xr*xd (2) zr_temp = xr(xr,, zr) :: (xr_temp 필요)
- line 12, 13: t1, t2 대신 사용된 큐비트의 필요한 원래 값으로 돌리는 Inverse 연산 수행

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

1: $t_1 = xp + zp$ ⇒ $xp \leftarrow F_p\_add(xp, zp)$
2: $xr = xq - zq$ ⇒ $xq \leftarrow F_p\_sub(zq, xq)$
3: $zr = t_1 \times xr$ ⇒ $zr_{ancl} \leftarrow F_p\_mul(xq, xp)$
4: $t_1 = xq + zq$ ⇒ $xq \leftarrow F_p\_add(zp, xq)$
5: $\clubsuit_1$ $t_2 = xp - zp$ ⇒ $xp \leftarrow F_p\_sub(zp, xp)$ $F_p\_sub(zp, xp)$
6: $xr = t_1 \times t_2$ ⇒ $xr_{anc2} \leftarrow F_p\_sqr(xr_{ancl}, xr_{anc2})$
7: $t_2 = xr + zr$ ⇒ $xr_{ancl} \leftarrow F_p\_add(zr_{ancl}, xr_{ancl})$
8: $xr = t_2 \times t_2$ ⇒ $xr_{anc2} \leftarrow F_p\_sqr(xr_{ancl}, xr_{anc2})$
9: $\clubsuit_2$ $t_1 = xr - zr$ ⇒ $xr_{ancl} \leftarrow F_p\_sub(zr_{anc}, xr_{ancl})$ $F_p\_sub(zr_{anc}, xr_{ancl})$

10: $t2 = t_1 \times t_1$ ⇒ Combine lines 10 and 11:
11: $zr = xd \times t_2$ $zr_{temp} \leftarrow F_p\_mul(xr_{ancl}, xd)$,
$zr_{anc2} \leftarrow F_p\_mul(xr_{ancl}, zr_temp)$

// **Inverse operations**
12: $xq \leftarrow fp_add(zq, xq)$
13: $xp \leftarrow fp_add(zp, xp)$

Curve25519 양자회로 구현

- line 1에서 xp에 xp+zp 결과를 저장하면 line 5 에서 xp-zp 값을 얻기 위해 단순히 fp_sub(zp, xp)을 두 번 사용하면 됨
- line 6: out-of-place 연산으로 진행되므로 결과를 clean 상태의xr_anc2에 저장
- line 7: zr과 xr의 결과를 담은 ancilla qubits zr_anc1, xr_anc1에 대한 뺄셈 결과 (xr_anc1-zr_anc1) 를 xr_anc1에 저장

*기존 알고리즘의 line7을 line 9로 이동하여 line 8에서 사용한 xr_anc1을 line 9에서 재사용하도록 구성

Algorithm 5 Modified PointAdd function ⇒ quantum circuit

1: $t_1 = xp + zp \Rightarrow xp \leftarrow F_p\_add(xp, zp)$
2: $xr = xq - zq \Rightarrow xq \leftarrow F_p\_sub(zq, xq)$
3: $zr = t_1 \times xr \Rightarrow zr_{anc1} \leftarrow F_p\_mul(xq, xp)$
4: $t_1 = xq + zq \Rightarrow xq \leftarrow F_p\_add(zp, xq)$
5: $\clubsuit_1\ t_2 = xp - zp \Rightarrow xp \leftarrow F_p\_sub(zp, xp)\ F_p\_sub(zp, xp)$
6: $xr = t_1 \times t_2 \Rightarrow xr_{anc2} \leftarrow F_p\_sqr(xr_{anc1}, xr_{anc2})$
7: $t_2 = xr + zr \Rightarrow xr_{anc1} \leftarrow F_p\_add(zr_{anc1}, xr_{anc1})$
8: $xr = t_2 \times t_2 \Rightarrow xr_{anc2} \leftarrow F_p\_sqr(xr_{anc1}, xr_{anc2})$
9: $\clubsuit_2\ t_1 = xr - zr \Rightarrow xr_{anc1} \leftarrow F_p\_sub(zr_{anc}, xr_{anc1})\ F_p\_sub(zr_{anc}, xr_{anc1})$

10: $t2 = t_1 \times t_1 \Rightarrow$ Combine lines 10 and 11:
11: $zr = xd \times t_2$
$zr_{temp} \leftarrow F_p\_mul(xr_{anc1}, xd)$,
$zr_{anc2} \leftarrow F_p\_mul(xr_{anc1}, zr_temp)$

// **Inverse operations**
12: $xq \leftarrow fp_add(zq, xq)$
13: $xp \leftarrow fp_add(zp, xp)$

Table 1: Qubit state for Algorithm 5 (Each operation represents a prime field operation)

Line	Qubit State					
	xr_{anc1}	xr_{anc2}	zr_{anc1}	zr_{anc2}	xp	xq
1	-	-	-	-	$xp + zp$	xq
2	-	-	-	-	$xp + zp$	$xq - zq$
3	-	-	$xq * xp$	-	$xp + zp$	$xq - zq$
4	-	-	$xq * xp$	-	$xp + zp$	$xq + zq$
5	-	-	$xq * xp$	-	$xp - zp$	$xq + zq$
6	$xq * xp$	-	$xq * xp$	-	$xp - zp$	$xq + zq$
7	$xr_{anc1} + zr$	-	$xq * xp$	-	$xp - zp$	$xq + zq$
8	$xr_{anc1} + zr$	$(xr_{anc1})^2$	$xq * xp$	-	$xp - zp$	$xq + zq$
9	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	-	$xp - zp$	$xq + zq$
10 11	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	$xr_{anc1} * xr_{anc1} * xd$	$xp - zp$	$xq + zq$
12	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	$xr_{anc1} * xr_{anc1} * xd$	$xp - zp$	xq
13	$xr_{anc1} - zr$	$(xr_{anc1})^2$	$xq * xp$	$xr_{anc1} * xr_{anc1} * xd$	xp	xq

Curve25519 양자회로 구현

- PointAdd function 양자회로 최척화 구현
- PointDbl은 모든 line에 대해 순서 변형이 어려움
- 따라서 회로 구성만을 통해 t1 사용을 제외함

Algorithm 6 PointDbl function ⇒ quantum circuit

$t_1 = xp + zp \Rightarrow xp \leftarrow F_p\_add(xp, zp)$
$t_2 = t_1 \times t_1 \Rightarrow t_2 \leftarrow F_p\_sqr(t_1)$
$t_1 = xp - zp \Rightarrow xp \leftarrow F_p\_sub(zp, xp)\ F_p\_sub(zp, xp)$
$zr = t_1 \times t1 \Rightarrow zr_{anc1} \leftarrow F_p\_mul(xp, zr_{anc1})$
$xr = t_2 \times zr \Rightarrow xr \leftarrow F_p\_mul(t_2, zr)$
$t_1 = t_2 - zr \Rightarrow t_2 \leftarrow F_p\_sub(zr, t_2)$
$t_2 = t_1 \times c \Rightarrow t_{2anc1} \leftarrow F_p\_mul(t_2, c)$
$zr = t_1 \times t_2 \Rightarrow zr_{anc2} \leftarrow F_p\_mul(t_1, t_2)$

// **Inverse operations** $xp \leftarrow F_p\_add(zp, xp)$

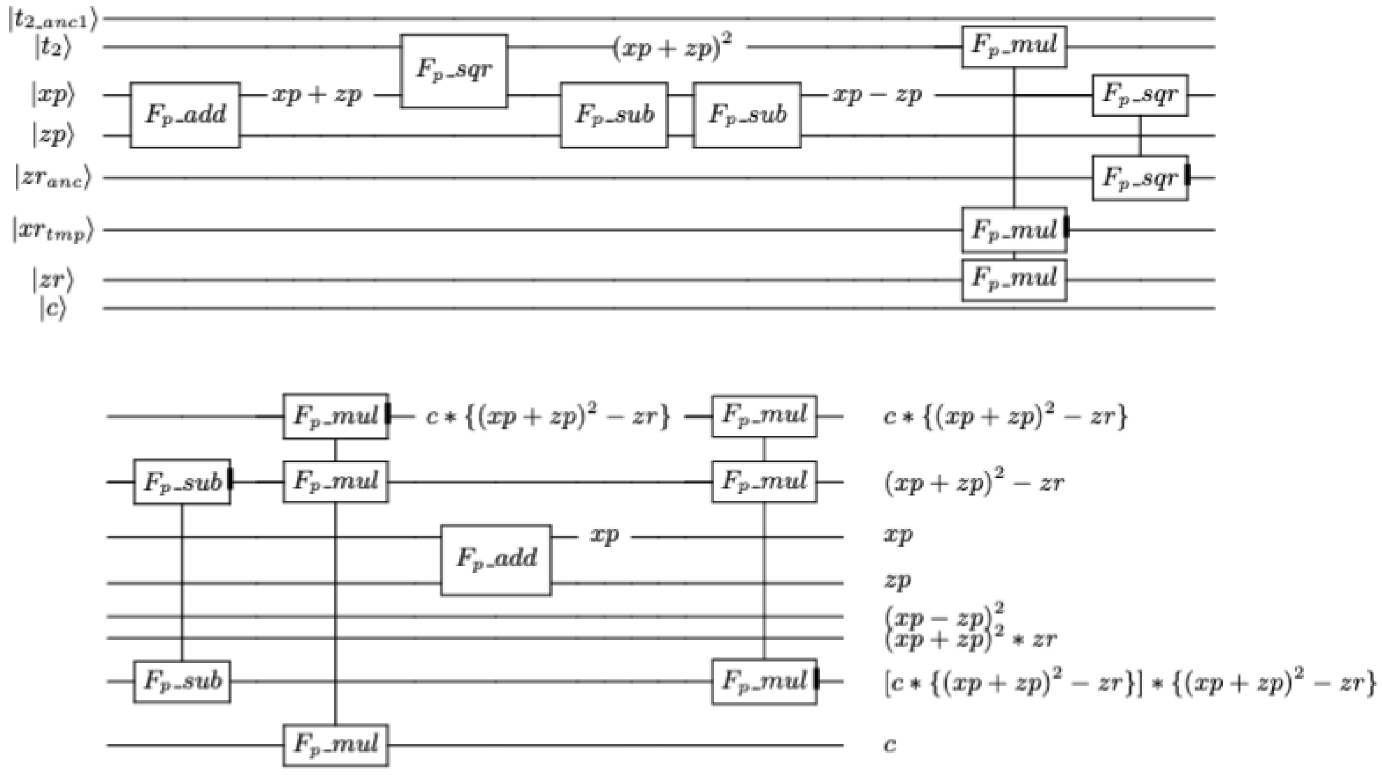


Fig. 3: Quantum circuit diagram of PointDbl. (The circuit is divided into two sections (connected from top to bottom))

Curve25519 양자회로 구현

- line 1: xp+zp의 결과는 t1대신 xp에 저장
- line2: multiplication은 oput-of-place 연산이므로 t1*t1 (즉, xp*xp) 결과가 t2에 저장
- line 3: 결과는 xp에 저장, line 1에서 계산된 xp+zp는 line 2에서 사용되었으므로 Fp_sub을 두 번 사용하여 xp-zp의 결과를 생성
- line 4, 5: out-of-place multiplication 연산 결과를 zr_anc1와 xr에 각각 저장
- line 6: t2-zr 결과를 t2에 저장하며 line 7에서 t2, c의 곱셈 결과를 t2_{anc1}에 저장
- line 8: zr_anc2에 t1, t2의 곱셈을 저장
- line9: 연산에서 xp-zp로 변경된 xp에 대해 되돌리는 Fp_add 연산 추가

Algorithm 6 PointDbl function $\Rightarrow$ quantum circuit

1: $t_1 = xp + zp$	$\Rightarrow$	$xp \leftarrow F_p\_add(xp, zp)$
2: $t_2 = t_1 \times t_1$	$\Rightarrow$	$t_2 \leftarrow F_p\_sqr(t_1)$
3: $t_1 = xp - zp$	$\Rightarrow$	$xp \leftarrow F_p\_sub(zp, xp)\ F_p\_sub(zp, xp)$
4: $zr = t_1 \times t1$	$\Rightarrow$	$zr_{anc1} \leftarrow F_p\_mul(xp, zr_{anc1})$
5: $xr = t_2 \times zr$	$\Rightarrow$	$xr \leftarrow F_p\_mul(t_2, zr)$
6: $t_1 = t_2 - zr$	$\Rightarrow$	$t_2 \leftarrow F_p\_sub(zr, t_2)$
7: $t_2 = t_1 \times c$	$\Rightarrow$	$t_{2anc1} \leftarrow F_p\_mul(t_2, c)$
8: $zr = t_1 \times t_2$	$\Rightarrow$	$zr_{anc2} \leftarrow F_p\_mul(t_1, t_2)$

// **Inverse operations** $xp \leftarrow F_p\_add(zp, xp)$

PointDbl quantum circuit

Curve25519 양자회로 구현 결과

Table 2: Quantum resources for PointDbl and PointAdd function

Function	Qubit	Quantum gates			
		CCCNOT	Toffoli	CNOT	X
PointDbl	12080 (-746)	56320	125092	18947	15
PointAdd	12604 (-1737)	81920	178902	23866	854

Table 3: Quantum resources for prime field $\mathbb{Z}/(2^{255}-19)$ operations

Function	Qubit	Quantum gates			
		CCCNOT	Toffoli	CNOT	X
F_p_add	780	5120	10762	1035	0
F_p_sub	781	5120	10762	1035	0
F_p_sqr, mul	3121	5120	12104	2702	3

Q & A