Verifying Curve25519 Software

Ming-Hsien Tsai
Institute of Information Science, Academia Sinica

Joint work with Yu-Fang Chen, Chang-Hong Hsu, Hsin-Hung Lin, Peter Schwabe, Bow-Yaw Wang, Bo-Yin Yang, and Shang-Yi Yang

Sep 19-20, 2014
Clarke Symposium

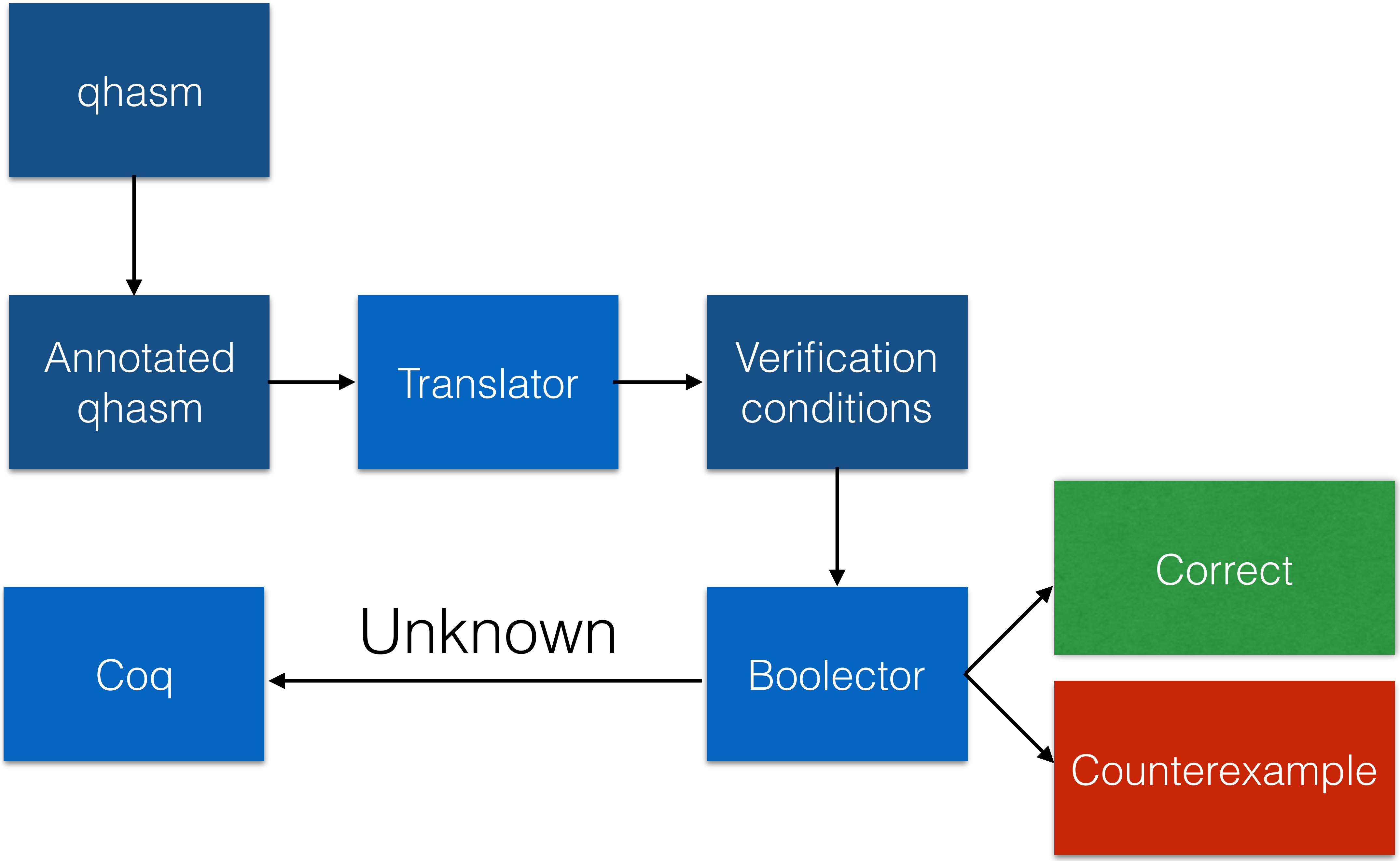
Cryptography Software

- Primitive operations are typically small
- Executed very often
- Serious optimization in low-level assembly is feasible and worth the effort
- Correctness may not be guaranteed
- Bug attack
  - Elliptic-curve implementation in OpenSSL 0.9.8g [BBPV12]

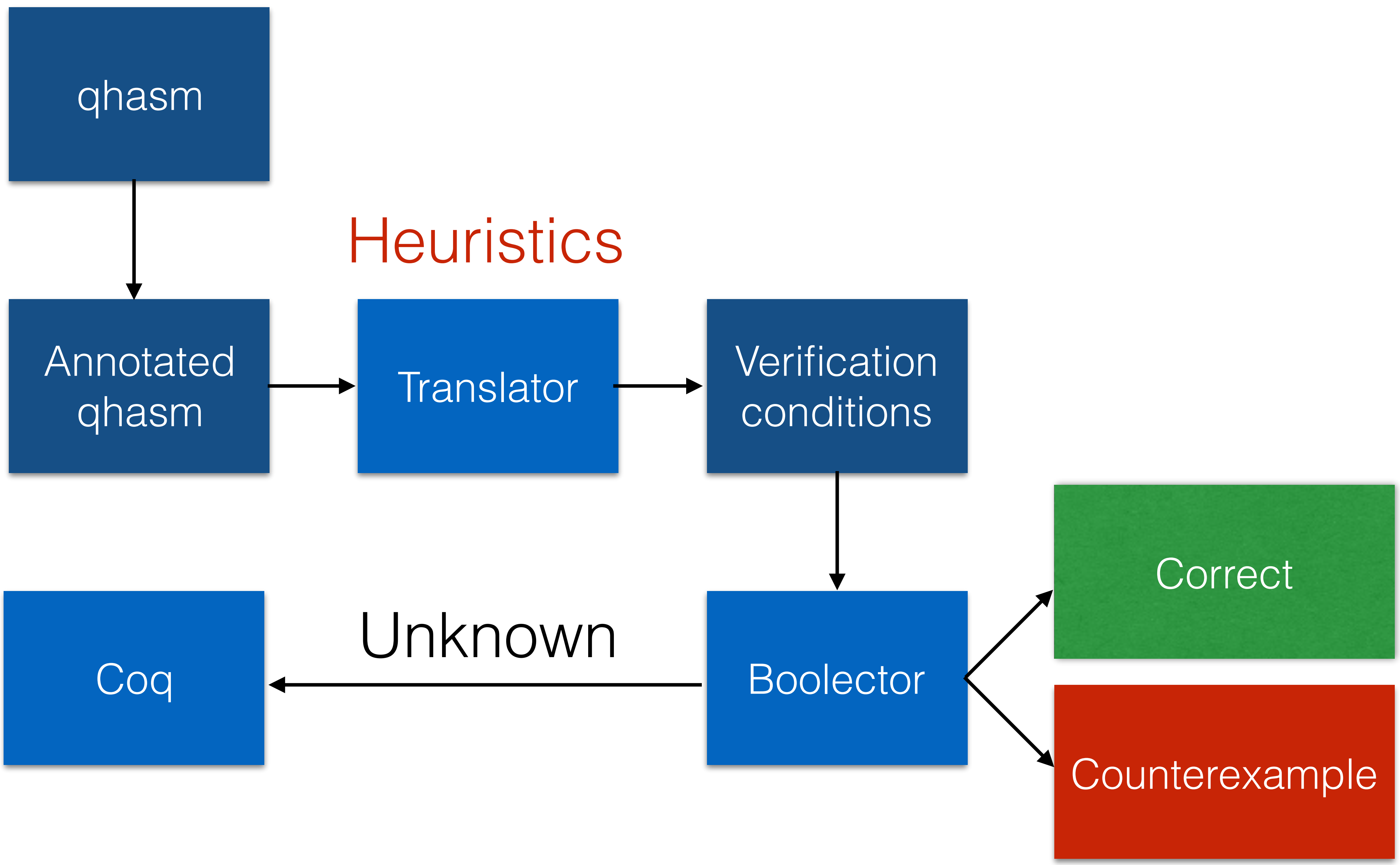
In This Work

- Formal verification of the central hand-optimized assembly routine (ladderstep) of Curve25519 Diffie-Hellman key-exchange software [Ber06]
  - Two implementations [BDL+11] written in qhasm [Ber] (~1.5K LOC)
  - Speed-record holder
- A hybrid methodology

Hybrid Methodology



Hybrid Methodology



Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP(X_1, X_2, Z_2, X_3, Z_3)

$T_1 \leftarrow X_2 + Z_2$
$T_2 \leftarrow X_2 - Z_2$
$T_7 \leftarrow T_2^2$
$T_6 \leftarrow T_1^2$
$T_5 \leftarrow T_6 - T_7$
$T_3 \leftarrow X_3 + Z_3$
$T_4 \leftarrow X_3 - Z_3$
$T_9 \leftarrow T_3 \cdot T_2$
$T_8 \leftarrow T_4 \cdot T_1$
$X_3 \leftarrow (T_8 + T_9)$
$Z_3 \leftarrow (T_8 - T_9)$
$X_3 \leftarrow X_3^2$
$Z_3 \leftarrow Z_3^2$
$Z_3 \leftarrow Z_3 \cdot X_1$
$X_2 \leftarrow T_6 \cdot T_7$
$Z_2 \leftarrow 121666 \cdot T_5$
$Z_2 \leftarrow Z_2 + T_7$
$Z_2 \leftarrow Z_2 \cdot T_5$
return (X_2, Z_2, X_3, Z_3)

end function

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP(X_1, X_2, Z_2, X_3, Z_3)

$T_1 \leftarrow X_2 + Z_2$
$T_2 \leftarrow X_2 - Z_2$
$T_7 \leftarrow T_2^2$
$T_6 \leftarrow T_1^2$
$T_5 \leftarrow T_6 - T_7$
$T_3 \leftarrow X_3 + Z_3$
$T_4 \leftarrow X_3 - Z_3$
$T_9 \leftarrow T_3 \cdot T_2$
$T_8 \leftarrow T_4 \cdot T_1$
$X_3 \leftarrow (T_8 + T_9)$
$Z_3 \leftarrow (T_8 - T_9)$
$X_3 \leftarrow X_3^2$
$Z_3 \leftarrow Z_3^2$
$Z_3 \leftarrow Z_3 \cdot X_1$
$X_2 \leftarrow T_6 \cdot T_7$
$Z_2 \leftarrow 121666 \cdot T_5$
$Z_2 \leftarrow Z_2 + T_7$
$Z_2 \leftarrow Z_2 \cdot T_5$
return (X_2, Z_2, X_3, Z_3)

end function

Arithmetic operations in F_p ($p = 2^{255}-19$)

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP$(X_1, X_2, Z_2, X_3, Z_3)$

$T_1 \leftarrow X_2 + Z_2$
$T_2 \leftarrow X_2 - Z_2$
$T_7 \leftarrow T_2^2$
$T_6 \leftarrow T_1^2$
$T_5 \leftarrow T_6 - T_7$
$T_3 \leftarrow X_3 + Z_3$
$T_4 \leftarrow X_3 - Z_3$
$T_9 \leftarrow T_3 \cdot T_2$
$T_8 \leftarrow T_4 \cdot T_1$
$X_3 \leftarrow (T_8 + T_9)$
$Z_3 \leftarrow (T_8 - T_9)$

255-bits variables

$X_3 \leftarrow X_3^2$
$Z_3 \leftarrow Z_3^2$
$Z_3 \leftarrow Z_3 \cdot X_1$
$X_2 \leftarrow T_6 \cdot T_7$
$Z_2 \leftarrow 121666 \cdot T_5$
$Z_2 \leftarrow Z_2 + T_7$
$Z_2 \leftarrow Z_2 \cdot T_5$
return (X_2, Z_2, X_3, Z_3)

end function

Arithmetic operations in F_p ($p = 2^{255}-19$)

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP$(X_1, X_2, Z_2, X_3, Z_3)$

255-bits variables

$T_1 \leftarrow X_2 + Z_2$
$T_2 \leftarrow X_2 - Z_2$
$T_7 \leftarrow T_2^2$
$T_6 \leftarrow T_1^2$
$T_5 \leftarrow T_6 - T_7$
$T_3 \leftarrow X_3 + Z_3$
$T_4 \leftarrow X_3 - Z_3$
$T_9 \leftarrow T_3 \cdot T_2$
$T_8 \leftarrow T_4 \cdot T_1$
$X_3 \leftarrow (T_8 + T_9)$
$Z_3 \leftarrow (T_8 - T_9)$

$X_3 \leftarrow X_3^2$
$Z_3 \leftarrow Z_3^2$
$Z_3 \leftarrow Z_3 \cdot X_1$
$X_2 \leftarrow T_6 \cdot T_7$
$Z_2 \leftarrow 121666 \cdot T_5$
$Z_2 \leftarrow Z_2 + T_7$
$Z_2 \leftarrow Z_2 \cdot T_5$
return (X_2, Z_2, X_3, Z_3)

end function

$T_9 \equiv T_3 T_2 \pmod{p}$

Arithmetic operations in F_p $(p = 2^{255} - 19)$

Multiplication (Radix-2^{51})

- Compute $R \equiv XY \pmod{p}$

$$\begin{aligned} X &= x_4 2^{204} + x_3 2^{153} + x_2 2^{102} + x_1 2^{51} + x_0 \\ Y &= y_4 2^{204} + y_3 2^{153} + y_2 2^{102} + y_1 2^{51} + y_0 \\ R &= r_4 2^{204} + r_3 2^{153} + r_2 2^{102} + r_1 2^{51} + r_0 \end{aligned}$$

- The naive approach has three steps
  - Multiply
  - Reduce
  - Delayed carry
- The efficient implementation merges Multiply and Reduce

Specification of Multiplication

$\{ \ 0 \leq x_0, \ x_1, \ x_2, \ x_3, \ x_4 < 2^{52} \ \&\& \ 0 \leq y_0, \ y_1, \ y_2, \ y_3, \ y_4 < 2^{52} \ \}$

`Multiply`

`Reduce`

`Delayed-Carry`

$\{$

$R \equiv XY \pmod{p} \ \&\&$

$0 \leq r_0 < 2^{52} \ \&\&$

$0 \leq r_1 < 2^{52} \ \&\&$

$0 \leq r_2 < 2^{52} \ \&\&$

$0 \leq r_3 < 2^{52} \ \&\&$

$0 \leq r_4 < 2^{52}$

$\}$

Specification of Multiplication

$\{ \ 0 \leq x_0, x_1, x_2, x_3, x_4 < 2^{52} \ \&\& \ 0 \leq y_0, y_1, y_2, y_3, y_4 < 2^{52} \ \}$

```
Multiply
Reduce
Delayed-Carry
```

$\{$

$R \equiv XY \pmod{p} \ \&\&$

$0 \leq r_0 < 2^{52} \ \&\&$

$0 \leq r_1 < 2^{52} \ \&\&$

$0 \leq r_2 < 2^{52} \ \&\&$

$0 \leq r_3 < 2^{52} \ \&\&$

Not proven!

$0 \leq r_4 < 2^{52}$

$\}$

Specification of Multiplication Revisited

$\{\ P\ \}$

`Multiply`

$\{\ R_1\ \}$

`Reduce`

$\{\ R_2\ \}$

`Delayed-Carry`

$\{\ Q\ \}$

Specification of Multiplication Revisited

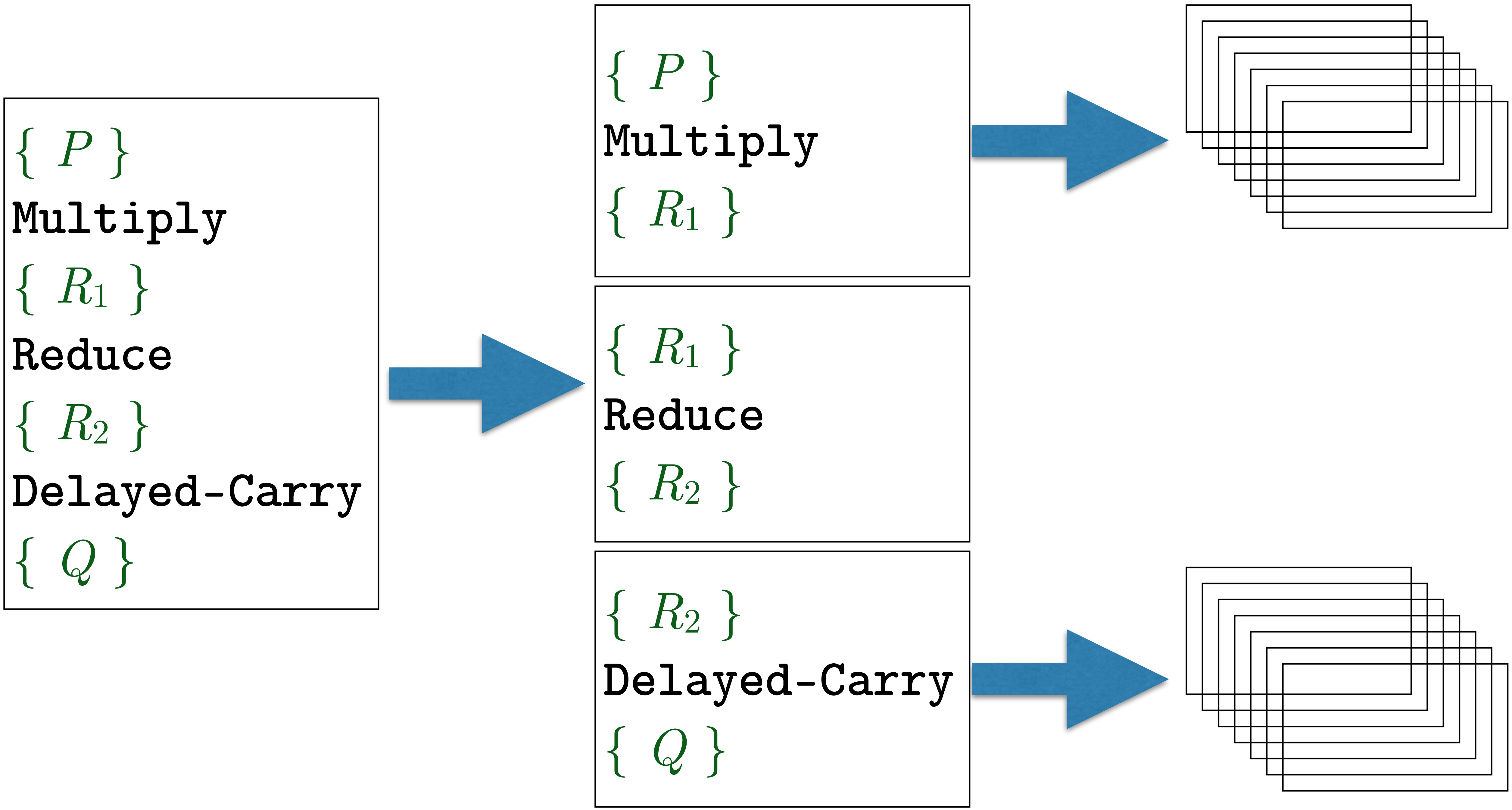
$\{\ P\ \}$
`Multiply`
$\{\ R_1\ \}$
`Reduce`
$\{\ R_2\ \}$
`Delayed-Carry`
$\{\ Q\ \}$

$\{\ P\ \}$
`Multiply`
$\{\ R_1\ \}$

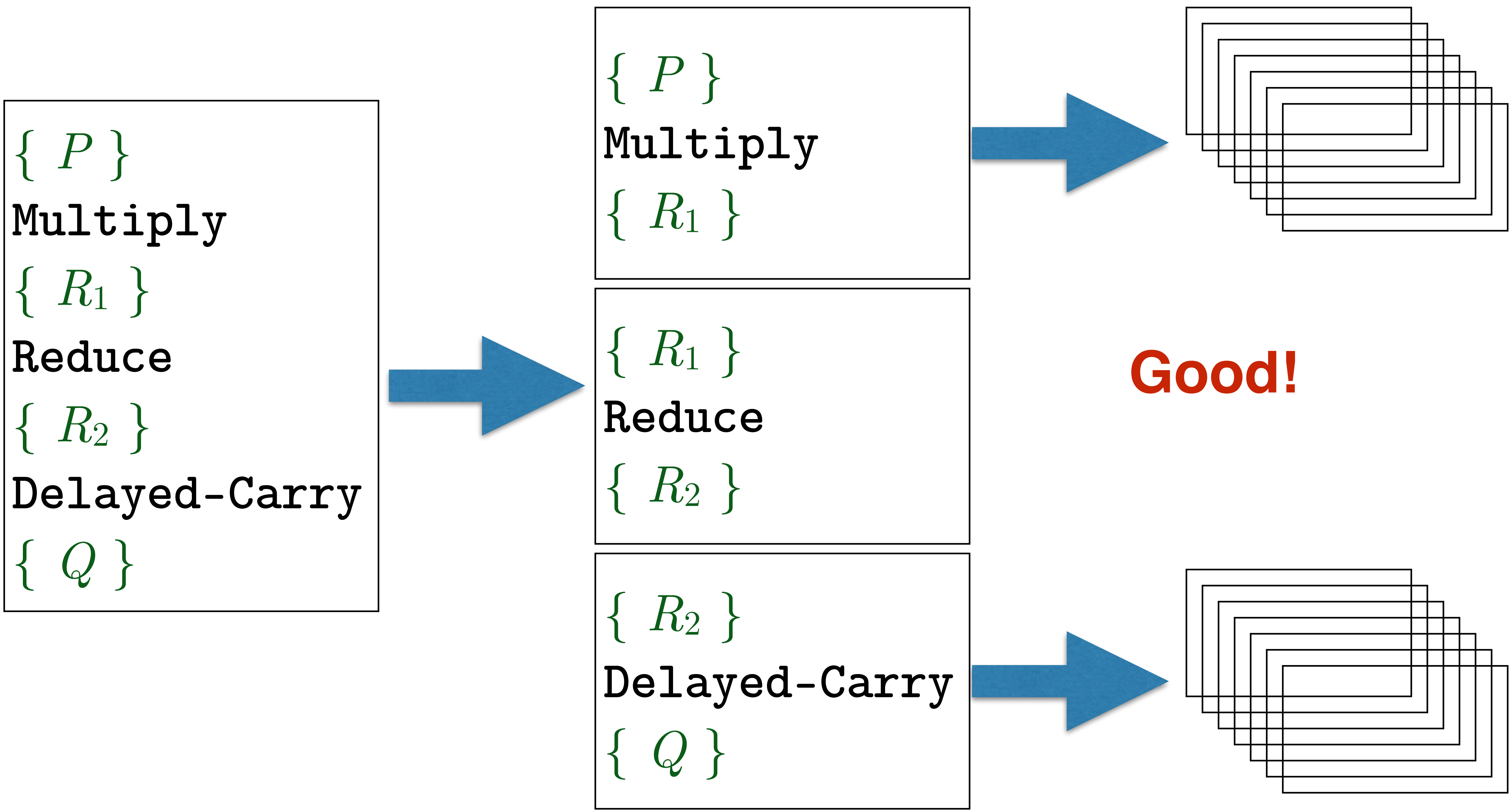
$\{\ R_1\ \}$
`Reduce`
$\{\ R_2\ \}$

$\{\ R_2\ \}$
`Delayed-Carry`
$\{\ Q\ \}$

Specification of Multiplication Revisited



Specification of Multiplication Revisited



Specification of Multiplication Revisited

$\{\ P\ \}$
`Multiply`
$\{\ R_1\ \}$
`Reduce`
$\{\ R_2\ \}$
`Delayed-Carry`
$\{\ Q\ \}$

$\{\ P\ \}$
`Multiply`
$\{\ R_1\ \}$

$\{\ R_1\ \}$
`Reduce`
$\{\ R_2\ \}$

$\{\ R_2\ \}$
`Delayed-Carry`
$\{\ Q\ \}$

Good!

Not enough!

Simple but Failed

$\{ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \text{ \&\& } r11.r1 = 2 * xp[0]@128 * xp[8]@128 \}$

read memory
shift left
128-bit multiplication

```
rax = *(uint64 *)(xp + 0)
rax <<= 1
(uint128) rdx rax = rax * *(uint64 *)(xp + 16)
r2 = rax
r21 = rdx
```

$\{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 \}$

xp[n] is a shorthand of *(uint64*)(xp + n)

x@n: extension of x to n bits

Simple but Failed

read memory
shift left
128-bit multiplication

$\{ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \ \&\& \ r11.r1 = 2 * xp[0]@128 * xp[8]@128 \}$

```
rax = *(uint64 *)(xp + 0)
rax <<= 1
(uint128) rdx rax = rax * *(uint64 *)(xp + 16)
r2 = rax
r21 = rdx
```

$\{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 \}$

xp[n] is a shorthand of *(uint64*)(xp + n)
x@n: extension of x to n bits

Need more heuristics to reduce the complexity

Heuristic 1

- Split Conjunctions -

$\{ P \}$
Code
$\{ Q_1 \ \&\& \ Q_2 \}$

→

$\{ P \}$
Code
$\{ Q_1 \}$

$\{ P \}$
Code
$\{ Q_2 \}$

Heuristic 2
- Delayed Extension -

$$R = (x_0@256 * \mathrm{y}_0@256)\ 2^{64} + ...$$

$$R = (\mathrm{x}_0@128 * \mathrm{y}_0@128)@256\ 2^{64} + ...$$

R is a 256-bit vector

x_0 and y_0 are 64-bit vectors

Heuristic 3

- Match Code -

$\{\ P\ \}$

$\texttt{rh.r1 = 19x}_0\texttt{y}_1$

$\texttt{rh.rl += 19x}_1\texttt{y}_0$

$\{\ \mathrm{rh.rl} = 19(x_0y_1 + x_1y_0)\ \}$

$\{\ P\ \}$

$\texttt{rh.rl = 19x}_0\texttt{y}_1$

$\texttt{rh.rl += 19x}_1\texttt{y}_0$

$\{\ \mathrm{rh.rl} = 19x_0y_1 + 19x_1y_0\ \}$

Heuristic 4
- Over-approximation -

$\{ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \ \&\& \ r11.r1 = 2 * xp[0]@128 * xp[8]@128 \}$

```
rax = *(uint64 *)(xp + 0)
rax <<= 1
(uint128) rdx rax = rax * *(uint64 *)(xp + 16)
r2 = rax
r21 = rdx
```

$\{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 \}$

Heuristic 4
- Over-approximation -

{ $0 \leq$ xp[0], xp[8], xp[16] $< 2^{54}$ && ~~r11.r1 = 2 * xp[0]@128 * xp[8]@128~~ }

```
rax = *(uint64 *)(xp + 0)
rax <<= 1
(uint128) rdx rax = rax * *(uint64 *)(xp + 16)
r2 = rax
r21 = rdx
```

{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 }

Heuristic 4

- Over-approximation -

{ $0 \leq$ xp[0], xp[8], xp[16] $< 2^{54}$ && ~~r11.r1 = 2 * xp[0]@128 * xp[8]@128~~ }

```
rax = *(uint64 *)(xp + 0)
rax <<= 1
(uint128) rdx rax = rax * *(uint64 *)(xp + 16)
r2 = rax
r21 = rdx
```

{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 }

Success: done

Fail: prove the original one

Experimental Results

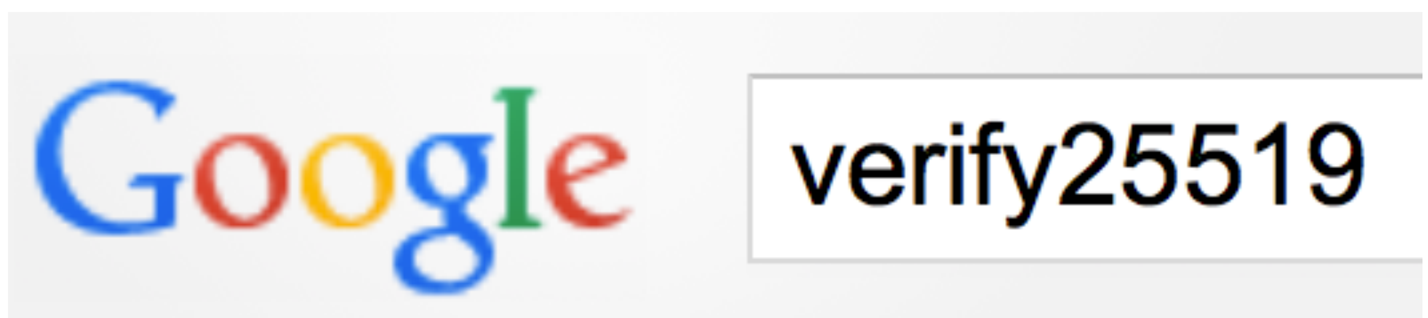
File Name	Description		# of *limb*	# of *MC*	Time
radix-2^{64} representation					
fe25519r64_mul-1	$r = x * y \pmod{2^{255} - 19}$, a buggy version		4	1	0m8.73s
fe25519r64_add	$r = x + y \pmod{2^{255} - 19}$	Operations of Algorithm 2	4	0	0m3.15s
fe25519r64_sub	$r = x - y \pmod{2^{255} - 19}$		4	0	0m16.24s
fe25519r64_mul-2	$r = x * y \pmod{2^{255} - 19}$, a fixed version of fe25519r64_mul-1		4	19	73m55.16s
fe25519r64_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		4	2	0m2.03s
fe25519r64_sq	$r = x * x \pmod{2^{255} - 19}$		4	15	3m16.67s
ladderstepr64	The implementation of Algorithm 2		4	14	0m3.23s
fe19119_mul	$r = x * y \pmod{2^{191} - 19}$		3	12	8m43.07s
mul1271	$r = x * y \pmod{2^{127} - 1}$		2	1	141m22.06s
radix-2^{51} representation					
fe25519_add	$r = x + y \pmod{2^{255} - 19}$	Operations of Algorithm 2	5	0	0m16.35s
fe25519_sub	$r = x - y \pmod{2^{255} - 19}$		5	0	3m38.62s
fe25519_mul	$r = x * y \pmod{2^{255} - 19}$		5	27	5658m2.15s
fe25519_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		5	5	0m12.75s
fe25519_sq	$r = x * x \pmod{2^{255} - 19}$		5	17	463m59.5s
ladderstep	The implementation of Algorithm 2		5	14	1m29.05s
mul25519	$r = x * y \pmod{2^{255} - 19}$, a 3-phase implementation		5	3	286m52.75s
mul25519-p2-1	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$		5	1	2723m16.56s
mul25519-p2-2	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$ with two sub-phases		5	2	263m35.46s
muladd25519	$r = x * y + z \pmod{2^{255} - 19}$		5	7	1569m11.06s
re15319	$r = x * y \pmod{2^{153} - 19}$		3	3	2409m16.89s

Future Work

- Automatic generation of mid-conditions
- Verification of our translator
- Better connection with Coq
- Verify other parts in Curve25519

Tools and software presented in this paper are available at

http://cryptojedi.org/crypto/#verify25519



Thank you for your attention!