Tessera

An Open-Source Hardware Framework for Cryptographic Kernels

Gaurav Kuwar, Alhad Daftardar, Jianqiao Mo, Brandon Reagen

New York University

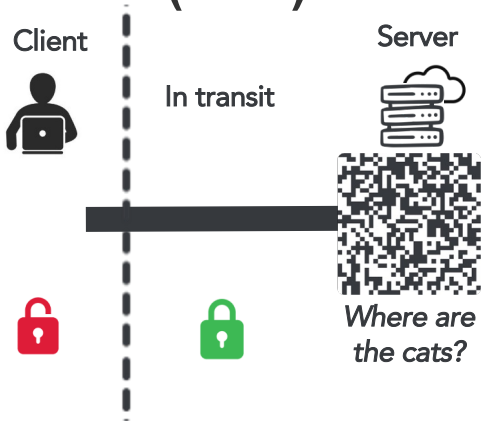
OSCAR Workshop @ ISCA'26

June 28, 2026 – Raleigh, NC, USA

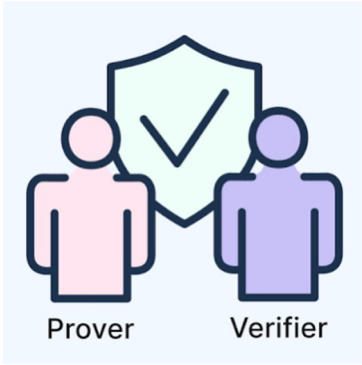
NYU TANDON

Privacy-Preserving Computation

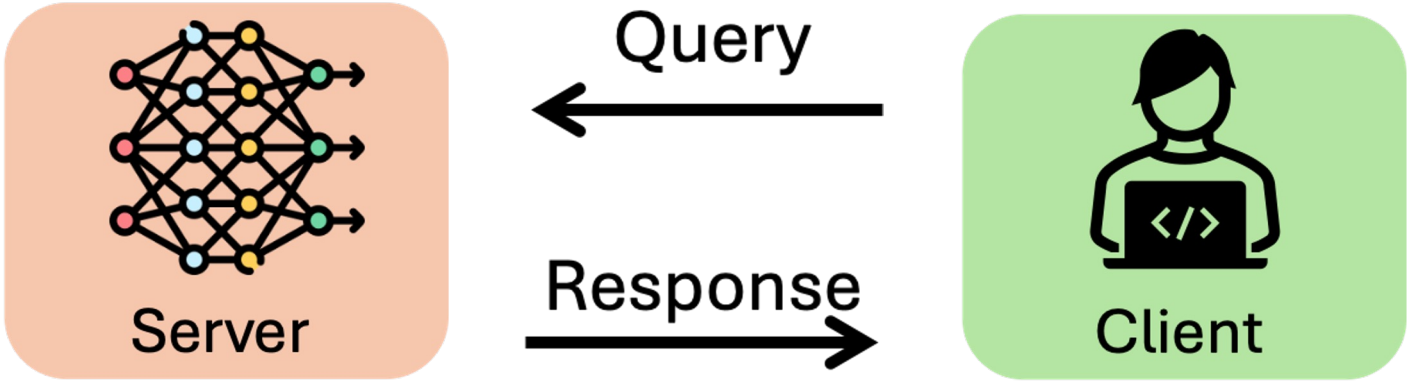
Fully Homomorphic Encryption (FHE)



Zero-Knowledge Proofs (ZKPs)



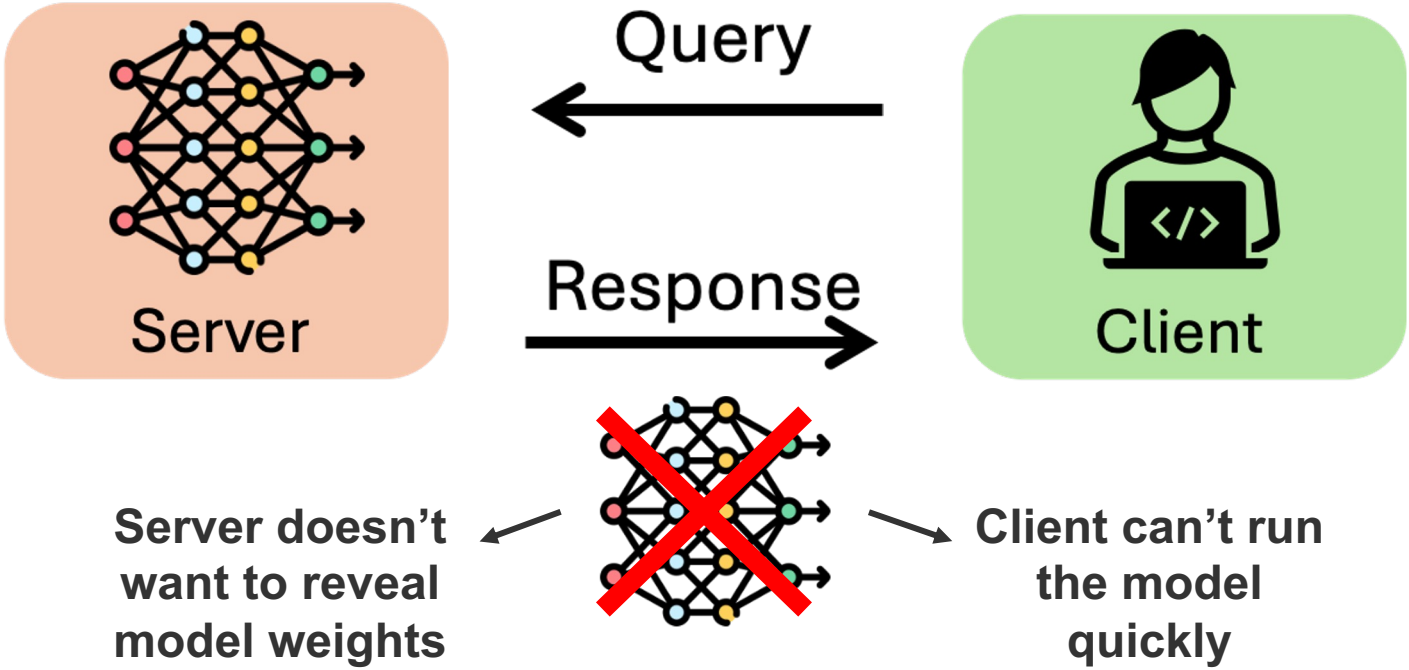
The Current Paradigm



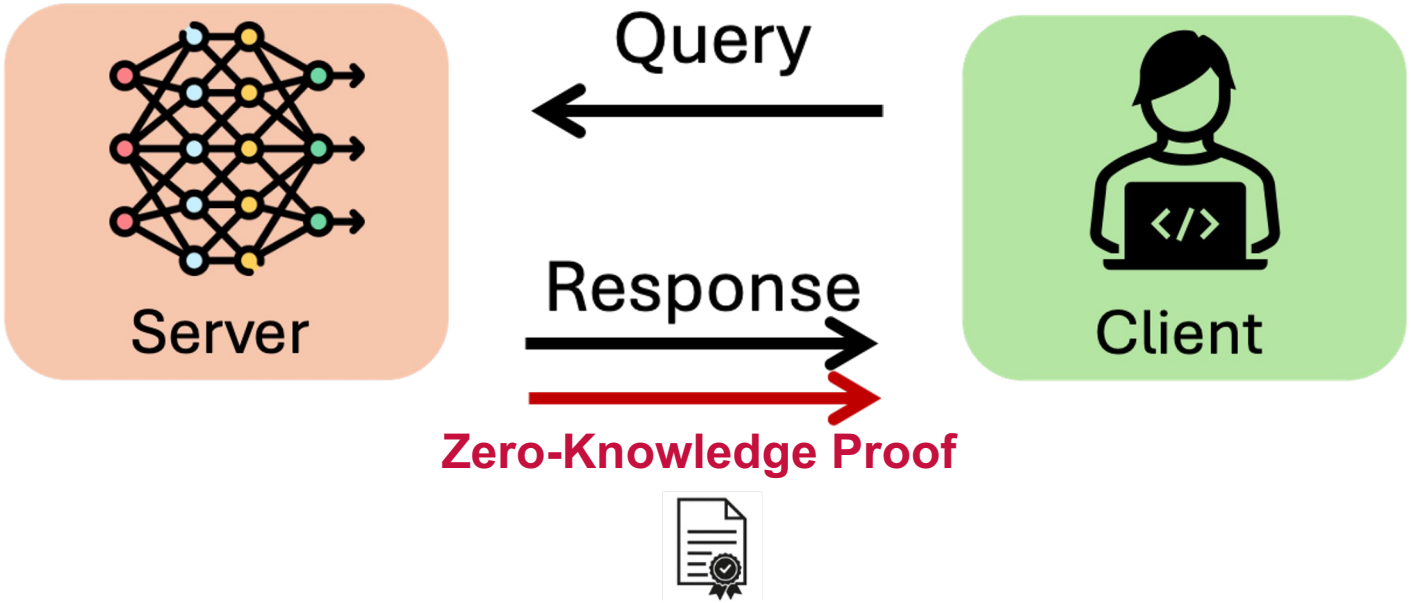
Did the server use the claimed model?
Are the outputs correct?
Is the model regulatory compliant?

NYU TANDON

The Current Paradigm



A Potential Solution



ZKPs certify correct computation,
***without* exposing secrets**

More Applications

Electronic Voting

Private Transactions

General-purpose Verifiable Compute

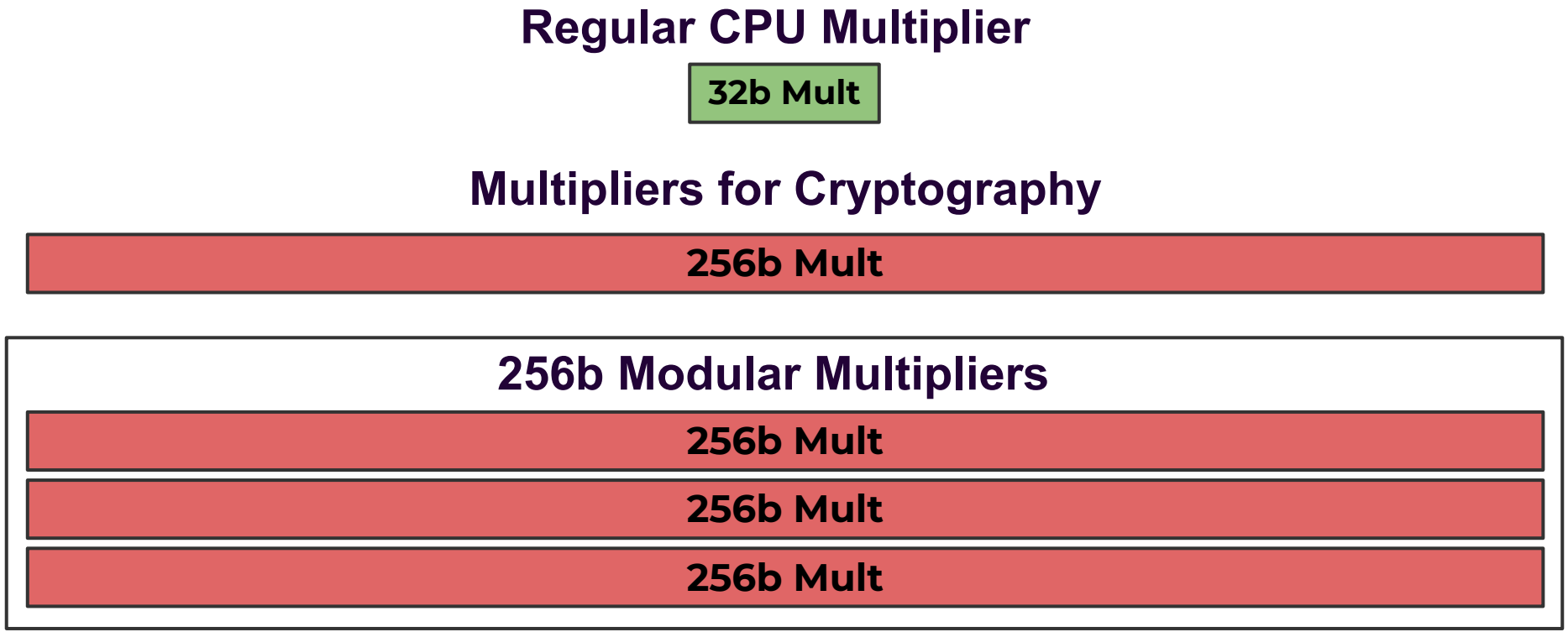
Why aren’t these widely adopted?

They are Slow!

Why are they slow?

Very Large Bitwidth Arithmetic Kernels

How big are these kernels?



Each protocol run requires

Millions of
Modular Multiplications

Prior Works

Build dedicated hardware chips with **order of magnitudes** faster than CPU

ZKP

zkSpeed [ISCA'25]

zkPHIRE [HPCA'26]

PipeZK [ISCA'21]

FHE

F1 [MICRO'21]

SHARP [ISCA'23]

Anaheim [HPCA'25]

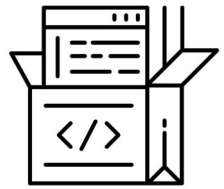
Osiris [TACO'26]

Challenges Today

- Prior Works operate in Silos
- Focused on end-to-end accelerator and high-level kernels
- **High Barrier to Entry**

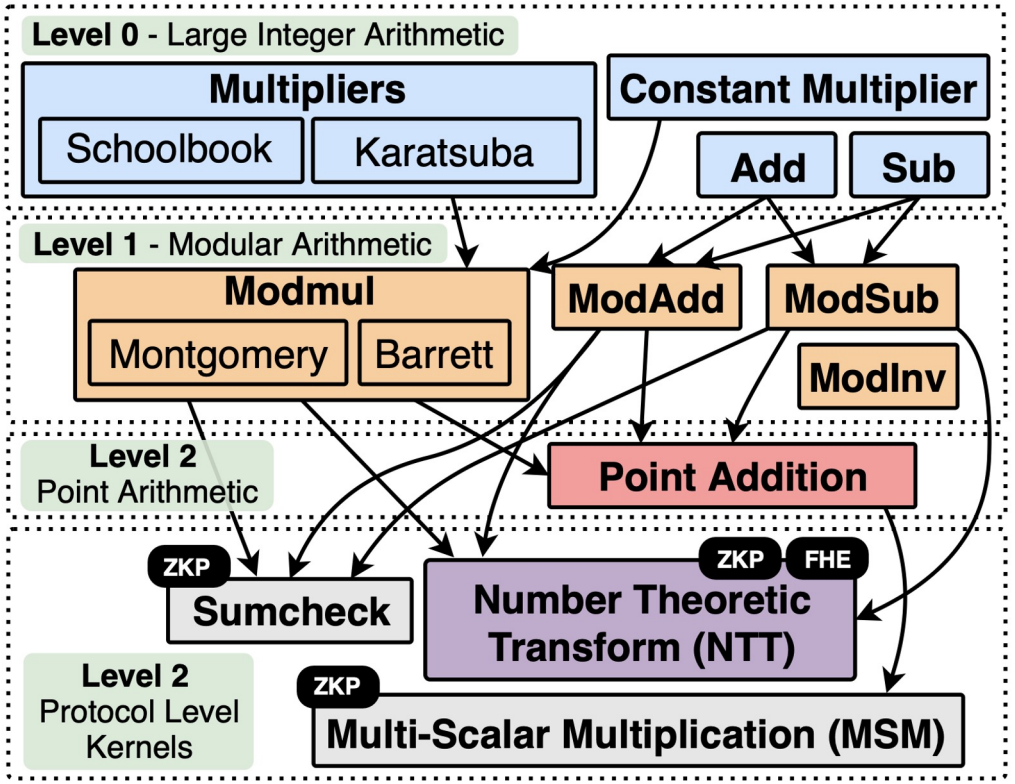
Tessera

- Framework built on High-Level Synthesis (HLS)
- Open-source and encourages community effort
- Enables Large-Scale exploration and optimization
- Foundational Library of Cryptographic kernels
- Learning Resource



Library of Kernels

- Modular Framework
- Smaller kernels build into larger ones
- Every kernel can perform its own DSE



Tessera Optimizations

Schoolbook and Karatsuba Decomposition

- Operands can be split in 2 parts
- Schoolbook decomposes into **4 partial product**
- Karatsuba reduces it into **3 partial product**
- Combined with Shifts and Adds

Operand Decomposition

$$A = \overbrace{A_H}^{32b/16b} \cdot 2^{16} + \overbrace{A_L}^{16b} \qquad B = \overbrace{B_H}^{32b/16b} \cdot 2^{16} + \overbrace{B_L}^{16b}$$

Schoolbook

$$A \cdot B = \underbrace{A_H B_H}_{16\times16} \cdot 2^{32} + (\underbrace{A_H B_L}_{16\times16} + \underbrace{A_L B_H}_{16\times16}) \cdot 2^{16} + \underbrace{A_L B_L}_{16\times16}$$

$$A \cdot B = z_3 \cdot 2^{32} + (z_2 + z_1) \cdot 2^{16} + z_0$$

Karatsuba

$$\text{Let } z_{1,2} = (A_H + A_L)(B_H + B_L)$$

$$A \cdot B = \underbrace{z_3}_{16\times16} \cdot 2^{32} + (\underbrace{z_{1,2}}_{17\times17} - z_3 - z_0) \cdot 2^{16} + \underbrace{z_0}_{16\times16}$$

Hardware Effect

Karatsuba has less area, but longer latency compared to Schoolbook

Operand Decomposition

$$A = \underbrace{A_H}_{32b} \cdot 2^{16} + \underbrace{A_L}_{16b} \qquad B = \underbrace{B_H}_{32b} \cdot 2^{16} + \underbrace{B_L}_{16b}$$

Schoolbook

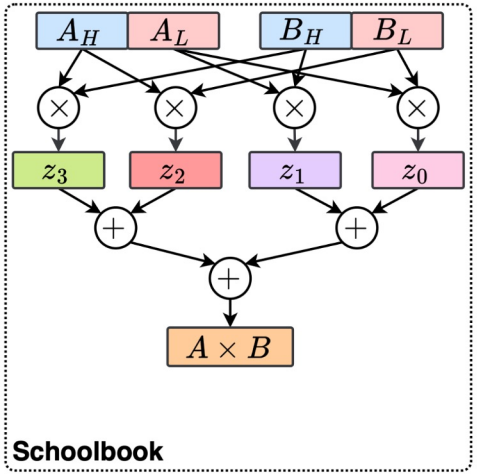
$$A \cdot B = \underbrace{A_H B_H}_{16\times16} \cdot 2^{32} + (\underbrace{A_H B_L}_{16\times16} + \underbrace{A_L B_H}_{16\times16}) \cdot 2^{16} + \underbrace{A_L B_L}_{16\times16}$$

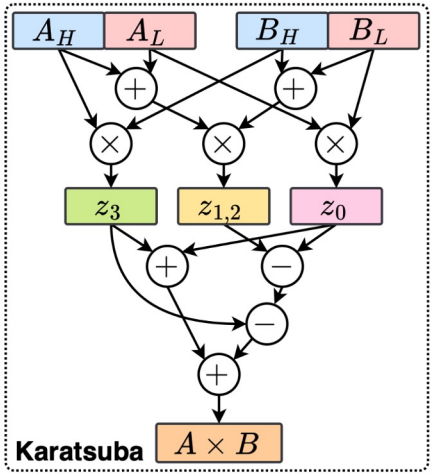
$$A \cdot B = z_3 \cdot 2^{32} + (z_2 + z_1) \cdot 2^{16} + z_0$$

Karatsuba

$$\text{Let } z_{1,2} = (A_H + A_L)(B_H + B_L)$$

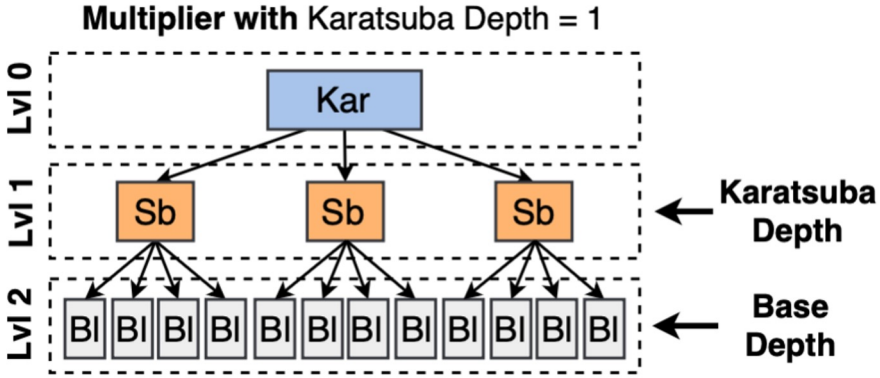
$$A \cdot B = \underbrace{z_3}_{16\times16} \cdot 2^{32} + (\underbrace{z_{1,2}}_{17\times17} - z_3 - z_0) \cdot 2^{16} + \underbrace{z_0}_{16\times16}$$

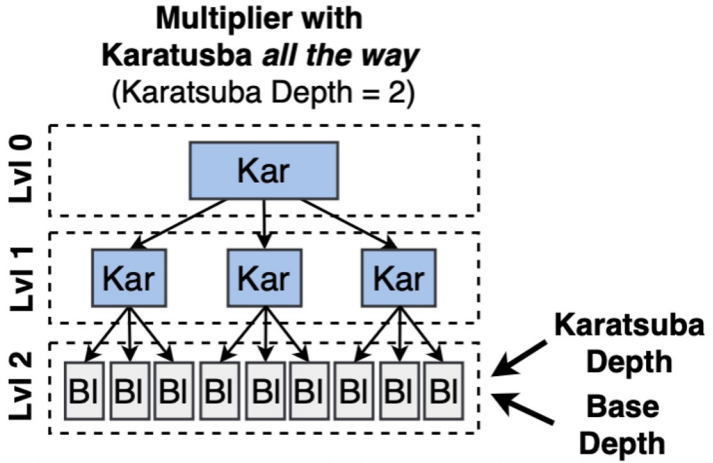




Applied Recursively

- We can apply the decompositions again on decomposed partial products recursively.
- Explore a range of area-latency trade-offs.





Modular Multipliers and Reductions Algorithms

// Montgomery Modular Multiplication
Require: $n \leftarrow bitwidth(q)$
$R \leftarrow 2^n$
$q' \leftarrow -q^{-1} \bmod R$

Function ModmulMont(a, b):
// a, b must be in Montgomery Domain
// returns $a\,b\,R^{-1} \bmod q$
$t \leftarrow a \cdot b;$
$m \leftarrow (t \cdot q') \ \& \ (R-1);$
$u \leftarrow (t + m \cdot q) \gg n;$
if $u \geq q$ **then**
  $u \leftarrow u - q;$
end
return $u;$

// Barrett Modular Multiplication
Require: $n \leftarrow bitwidth(q)$
$\mu \leftarrow \lfloor 2^{2n}/q \rfloor$

Function ModmulBarrett(a, b):
// returns $a\,b \bmod q$
$t \leftarrow a \cdot b;$
$m \leftarrow (t \cdot \mu) \gg 2n;$
$u \leftarrow t - m \cdot q;$
if $u \geq 2q$ **then**
  $u \leftarrow u - 2q;$
else if $u \geq q$ **then**
  $u \leftarrow u - q;$
return $u;$

Large Integer Multiplications

Constant Multipliers

- 2/3 Multiplications in Modmuls are by a constant curve parameter.
- Enables us to use Constant Multipliers.

```
// Montgomery Modular Multiplication
Require: n ← bitwidth(q)
         R ← 2^n
         q' ← −q^{−1} mod R
Function ModmulMont(a, b):
    // a, b must be in Montgomery Domain
    // returns a b R^{−1} mod q
    t ← a · b;
    m ← (t · q') & (R − 1);
    u ← (t + m · q) ≫ n;
    if u ≥ q then
        u ← u − q;
    end
    return u;
```

Multiplications by constant

Curve Parameters Matter

secp256k1 (Bitcoin)

Barrett is better

```
"q": "fffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f",
"q_prime": "c9bd1905155383999c46c2c295f2b761bcb223fedc24a059d838091dd2253531",
"mu": "1000000000000000000000000000000000000000000000001000003d1",
```

BN254 (Popular ZKP curve)

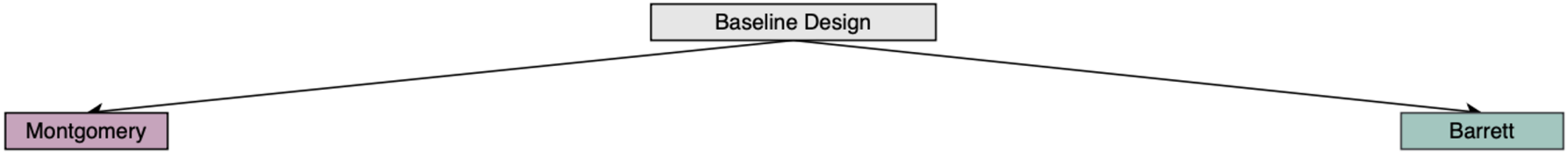
Variable multiplier is better

```
"q": "2523648240000001ba344d80000000086121000000000013a700000000000013",
"q_prime": "365373ccba60808c92022379c45b843c6e371ba81104f6c808435e50d79435e5",
"mu": "6e4a6484093af33947fcedba37bb36c39240464503b56f6105a5f8d224c735a9",
```

Large Scale Design Space Exploration

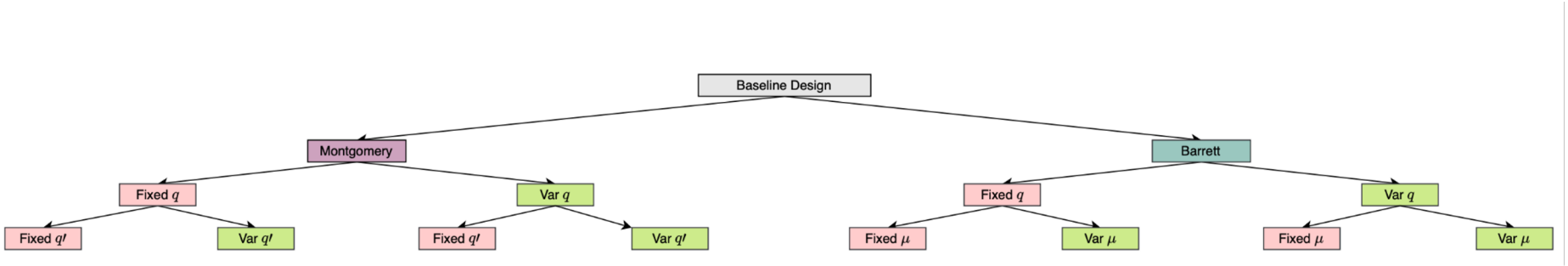
Unoptimized Baseline Design

Large Scale Design Space Exploration



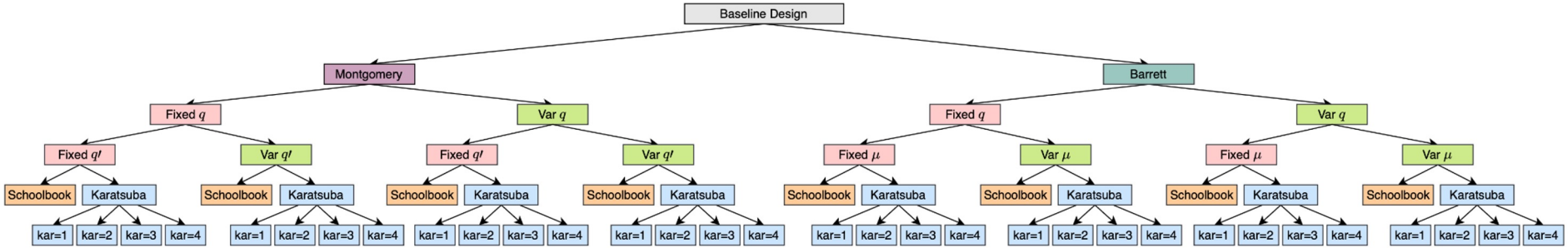
Modular Reduction Types

Large Scale Design Space Exploration



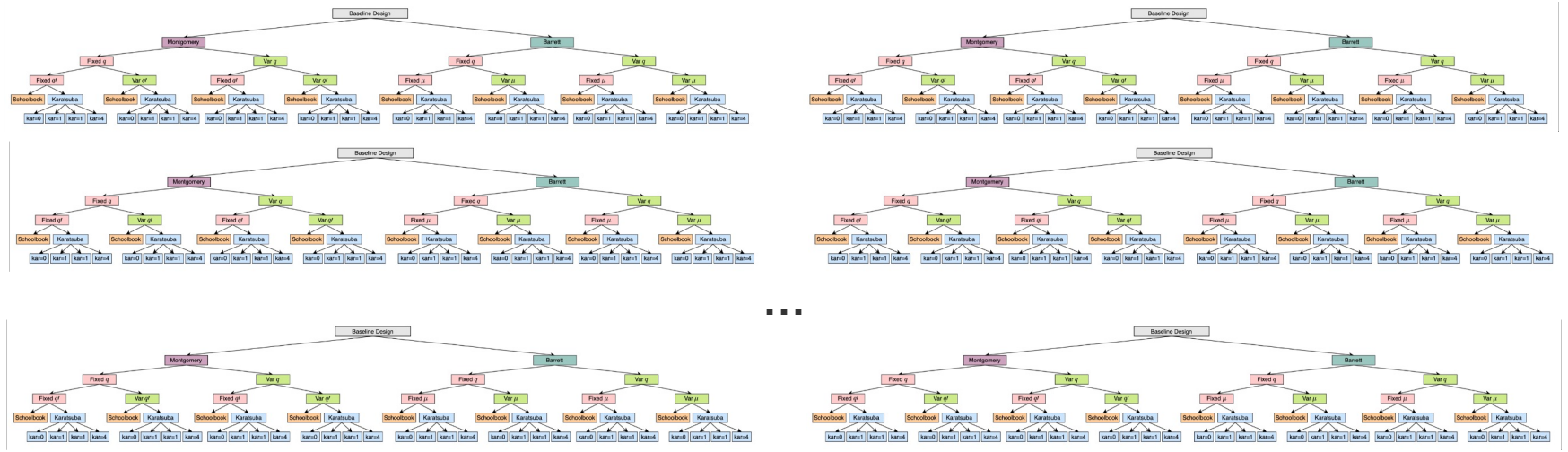
Different Constant Fixing Combinations

Large Scale Design Space Exploration



With Multiplier selections added in
Complete Design Space for a *single* curve

Large Scale Design Space Exploration

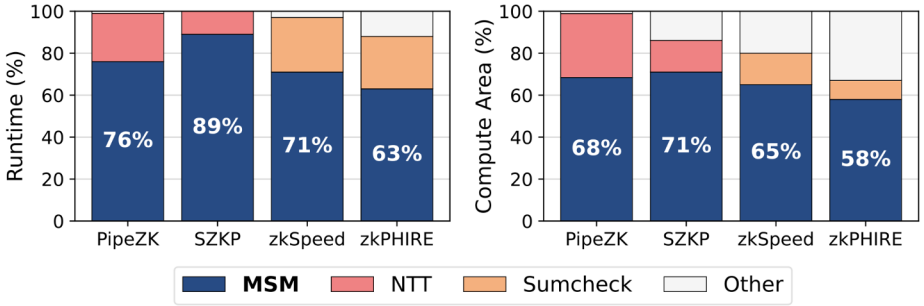


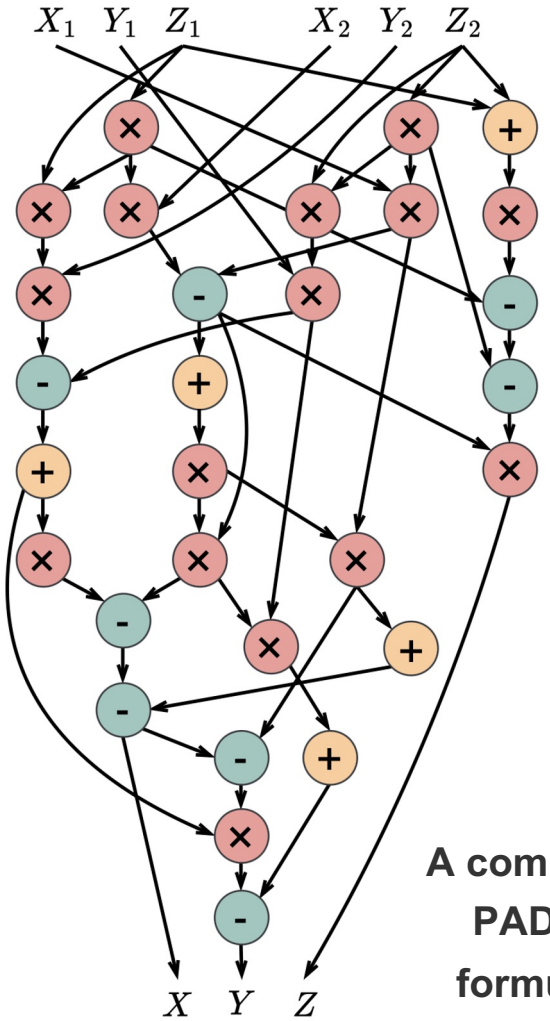
Massive Design Space across many Curves

Case Study
Point Addition Kernel

What is Point Addition (PADD)?

- Multi-Scalar Multiplication (MSM) is a runtime dominate kernel in ZKP
- Most of the MSM runtime is spent doing repeated PADD operations
- A DAG of **modular operations**





A common PADD formula

Design Space Setup

- Sweep over all the optimizations on relevant elliptic curves.
- Clocked at 1GHz on GF12nm

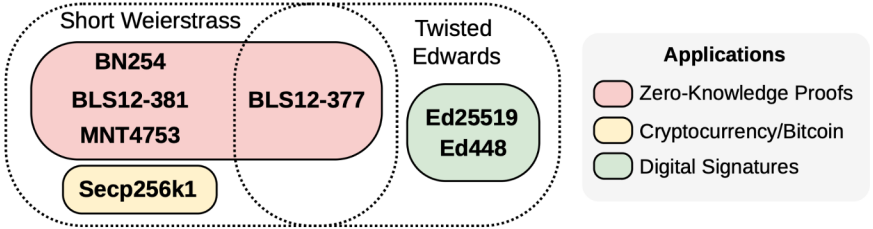
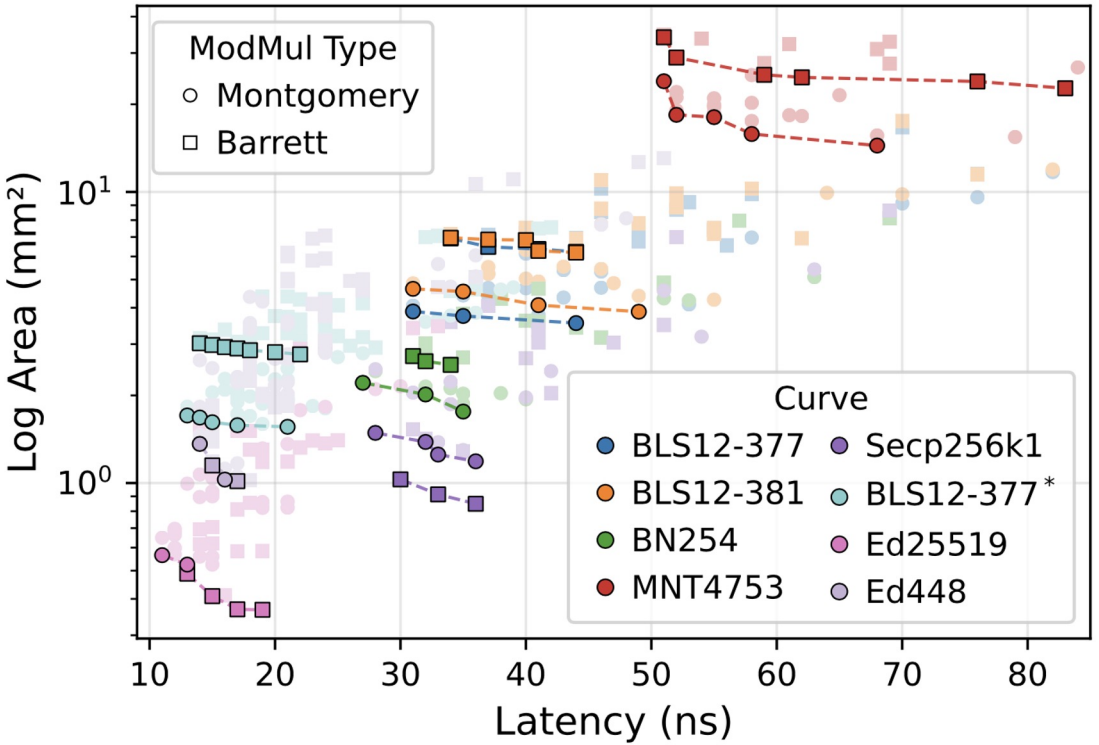


Table 1: PADD Design Space for ASIC

Design Setting	Values
Curve Type	BN254, BLS12-377, BLS12-381, MNT4753, Secp256k1, Ed25519, Ed448, BLS12-377*
ModMul Algorithm	Montgomery, Barrett
Multiplier Type	Baseline, Schoolbook, Karatsuba
Karatsuba Depth	1, 2, 3, 4
Prime (q) Type	Fixed, Variable
Reduction Const Type	Fixed, Variable
Equation Coefficient Type	Fixed, Variable
Precision Type	Single-Precision, Multi-Precision
Multi-Prec word width	16, 32, 64, 128

Pareto Analysis

We can explore a massive PADD design space!



Non-obvious Optimal Design choices

- The configs are *not* obvious, but curve-dependent
- Tessera allows us to find these optimal points easily

Table 2: Design Knobs for Best Performance Designs

Curve	Montgomery								Barrett							
	Fastest				Smallest				Fastest				Smallest			
	ct	qt	rct	kar	ct	qt	rct	kar	ct	qt	rct	kar	ct	qt	rct	kar
BN254	-	F	V	0	-	V	F	2	-	V	F	0	-	V	F	2
BLS12-377	-	F	F	2	-	V	F	3	-	F	F	2	-	V	F	3
BLS12-381	-	F	F	2	-	V	F	3	-	F	F	2	-	V	F	3
MNT4753	-	V	F	0	-	V	F	4	-	F	F	3	-	V	V	4
Secp256k1	-	F	V	0	-	F	F	2	-	F	F	0	-	F	F	2
Ed25519	F	F	F	1	F	F	F	2	F	F	F	0	V	F	F	2
BLS12-377*	F	F	F	2	V	V	F	3	F	F	F	2	V	V	F	3
Ed448	F	F	F	1	F	F	F	3	F	F	F	2	F	F	F	3

ct=curve coefficient type, **qt**=prime q type, **rct**=reduction constant type, **V**=variable, **F**=fixed, **kar**=karatsuba depth.

Performance on ASICs

Tessera vs straight line HLS code used by prior works

Curve	Prior Work		Fastest		Smallest	
	Latency (ns)	Area (mm^2)	Latency (ns)	Area (mm^2)	Latency (ns)	Area (mm^2)
BN254	63	5.10	27 (**2.33×**)	2.21 (**2.31×**)	35 (**1.80×**)	1.76 (**2.90×**)
BLS12-381	82	11.94	31 (**2.65×**)	4.65 (**2.57×**)	49 (**1.67×**)	3.88 (**3.08×**)
BLS12-377	82	11.70	31 (**2.65×**)	3.88 (**3.01×**)	44 (**1.86×**)	3.54 (**3.30×**)
MNT4753	169.5	45.89	51 (**3.32×**)	24.00 (**1.91×**)	68 (**2.49×**)	14.43 (**3.18×**)

Up to **3.32× faster** and **3.30× smaller** designs

Performance on FPGAs

PADD designs for BLS12-377 on Twisted Edwards curve

Design	Model	LUTs	Regs	DSPs	Cycles	Fmax (MHz)	Latency (µs)
Fastest	VU9P	313,438	262,880	4,361	50	291	0.172
Min-DSP	VU9P	498716	369423	2,436	52	259	0.201
Balanced	VU9P	416,597	289,739	3,402	53	277	0.191
High-Fmax	VU9P	505,704	437,901	4,032	75	408	0.184
Fastest	VH1782	344,658	296,964	4,592	51	308	0.166
Min-DSP	VH1782	569,452	417,245	1,512	69	241	0.286
Balanced	VH1782	355,861	289,557	2,898	65	279	0.233
CycloneMSM [1]	VU9P	310,717	337,944	2,268	96	250	0.384
HardcamlMSM [20]	VU9P	–	–	–	238	278	0.856
BSTMSM [26]	U250	–	–	–	260	300	0.867

Summary

- We present Tessera an open-source hardware framework for exploring cryptographic kernels
- A foundational library of kernels with targeted optimizations
- Generates RTL for both ASIC and FPGA
- Lowers the barrier to entry for cryptographic hardware research

Thank you!
Check out Tessera!

https://github.com/cryptolets/cryptolets

NYU TANDON

Q&A