LaZer: a Lattice Library for Zero-Knowledge and Succinct Proofs

Vadim Lyubashevsky, Gregor Seiler, Patrick Steuer

IBM Research Europe, Zurich

Quantum-Safe Zero-Knowledge (state of the art)

The only concretely-efficient ZK SNARKs used today are hash-based

Up until a few years ago, even linear-size lattice-based proofs were not concretely efficient

But lattice-based proofs can be more compact and more efficient

… especially for proving “lattice relations” like As=t

Lattices can give the best of all worlds – quantum-safe and the most efficient

Progress on Lattice ZK

- Linear-Size Proofs [… , LPS' 22+some improvements]
  - Useful when witnesses are "short"
  - e.g. Ring Signatures, Kyber key well-formedness, Anonymous credentials in a (few) dozen KB
  - Results in quite efficient lattice-based privacy protocols

- SNARKS [..., BS '23]
  - Any circuit has proof size 40 – 60KB. Very slow asymptotic growth
  - Seems to give the most space-efficient quantum-safe multi-party (e.g. threshold) crypto

Applications

	LaBRADOR	LNP ‘22
Key / Ciphertext Well-Formedness		✓
Privacy-Based Crypto	✓	✓
Threshold Crypto	✓	
General Circuits	✓	

Lattice ZK in Privacy Protocols

ZK proofs are a crucial part of the protocols for:

- Blind signatures
- Anonymous credentials
- NIKE
- Threshold signatures
- …

Problem: ZK is very complicated to implement

Efficient parametrization of the ZK proof depends on the instance

- ZK proof consists of many parts
- Each part requires its own parameter set
- All the parameters are intertwined, many trade-offs

Non-trivial to do manually even for experts

For previous schemes, it was redone every time

In the newer schemes, it’s simply not done

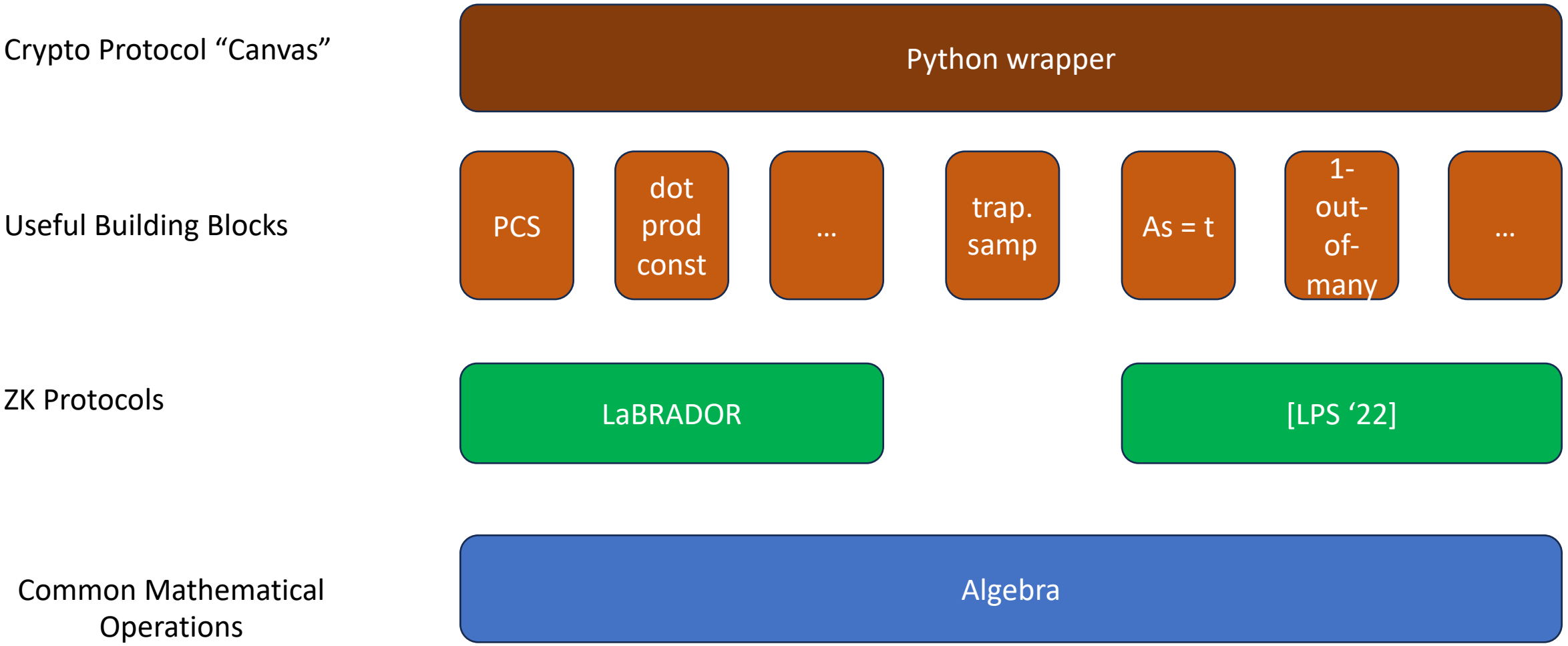
Goals of LaZer

1. Simple to Use:
   - No understanding of Lattice ZK proofs necessary
   - Specify the relation, and just call the ZK proof
   - Can write full protocols using the easy-to-use python wrapper
   - The ZK parameters are automatically figured out “under the covers”

2. Universal and Modular:
   - Should be able to prove any relation – ZK proofs automatically adjust to the instance
   - Allow for (efficiency) improvements to be easily added to the library

3. Efficient:
   - The flexibility and ease-of-use should only have a minor effect on efficiency

LaZer


Crypto Protocol “Canvas”
Python wrapper
Useful Building Blocks
PCS
dot prod const
…
trap. samp
As = t
1-out-of-many
…
ZK Protocols
LaBRADOR
[LPS ‘22]
Common Mathematical Operations
Algebra

Blind signatures

user(pk), signer(sk), verifier(pk)

- user, signer run protocol s.t user obtains a signature on their message
- verifier can verify the signature on the message

Properties:

1. correctness
2. anonymity/unlinkability: signer cannot link signatures to interactions
3. one-more unforgability: a user running the protocol n times cannot obtain more than n signatures

Blind signatures from lattices

Need two ZKPs, each proving knowledge of a "short" solution w to a linear relations of the form

$$\mathbf{Aw + t = 0}$$

"short": for a partition of w, each part is either bounded in l_2-norm or has binary coefficients only

statement: (A,t) + shortness, witness: w

shortness constraints are fixed

example: parameter specification for codegen

psteuer@li-83147acc-2e21-11b2-a85c-f46fa6c726ce:~/github/lazer

```
# Create a header file with proof system parameters for
# proving knowledge of a witness w in Rp^n (Rp = Zp[X]/(X^d + 1))
# such that
#
#   1. w satisfies a linear relation over Rp: Aw + u = 0
#   2. each element in a partition of w either ..
#      2.1 has binary coefficients only
#      2.2 satisfies an l2-norm bound

from math import sqrt

vname = "p2"  # variable name

deg   = 512    # Rp degree
mod   = 12289    # Rp modulus
dim   = (1,5)    # dimensions of A in Rp^(m,n)

wpart = [[0,1], [2],          [3,4]]  # partition of w
wl2   = [  109,   0, sqrt(34034726)]  # l2-norm bounds on parts of w
wbin  = [    0,   1,              0]  # binary coeffs condition on parts of w

wlinf = 5833 # optional: linf-norm bound on w
~
:wq
```

code example: proving knowledge of a signature

psteuer@li-83147acc-2e21-11b2-a85c-f46fa6c726ce:~/github/lazer/python

```python
    def unblind(self, blindsig: bytes):
        tau, s1, s2 = bytes(64), poly_t(Rp), poly_t(Rp)

        # [...] decode blindsig -> tau, s1, s2

        A = polymat_t(Rp, 1, 5, [Ar1, Ar2, Atau, -B1, -self.B2])
        u = polyvec_t(Rp, 1, [Am * self.m])
        w = polyvec_t(Rp, 5, [self.r1, self.r2, poly_t(Rp, tau), s1, s2])

        self.p2_prover.set_statement(A, u)
        self.p2_prover.set_witness(w)
        p2 = self.p2_prover.prove()

        sig = p2
        return sig
```

:w

code example: proof verification

psteuer@li-83147acc-2e21-11b2-a85c-f46fa6c726ce:~/github/lazer/python

```
    def verify(self, msg: bytes, sig: bytes):
        m = poly_t(Rp, msg)
        p2 = sig

        A = polymat_t(Rp, 1, 5, [Ar1, Ar2, Atau, -B1, -self.B2])
        u = polyvec_t(Rp, 1, [Am * m])

        self.p2_verifier.set_statement(A, u)
        self.p2_verifier.verify(p2)
```

:w

Current State – LPS'22

- privacy protocols (using [LPS '22]):
  - e.g. blind signatures / anonymous credentials / well-formedness proofs
    - Have simple python implementations
    - All have fairly small proofs (smallest of all the quantum-safe constructions)
    - Acceptable runtime, but can be greatly optimized – (a 100x speedup should be possible)

	~ proof size (KB)	~ prover runtime (ms)
blind signature	27	420
anonymous credentials	29	610
Kyber1024	19	190

LaBRADOR SNARK

- Recursive folding protocol
- Small concrete proof size of about 60 KB (optimized params)
- Fully vectorized polynomial arithmetic using AVX-512 instructions
- Multimodular 16-bit arithmetic via explicit CRT
- Multiprecision arithmetic with 14-bit limbs
- Fast binary matrix multiplication for Johnson-Lindenstrauss projection using Four Russians algorithm and in-register shuffles for table lookups

Chihuahua Frontend

Dot-product constraints and norm constraint on witness $s = (s_1, \dots, s_r)$

$$\sum_{i,j} a_{i,j} \langle s_i, s_j \rangle + \sum_i \langle \varphi_i, s_i \rangle + b = 0$$

$$\sum_i \|s_i\|^2 \leq \beta$$

Dachshund Frontend

- Same dot-product constraints as in Chihuahua
- But individual norm constraints on all witness vectors (l_2-norm or binary)

$$\sum_{i,j} a_{i,j}\langle s_i, s_j\rangle + \sum_i \langle \varphi_i, s_i\rangle + b = 0$$

$$\|s_i\|^2 \leq \beta_i \text{ for all } i$$

Greyhound Frontend

- Polynomial commitment scheme joint work with Khanh Nguyen
- Transparent setup
- Sublinear verifier complexity (square root)
- Faster proving time than STARKs
- Smaller proof sizes than STARKs

Dachshund Example: Signature Aggregation

- Prove many Falcon signatures for same message under different public keys

No of signatures	100	1’000	10’000
Size	84.82 KB	81.2 KB	83.91 KB
Prover Time	102 ms	590 ms	5.56 s
Verifier Time	81 ms	396 ms	3.22 s

Greyhound Runtimes

- Commit to polynomial f of degree n in $\mathbb{Z}_q[X]$ for q $\approx 2^{32}$
- Prove evaluation of f at $x \in \mathbb{Z}_q$, $f(x) = y$

Degree n	2^{20}	2^{22}	2^{24}	2^{26}	2^{28}
Proof Size	74 KB	74 KB	79 KB	77 KB	91 KB
Commit Time	0.0497 s	0.249 s	1.199 s	6.77 s	23.5 s
Prove Time	0.107 s	0.272 s	0.771 s	2.65 s	8.47 s
Verify Time	0.072 s	0.153 s	0.318 s	0.676 s	1.28 s

Comparison to STARKs for $n = 2^{28}$

Scheme	Size	Commit	Prove	Verify
Brakedown-PC	93767 KB	150 s	13 s	2.56 s
Ligero-PC	14383 KB	169 s	12.4 s	0.402 s
Greyhound	**91 KB**	**23.5 s**	**8.47 s**	**1.28 s**

THANK YOU!!