WP-031
June 2026
12 sections

ZK CIRCUIT RESEARCH & POST-QUANTUM CRYPTOGRAPHY

ZK Post-Quantum Hybrid Key Exchange: ML-KEM-768 in ZK Circuits with SNARK Verification

ML-KEM secures TLS handshakes. ML-DSA signs documents. Neither operates inside a zero-knowledge circuit. This paper presents the architecture that combines both.

AffixIO Research | June 2026 | Download PDF

ABSTRACT

The NIST post-quantum cryptography standards (FIPS 203 ML-KEM, FIPS 204 ML-DSA) have seen rapid deployment in transport protocols, particularly TLS 1.3 hybrid key exchange. This deployment pattern treats PQC algorithms as black boxes within existing protocol stacks. It does not enable a fundamentally different capability: zero-knowledge proofs about PQC operations. This paper presents the architecture for expressing ML-KEM-768 and ML-DSA operations as arithmetic circuit constraints, enabling SNARK proofs that attest to the correct execution of post-quantum key encapsulation and signature verification without revealing private keys or plaintext messages. We describe the key engineering challenge (the prime mismatch between ML-KEM's working

modulus q = 3,329 and the BN254 field prime), circuit design patterns for Number Theoretic Transform operations in R1CS, the hybrid SNARK+PQC proving construction (Groth16 privacy with ML-DSA-65 post-quantum binding), a ZK proof of ML-KEM key validity without private key disclosure, and the application to verifiable PQC inference in AI governance contexts. Specific circuit implementations are omitted from public documentation; this paper describes the architecture and engineering approach.

CONTENTS

SECTION 01

The Gap Between PQC and ZK

Post-quantum cryptography and zero-knowledge proofs are the two most significant active deployments in applied cryptography in 2026. They have evolved largely in parallel, with minimal intersection. The PQC deployment is primarily a protocol-level substitution: replacing ECDH with ML-KEM in TLS key exchange, replacing ECDSA with ML-DSA in certificate signing. The ZK

deployment is primarily an application-level proof system: proving membership in sets, proving eligibility criteria, proving computational integrity. Neither community has substantially engaged with the other's core problem.

The gap is this: when ML-KEM is used in a TLS handshake, the two parties who complete the handshake know that the key exchange was successful. A third-party observer knows only that some TLS handshake occurred. There is no mechanism for one of the parties to prove, to a verifier who was not part of the handshake, that a specific ML-KEM operation was performed correctly, without revealing the private key or the shared secret. Similarly, when ML-DSA signs a document, the signer proves to anyone with the public key that the signature is valid over that document. There is no mechanism for the signer to prove that they hold a valid ML-DSA signing key without producing a signature, and no mechanism to prove that a signature is valid without revealing the signed document.

These capabilities require ZK circuits that implement ML-KEM and ML-DSA operations as arithmetic constraints. With such circuits, the following statements become provable without revealing private inputs:

- "I hold an ML-KEM private key corresponding to this public key." (Proof of key validity without key disclosure.)
- "I correctly decapsulated this ML-KEM ciphertext and the shared secret matches this commitment." (Proof of decapsulation correctness without revealing the shared secret.)
- "There exists a valid ML-DSA signature over some document satisfying predicate P under this public key." (Proof of signature existence without revealing the document.)
- "This ML inference was performed correctly and the model's output was signed with a valid ML-DSA key." (Verifiable PQC inference.)

These are not merely academic curiosities. They correspond to real use cases in identity, compliance, and AI governance: post-quantum identity credentials whose holder can prove possession without disclosing the credential; post-quantum AI governance records where the model's output is verifiably signed and the signing can be audited without accessing the output; and post-

quantum zero-knowledge compliance proofs where the underlying data remains confidential. The combination of ZK and PQC at the circuit level is the enabling technology for all of these.

This paper describes the engineering architecture for building ZK circuits around ML-KEM-768 and ML-DSA operations. It is explicit about what is known, what is in development, and what remains open. Specific circuit implementations are omitted from public documentation; the paper describes the architecture and the research approach in sufficient detail for the research community to understand the problem space and the direction of work.

SECTION 02

ML-KEM-768 and Arithmetic Circuits

ML-KEM-768 (NIST FIPS 203, Module-Lattice-based Key Encapsulation Mechanism at security level 3) operates over the polynomial ring R_q = Z_q[x]/(x^256 + 1), where q = 3,329 is a prime and the polynomials have degree at most 255. The algorithm involves the following primitive operations:

- **Polynomial addition and subtraction modulo q:** coefficient-wise addition and subtraction in Z_q.
- **Polynomial multiplication modulo (x^256 + 1) and q:** schoolbook multiplication followed by reduction, or equivalently Number Theoretic Transform (NTT) multiplication.
- **NTT (Number Theoretic Transform):** the analogue of the Fast Fourier Transform over Z_q, enabling fast polynomial multiplication.
- **Compression and decompression:** lossy rounding functions that reduce the precision of polynomial coefficients from full Z_q representation to fewer bits.
- **Sampling:** sampling polynomials from specific distributions (the centred binomial distribution for small-norm vectors; uniform from a hash).
- **Hashing:** SHA3-256, SHA3-512, SHAKE-128, SHAKE-256 for key derivation and pseudorandom generation.

A Rank-1 Constraint System (R1CS), the constraint model used by Groth16 and many other SNARKs, represents computations as triples (A, B, C) of matrices such that (A · w) * (B · w) = C · w for witness vector w. Addition constraints are "free" (they do not increase constraint count); multiplication constraints each cost one R1CS constraint. The primary cost metric for a ZK circuit is therefore the number of multiplication constraints (gates).

The most expensive operations in ML-KEM-768, when expressed as R1CS constraints, are: the NTT butterfly operations (each involving modular multiplications and modular additions); the SHA3/SHAKE hash functions (SHAKE-128 produces the highest constraint counts due to the Keccak permutation's bitwise operations being expensive in a field-based constraint system); and the compression/decompression operations (which require range proofs).

Expressing ML-KEM-768's full key generation, encapsulation, and decapsulation in R1CS produces circuits with constraint counts in the range of several million to tens of millions of constraints, depending on implementation choices. This is large but not unmanageable by current SNARK prover standards: Groth16 provers on 2025-generation hardware can handle circuits of 10 to 100 million constraints with proving times of seconds to minutes. The more important design question is which subset of ML-KEM operations needs to be proved, because in most practical use cases full key generation, encapsulation, and decapsulation do not all need to be in the same circuit.

SECTION 03

The Prime Mismatch Problem

The most fundamental engineering challenge in building ML-KEM ZK circuits is the prime mismatch. ML-KEM-768 performs arithmetic modulo q = 3,329. Groth16 over BN254 performs arithmetic over a field with prime:

```
p = 21888242871839275222246405745257275088548364400416034343698204186575808495617
```

This is a 254-bit prime. In the BN254 field, every arithmetic operation automatically reduces modulo p. This means that additions and multiplications in the BN254 field are not the same as additions and multiplications modulo 3,329 unless additional constraints are added to enforce the smaller modulus.

Consider a simple ML-KEM coefficient addition: a + b mod 3,329. In the BN254 field, computing a + b gives the correct integer sum (since both a and b are at most 3,328, their sum is at most 6,656, which is much smaller than p and does not reduce modulo p). However, reducing this sum modulo 3,329 is not automatic: the circuit must explicitly compute the quotient q_val = floor((a + b) / 3329) and the remainder r = (a + b) - 3329 * q_val, and add constraints asserting that: (1) q_val is either 0 or 1 (since a + b < 2 * 3,329); (2) r = (a + b) - 3329 * q_val; and (3) r is in the range [0, 3,328].

This pattern, in which a single ML-KEM modular reduction requires multiple R1CS constraints and range proofs, is the core cost driver of ML-KEM circuit design. There are several strategies for managing this cost:

Strategy 1: Lazy Reduction

Defer modular reductions until the accumulated value would overflow the BN254 field. Since p is approximately 2^254 and q = 3,329, the maximum accumulated value before overflow is approximately p / q ≈ 2^241. This means up to about 2^241 ML-KEM additions can be performed before a modular reduction is necessary without risk of field overflow. In practice this allows many additions to be batched with a single reduction at the end, significantly reducing constraint counts for linear operations. Multiplications cannot use lazy reduction in the same way, because the product of two ML-KEM coefficients (each up to 3,328) is at most about 11 million, and intermediate values in NTT multiplications grow if not reduced promptly.

Strategy 2: Field-Friendly NTT Decomposition

The NTT over Z_q can be decomposed into operations that are more naturally expressed in the BN254 field. Specifically, the NTT butterfly operation a, b ↦ a + ω·b, a - ω·b (where ω is a primitive root of unity in Z_q) can be implemented with a multiplication (for ω·b), two additions, and two range checks. When

unrolled across all 256 NTT layers and 128 butterfly operations per layer, the total constraint count for one NTT transform is on the order of 50,000 to 100,000 constraints, depending on implementation.

Strategy 3: Native Modular Arithmetic Lookup Tables

Some ZK proof systems (Plonkish systems with lookup arguments, such as UltraPlonk or Halo2) support lookup table constraints that can implement small-modulus arithmetic more efficiently than pure R1CS. A table of all pairs (a, b, a + b mod q) for a, b in Z_q has 3,329^2 ≈ 11 million entries, which is tractable as a lookup table in a Plonkish system. This approach can reduce the constraint overhead of modular arithmetic by a factor of 2 to 5 compared to pure R1CS, at the cost of a larger trusted setup or more complex constraint system.

The choice between these strategies involves a trade-off between constraint count, prover complexity, trusted setup requirements, and backend compatibility. R1CS-based circuits using lazy reduction are the most portable and compatible with the widest range of proof systems. Plonkish circuits with lookup tables are more efficient for ML-KEM specifically but less portable. The architecture described in this paper is backend-agnostic at the design level; specific backend choices depend on the proving performance requirements of the application.

SECTION 04

NTT in R1CS: Circuit Architecture

The Number Theoretic Transform is central to efficient polynomial multiplication in ML-KEM. The NTT over Z_q for degree-256 polynomials operates in seven layers (since 256 = 2^8 and the NTT for ML-KEM uses a specific set of roots of unity compatible with q = 3,329). Each layer consists of 128 butterfly operations.

A single NTT butterfly at layer l with twiddle factor ω is:

```
a' = a + ω · b  mod q
b' = a - ω · b  mod q
```

In R1CS, this requires: one multiplication constraint for ω · b; two addition constraints for the sums (free in R1CS); and two range constraints asserting a' and b' are in [0, q-1], each requiring a logarithmic-depth range check (approximately 12 constraints for range [0, 3,328] using binary decomposition).

Total constraints per butterfly: approximately 1 multiplication + 24 range check constraints = approximately 25 constraints. Over 7 layers of 128 butterflies, a single NTT forward transform requires approximately 7 × 128 × 25 = 22,400 constraints. The inverse NTT requires the same, with an additional division by 256 modulo q (a multiplication by 256^{-1} mod q = 3,303, adding one further constraint). Two NTT transforms plus a pointwise multiplication (256 multiplications modulo q, each requiring a range check) gives approximately 50,000 to 60,000 constraints per degree-256 polynomial multiplication in the NTT domain.

ML-KEM-768 uses 3 x 3 matrix-vector multiplications over degree-256 polynomials (the matrix A in the key generation and encapsulation steps). This requires 9 polynomial multiplications, giving approximately 9 × 55,000 = 495,000 constraints for the main matrix-vector product. The remaining operations in encapsulation (sampling, compression, hashing) add further constraints, with the SHA3/SHAKE hashing contributing the largest additional cost due to the Keccak permutation.

Keccak in R1CS

ML-KEM uses SHAKE-128 and SHAKE-256 (Keccak-based XOFs) for its pseudorandom number generation (the matrix A is derived from a seed using SHAKE-128) and for key derivation. The Keccak-f[1600] permutation involves bitwise AND, XOR, and rotation operations that are expensive in field-based constraint systems. A single Keccak-f[1600] call requires approximately 150,000 to 250,000 R1CS constraints depending on implementation and field size. ML-KEM-768 encapsulation invokes SHAKE-128 once (for the matrix) and

uses SHA3-256 twice (for hashing in the KEM construction). This contributes approximately 500,000 to 750,000 additional constraints to the encapsulation circuit.

Total Constraint Estimate

A complete ML-KEM-768 encapsulation proof (proving that ciphertext ct was correctly produced from public key pk and randomness r, producing shared key commitment K) requires approximately 1.5 to 3 million R1CS constraints, depending on implementation choices and hash function handling. This is within the range of practical Groth16 proving times: at 2-10 million constraints per second (on a 2025-generation server CPU), the proving time is approximately 0.3 to 1.5 seconds. Trusted setup SRS size for a circuit of this scale is approximately 200 to 500 MB.

SECTION 05

ML-KEM Encapsulation Proof

The ML-KEM encapsulation proof demonstrates, to a verifier who was not party to the original encapsulation, that a specific ciphertext was correctly produced from a specific public key. The formal statement being proved is:

> **ML-KEM ENCAPSULATION STATEMENT**
>
> There exists randomness r and shared secret K such that: (1) (ct, K) = ML-KEM.Encaps(pk, r); (2) H(K) = k_com (the public key commitment); where pk is the public key (a public input) and ct is the ciphertext (a public input). The witness is (r, K).

This proof establishes that the ciphertext is a valid ML-KEM encapsulation under the given public key, without revealing the randomness r or the shared secret K. The prover holds r and K as private witnesses. The verifier receives (pk, ct, k_com) as public inputs and the Groth16 proof as the output.

Use Cases

The ML-KEM encapsulation proof enables several applications that are not possible with raw ML-KEM usage:

Post-quantum key escrow without key disclosure: A regulated entity can prove to an auditor that a specific ciphertext was correctly encapsulated (establishing that the corresponding decapsulation would yield a valid key) without disclosing the key material. The auditor verifies the proof against the public key and ciphertext; they learn that key establishment was performed correctly, not what the key is.

Post-quantum credential issuance proof: A credential issuer using ML-KEM to encrypt a credential to a holder can prove to a third-party auditor that the credential was correctly encrypted to a specific holder public key, without revealing the credential content. This supports audit of credential issuance programmes without creating a secondary disclosure of the credential data.

Post-quantum anonymous credential presentation: A credential holder can prove that they correctly decapsulated a credential (demonstrating they hold the corresponding private key) without revealing the private key or the credential. This is the post-quantum analogue of a Schnorr proof of discrete logarithm knowledge, applied to the ML-KEM key relationship.

Decapsulation Proof

The ML-KEM decapsulation proof is the complementary statement: there exists a private key sk such that ML-KEM.Decaps(sk, ct) = K, where ct, pk, and H(K) are public inputs and sk is the private witness. This is strictly harder to implement efficiently than the encapsulation proof, because it requires expressing the full decapsulation algorithm (including the implicit rejection mechanism that compares re-encapsulation with the original ciphertext) as circuit constraints. The implicit rejection adds branch logic that is expensive in R1CS; this is an active area of circuit optimisation research.

SECTION 06

ML-DSA Signature Verification Circuit

ML-DSA (NIST FIPS 204, Module-Lattice-based Digital Signature Algorithm) provides post-quantum digital signatures. The ML-DSA.Verify algorithm takes as input a public key pk, a message M, and a signature sig, and returns accept or reject. Expressed as a ZK circuit, the verify algorithm allows a prover to demonstrate that a valid ML-DSA signature exists over some message without revealing the message itself.

ML-DSA-65 (NIST security level 3) uses the following parameter structure: public key pk encodes a matrix A (derived from a seed) and a vector t = As + e (the public commitment); the signature sig encodes a vector z (a response vector), a polynomial c (a challenge), and hints h. The verify algorithm checks: (1) the challenge c is correctly computed as the hash of (A, t, μ, w1), where w1 is reconstructed from z and the public key; (2) the infinity norm of z is below a threshold; (3) the hint vector h correctly decompresses w0.

Circuit Structure for ML-DSA Verify

The ML-DSA.Verify circuit takes as public inputs the verification key (the seed of A and the vector t) and the signature components (z, c, h). The message M is a private input (the statement being proved is "there exists a message M satisfying predicate P such that sig is a valid ML-DSA signature over M"). The circuit verifies all the ML-DSA verification checks in constraints.

The dominant cost components are:

- **Matrix-vector multiplication:** computing Az requires 3 x 3 degree-256 polynomial multiplications, same structure as in ML-KEM, approximately 500,000 constraints.
- **Norm check:** verifying that the infinity norm of z is below the threshold requires 252 range checks (one per coefficient of z, which has 252 coefficients for ML-DSA-65), each requiring logarithmic range proof constraints. Approximately 30,000 constraints.
- **Hash verification:** computing the Keccak-based hash for the challenge reconstruction requires one or two Keccak-f[1600] invocations, approximately 200,000 to 400,000 constraints.

- **Hint processing:** applying the hint vector h requires modular rounding operations, approximately 50,000 constraints.

Total constraint estimate for ML-DSA-65 verification: approximately 1 to 2 million R1CS constraints, slightly lower than ML-KEM-768 encapsulation due to the absence of the pseudorandom matrix sampling step (the matrix seed is a public input in the verify circuit).

The Privacy Gain

The ZK ML-DSA verify circuit enables a prover to establish: "There exists a document M satisfying predicate P (for example, M is a valid JSON payload with field value x above threshold t) such that the ML-DSA signature sig is valid over M under public key pk." The verifier learns that such a document exists, satisfies the predicate, and is validly signed. The verifier does not learn M. This is a qualitatively different capability from standard ML-DSA: a verifier can now be convinced of facts about the signed content without receiving the content.

SECTION 07

Hybrid SNARK+PQC Proving Backend

The hybrid proving backend combines a classical ZK proof system (Groth16 SNARK) with a post-quantum signature (ML-DSA-65) to produce a proof transcript that has both zero-knowledge privacy and post-quantum binding. Neither component alone provides both properties.

Properties of Each Component

PROPERTY	GROTH16 SNARK ALONE	ML-DSA-65 ALONE	GROTH16 + ML-DSA-65 HYBRID
Zero-knowledge (private inputs hidden)	Yes (classical)	No	Yes (classical)
Soundness (false proofs rejected)	Yes (classical)	N/A	Yes (classical)
Post-quantum binding (anchor unforgeable by quantum adversary)	No (pairings quantum-vulnerable)	Yes (MLWE-based)	Yes (ML-DSA component)
Proof size	~192 bytes	~3,293 bytes sig + 1,952 bytes pk	~192 + 3,293 + 1,952 bytes = ~5.4 KB total
Verifier requirements	Verification key (classical)	ML-DSA public key	Both verification key and ML-DSA public key

The hybrid transcript for a single proof event is constructed as follows:

1. The prover runs the ZK circuit (ML-KEM encapsulation, ML-DSA verification, or eligibility predicate) to generate Groth16 proof π and public inputs x.
2. The proof hash h = SHA-256(π || x) is computed.
3. The proof hash is submitted to the AffixIO attestation service, which signs h with ML-DSA-65 to produce signature σ = ML-DSA.Sign(sk_AffixIO, h).
4. The complete hybrid transcript is (π, x, σ, pk_AffixIO).
5. A verifier verifies: (a) Verify(vk, x, π) = accept; (b) ML-DSA.Verify(pk_AffixIO, h, σ) = accept. Both checks must pass.

This construction is secure against a classical adversary under Groth16 soundness: a false proof cannot be produced without breaking the Groth16 knowledge soundness property. It is secure against a quantum adversary who can break Groth16 (by solving the discrete logarithm in the BN254 pairing groups) because such an adversary, having constructed a false proof π', would need a valid ML-DSA-65 signature on SHA-256(π' || x), which they cannot obtain without access to the AffixIO signing key and which they cannot forge

without breaking ML-DSA-65. The combined construction is therefore secure against quantum adversaries as long as ML-DSA-65 remains unbroken, which is the current consensus expectation.

Transcript Binding

The proof hash h = SHA-256(π || x) binds the ML-DSA signature to the specific proof and public inputs. A verifier who receives (π, x, σ) can check that the signature is over the correct proof without trusting the prover to have computed h correctly. The hash is a short, fixed-size commitment that the signing service computes from the presented proof transcript, removing the need for the signing service to understand the ZK circuit semantics.

SECTION 08

ZK Proof of PQC Key Validity

A particularly important application of ML-KEM ZK circuits is proving knowledge of a valid ML-KEM private key without disclosing the key. This is the post-quantum analogue of a Schnorr proof of discrete logarithm knowledge, which underlies most classical ZK identity schemes.

In classical systems, a party holding a private key sk corresponding to public key pk = g^sk can prove knowledge of sk by running a Schnorr identification protocol: commit to a random r, receive challenge c from the verifier, respond with z = r + c·sk. The verifier checks g^z = commit · pk^c. This proves knowledge of the discrete logarithm without revealing it.

For ML-KEM, the key relationship is pk = (A, t) where t = A·s + e and the private key is (s, e) (a pair of small-norm polynomial vectors). Proving knowledge of (s, e) satisfying t = A·s + e with small norms is exactly a Module Learning With Errors (MLWE) witness claim, and it can be expressed as an R1CS circuit: the circuit verifies that the witness (s, e) satisfies the linear relation with A and t (requiring the matrix-vector product circuit) and that all coefficients of s and e are within the specified norm bounds (requiring range checks).

> ML-KEM KEY VALIDITY STATEMENT

> There exist polynomial vectors s, e with coefficients in [−η, η] (for ML-KEM-768, η = 2) such that t = A·s + e in R_q, where A (derived from seed ρ) and t are public inputs derived from pk. The witness is (s, e).

This circuit proves that the prover holds the ML-KEM private key corresponding to the stated public key, without disclosing the private key. It does not require performing an actual encapsulation or decapsulation; it simply verifies the key consistency relationship. The constraint count is approximately that of one matrix-vector multiplication (500,000 constraints for the A·s product) plus range checks for the coefficients of s and e (approximately 500 coefficients × 12 constraints per range check = 6,000 constraints). This gives a substantially smaller circuit than the full encapsulation proof, in the range of 600,000 to 800,000 constraints.

Nullifier Construction

A ZK proof of ML-KEM key validity can be combined with a nullifier construction (as described in WP-014 for double-spend prevention) to create a post-quantum anonymous credential system with stateless double-use prevention. The nullifier is computed as H(s || context), where context is a domain-specific value (for example, a session identifier or timestamp range). The nullifier is published; the circuit proves that the nullifier was computed from the private key s without revealing s. A spent registry records used nullifiers; a repeat attempt with the same key in the same context produces a duplicate nullifier, which is detected without revealing which key produced it.

SECTION 09

Performance: Constraints and Proving Time

The following table summarises the estimated constraint counts and proving times for the circuits described in this paper, alongside comparisons with other ZK circuit benchmarks for context. Estimates are based on the engineering analysis in Sections 3 and 4; production implementation details may vary.

CIRCUIT	APPROX. R1CS CONSTRAINTS	EST. GROTH16 PROVING TIME (2025 SERVER CPU)	PROOF SIZE
ML-KEM-768 key validity	~600K-800K	~0.3-0.5s	~192 bytes
ML-KEM-768 encapsulation proof	~1.5M-3M	~0.7-2s	~192 bytes
ML-DSA-65 verify circuit	~1M-2M	~0.5-1.5s	~192 bytes
ML-KEM + ML-DSA combined	~3M-5M	~1.5-4s	~192 bytes
Groth16 + ML-DSA-65 anchor (hybrid transcript)	As above + signing	As above + ~5ms signing	~5.5 KB
Reference: SHA-256 in Groth16 (ZCash Sapling)	~26K per hash	<0.1s	~192 bytes
Reference: RSA-2048 verify in Groth16	~400K	~0.2s	~192 bytes
Reference: Ed25519 verify in Groth16	~3K-5K	<0.01s	~192 bytes

The key observation from this table is that ML-KEM and ML-DSA circuits are 100 to 300 times more expensive than classical signature verification circuits in terms of constraint counts. This reflects the fundamental difference in computational structure: classical elliptic curve operations involve small field arithmetic over the native proof system field, while lattice-based operations involve polynomial arithmetic over a foreign modulus ($q = 3,329$) that must be simulated within the proof system field.

The proving times, however, remain in the sub-second to low-seconds range, which is practical for many compliance and identity applications. For high-throughput scenarios, parallel proving on multiple cores or GPU-accelerated SNARK provers reduce proving times by one to two orders of magnitude. The proof output size is constant at approximately 192 bytes for Groth16,

regardless of circuit size, which is a major advantage: the verifier always processes the same small proof regardless of the complexity of the underlying computation.

Plonkish vs R1CS

UltraPlonk and Halo2-style Plonkish proof systems with lookup arguments can implement ML-KEM circuits more efficiently than pure R1CS, with estimated constraint savings of 3 to 5 times for the modular arithmetic components. The trade-off is larger proof size (approximately 400 bytes to 4 KB depending on the number of columns and polynomial commitments) and different trusted setup requirements. For applications where proof size matters, R1CS/Groth16 is preferred; for applications where proving time is the binding constraint, Plonkish systems with lookups are more efficient.

SECTION 10

Verifiable PQC Inference

The combination of ZK machine learning inference (ZKML) with post-quantum signatures opens a qualitatively new category of AI governance: verifiable PQC inference, where the output of an AI model can be proved to have been generated by a specific model, signed with a quantum-resistant key, without revealing the input data or model weights.

The ZKML Layer

ZKML (zero-knowledge machine learning) uses ZK circuits to prove that a neural network inference was performed correctly: given a committed model weight set and a public output, the circuit proves there exist inputs satisfying specific predicates that produce that output under the specified model. Tools including EZKL (Ethereum Zero-Knowledge for ONNX models) and Modulus Labs have demonstrated ZKML for models with millions of parameters, though with significant proving time costs for large models.

The PQC Signing Layer

The AI governance use case requires that the inference output is not merely proved correct (ZKML) but also post-quantum authenticated: a quantum-resistant signature over the inference output ensures that the signed result cannot be forged or backdated by a future quantum adversary. ML-DSA-65 provides this. The model operator signs the inference output hash with their ML-DSA-65 key; the signature is verifiable by anyone with the public key and remains valid against quantum attack.

The Combined Architecture

Verifiable PQC inference combines both layers:

1. The AI model performs inference on private input data, producing output O.
2. A ZKML circuit proves that inference on data satisfying predicate P produces output O under model M (identified by a weight commitment). The private inputs (the actual input data) are the ZK witnesses.
3. An ML-DSA-65 signature is produced over H(O || circuit_id || timestamp).
4. The hybrid transcript (ZKML proof, public output O, ML-DSA-65 signature) is submitted to AffixIO's Merkle-anchored audit registry.
5. A verifier can confirm: (a) the inference was correctly performed by the stated model (ZKML proof); (b) the result was ML-DSA-65 signed at a specific time (ML-DSA signature); (c) the result is in the audit registry (Merkle proof). None of these checks require the verifier to access the input data.

This architecture directly supports the EU AI Act's requirement for audit trails on high-risk AI decisions (Article 12) and the emerging US state AI law requirements for algorithmic accountability records (covered in WP-026), while maintaining the privacy of the individuals whose data informed the AI decision. It is an extension of the general AI governance framework described in WP-001 (Cryptographic AI Governance) with post-quantum signing replacing classical ECDSA anchoring.

SECTION 11

Security Analysis and Open Problems

Security of the Hybrid Construction

The hybrid SNARK+PQC construction provides the following security properties under standard assumptions:

- **Completeness:** If the prover holds a valid witness, the Groth16 proof will always be accepted and the ML-DSA-65 signature will always verify. This follows directly from the completeness of Groth16 and the correctness of ML-DSA.
- **Classical soundness:** A computationally bounded classical adversary without the proving key cannot produce an accepted proof for a false statement, under the knowledge soundness of Groth16 (which follows from the hardness of the discrete logarithm in the BN254 pairing groups).
- **Quantum binding:** A quantum adversary who can forge Groth16 proofs (by breaking the BN254 discrete logarithm with a quantum computer) still cannot produce a hybrid transcript that passes verification, because producing a valid ML-DSA-65 signature on the forged proof hash requires either access to the AffixIO signing key or breaking ML-DSA-65, neither of which is achievable by a quantum adversary under current assumptions.
- **Zero-knowledge:** The Groth16 zero-knowledge property ensures that the proof reveals nothing about the private witnesses (s, e for ML-KEM key validity; r, K for ML-KEM encapsulation; M for ML-DSA verification) beyond what the public inputs and the truth of the statement disclose.

Open Problem: Quantum Security of Groth16

As discussed in WP-030 (Stateless Post-Quantum Verification), Groth16's soundness relies on the knowledge of exponent assumption in pairing-friendly groups, which is quantum-vulnerable. A sufficiently powerful quantum computer could potentially forge Groth16 proofs. The ML-DSA-65 anchor mitigates the operational impact of this risk but does not eliminate it in principle. The complete solution is to replace Groth16 with a proof system whose soundness relies on quantum-hard assumptions: lattice-based SNARKs or hash-based STARKs.

Open Problem: Lattice-Based ZK Proofs for ML-KEM

Several recent papers propose ZK proof systems based on lattice hardness assumptions: Lyubashevsky's lattice-based ZK proofs; the Banquet protocol; the LaBRADOR framework (Bootle et al., 2023). These would provide a fully post-quantum ZK proof system that is a natural fit for ML-KEM circuit expressions, since both the proof system and the circuit operate over similar lattice structures. However, current lattice-based ZK proof sizes are significantly larger than Groth16 (typically tens to hundreds of kilobytes), and verification is more expensive. This is an active research area where rapid progress is expected.

Open Problem: Trusted Setup for ML-KEM Circuits

Groth16 requires a circuit-specific trusted setup. A trusted setup for an ML-KEM-768 encapsulation circuit (with 2 to 3 million constraints) requires generating a Structured Reference String (SRS) of corresponding size (approximately 60 to 90 MB at BN254 scale). The setup ceremony must involve sufficient participants to ensure security; existing ceremony frameworks (Zcash Powers of Tau, Ethereum Hermez ceremony) support circuits of this scale. However, each new ML-KEM circuit variant (encapsulation, decapsulation, key validity) requires a separate ceremony. Transparent proof systems (STARKs, DARK, FRI-based systems) avoid the trusted setup at the cost of larger proofs.

Open Problem: ML-KEM Decapsulation Circuit Efficiency

The implicit rejection mechanism in ML-KEM decapsulation (which re-encapsulates using the decapsulated key and compares with the original ciphertext) adds branch logic that is expensive in R1CS. Current estimates suggest the decapsulation circuit is 2 to 3 times larger than the encapsulation circuit. Optimisation approaches including branch-free constraint formulations and the use of non-deterministic hints to guide the decapsulation computation are active research directions.

SECTION 12

Conclusion

The deployment of NIST post-quantum cryptography standards in TLS, certificates, and document signing represents the first generation of the quantum transition. It treats ML-KEM and ML-DSA as black-box primitives within existing protocol stacks. The second generation, which this paper describes, treats these algorithms as computations that can be expressed as arithmetic constraints and proved in zero knowledge.

The engineering challenges are real. The prime mismatch between ML-KEM's working modulus (q = 3,329) and SNARK field primes requires explicit modular arithmetic in the constraint system, multiplying constraint counts by one to two orders of magnitude compared to classical curve operations. Hash function evaluation (Keccak for ML-KEM's PRG; SHA3 for key derivation) is expensive in R1CS. Full ML-KEM encapsulation proofs require 1.5 to 3 million constraints and proving times of 0.7 to 2 seconds on 2025-generation hardware, acceptable for many compliance applications but not for latency-sensitive interactive protocols.

The capabilities that ZK circuits over ML-KEM and ML-DSA unlock are correspondingly significant. Proofs of ML-KEM key ownership without key disclosure enable post-quantum anonymous credential systems. ML-DSA verification circuits enable proofs of signed-content predicates without content disclosure. The hybrid SNARK+PQC backend provides both zero-knowledge privacy (Groth16) and post-quantum binding (ML-DSA-65) in a single proof transcript. Verifiable PQC inference extends these properties to AI model outputs, enabling quantum-resistant AI governance records that satisfy GDPR and EU AI Act audit requirements without disclosing the underlying data.

The open problems are clearly identified: Groth16's quantum vulnerability requires a migration path to lattice-based or hash-based proof systems; the trusted setup requirement for ML-KEM circuits requires ceremony planning; the decapsulation circuit efficiency gap requires further research. Each of these is a tractable engineering problem with active work in the research community. The architecture described in this paper is designed to be proof-

system-agnostic in its upper layers, so that improvements in the underlying proof system (STARK proving backends, lattice-based SNARKs) can be adopted without redesigning the application-layer circuits.

The intersection of zero-knowledge proofs and post-quantum cryptography at the circuit level is the research frontier of both fields. The work described here represents the current state of AffixIO's investigation of this frontier; specific circuit implementations are the subject of ongoing engineering work and are omitted from public documentation. The architecture and the problem formulation are presented here as a contribution to the research community's understanding of what ZK PQC circuits can do, what they cost, and what remains to be solved.

> Related AffixIO whitepapers: WP-002: Post-Quantum Attestation covers ML-DSA-65 production deployment. WP-029: TLS 1.3 Hybrid PQC covers ML-KEM in transport protocols. WP-030: Stateless Post-Quantum Verification covers the hybrid SNARK+PQC proving model at the application layer. WP-016: ZKML Verifiable Inference covers the ZKML layer without PQC.

FREQUENTLY ASKED

Common Questions

Why implement ML-KEM inside a ZK circuit rather than just using ML-KEM directly?

Using ML-KEM directly in TLS proves nothing to a third-party observer: the session keys are shared between two parties with no auditable record of what was proved. A ZK circuit wrapping ML-KEM operations allows a party to prove to a verifier that they correctly performed a specific ML-KEM operation without revealing the private key or the encapsulated secret. This enables post-quantum identity credential systems where the holder proves knowledge of a valid ML-KEM private key without disclosing it, and creates auditable, verifiable PQC operations rather than simply performing them in private.

What is the prime mismatch problem in ML-KEM ZK circuits?

ML-KEM-768 operates modulo q = 3,329. Groth16 SNARKs operate in a field with a 254-bit prime. Modular arithmetic modulo 3,329 is not automatic in the SNARK field and must be enforced explicitly with range proofs and auxiliary constraints. This adds approximately 25 constraints per modular operation compared to essentially free operations in the native field, which is the main driver of the large constraint counts for ML-KEM circuits.

How does the hybrid SNARK+PQC backend differ from just using a PQC signature?

A standalone ML-DSA signature proves the signer held the private key and reveals which document was signed. A Groth16 ZK proof proves a computation was performed correctly without revealing private inputs. The hybrid system provides both: Groth16 gives zero-knowledge privacy; ML-DSA-65 gives post-quantum binding over the proof transcript. A quantum adversary who can forge Groth16 proofs cannot produce a validly ML-DSA-signed forged transcript without access to the signing key. Neither component alone provides both privacy and quantum resistance.

When will fully post-quantum ZK proofs be practical?

Lattice-based ZK proofs (LaBRADOR, Banquet, DPVW) and hash-based STARKs both provide security without relying on quantum-vulnerable assumptions. Current STARK proof sizes are 10 to 100 KB for ML-KEM-scale circuits, compared to 192 bytes for Groth16; current lattice-based SNARKs are similar or larger. Active research is reducing these sizes. A reasonable expectation is that practically-sized fully post-quantum ZK proofs for ML-KEM circuits will be available within two to three years as the proof system field matures.

WP-002: PQ Attestation | WP-029: TLS Hybrid PQC | WP-030: Stateless PQ Verification

▶ About

- ▶ Solutions
- ▶ Legal
- ▶ Trust & Security

Contact

truth layer | yes | no | proof