Sentient

A Proof of Zero Knowledge

Dylan Kawalec, Arvin Bhangu

February 14, 2025

Abstract

The ability to interact with digital assets privately has long been compromised by centralized gatekeepers, keyloggers, and surveillance systems. **Sentient: A Proof of Zero Knowledge** introduces a simple but effective method for proving control over a passkey without ever revealing it. By using zero-knowledge holy primes (*zkHPNG*), structured commitments (*Fullmen*), and the first cybernetic cryptographic proof system (*Secure Authentication Proof*), users can authenticate securely while keeping their private keys hidden, even under full surveillance.

This system is designed to work with any blockchain that supports compact proofs, including Bitcoin via Taproot. It enables a new form of decentralized identity in which individuals can sign transactions, authorize access, and prove ownership of digital assets without relying on third-party verifiers, biometric authentication, or traditional two-factor systems. Unlike conventional multisignature schemes or trusted key storage, this method requires no intermediaries and leaves no exploitable metadata behind.

By reducing the proof size to near-minimal levels and ensuring verification remains efficient, this protocol makes it feasible to deploy zero-knowledge authentication at scale. Provides a foundation for private human-machine interactions in decentralized networks, allowing individuals to maintain control over their assets without sacrificing security or privacy. This paper consolidates previous research, introduces key refinements, and outlines a practical framework for making private access to provable digital assets a reality.

1 Introduction

The ability to prove ownership and authorization without revealing private keys is essential for maintaining financial privacy and security in decentralized networks. The **zkHPNG-Fullmen-SAP** protocol introduces a way to authenticate without exposing sensitive information, using a combination of zero-knowledge proofs, structured commitments, and linked verification chains. This method allows users to prove control over a passkey, interact with smart contracts, and sign transactions under complete surveillance resistance.

Unlike traditional authentication systems, which rely on external authorities, custodians, or biometric databases, this protocol ensures that proof-of-knowledge remains private and independent of any centralized entity. It works with any blockchain that supports compact proofs, including Bitcoin via Taproot, making it possible to build private authentication systems directly on decentralized networks.

1.1 Motivation

Decentralized applications require privacy, efficiency, and security. The zkHPNG-Fullmen-SAP protocol addresses these needs by:

- Using **zero-knowledge proofs** (Groth16) [2] to verify statements without revealing the underlying data.
- Structuring commitments through **Fullmen**, ensuring immutable and verifiable authentication.
- Enabling **Secure Authentication Proofs (SAP)**, which allow proofs to be linked, aggregated, and efficiently validated.
- Ensuring **quantum resilience** through Holy Prime Number Generation (HPNG), strengthening cryptographic integrity.
- Maintaining **blockchain compatibility**, leveraging Bitcoin's Taproot and other decentralized systems to anchor the chain.

This system allows private interaction with digital assets, ensuring that keyloggers, surveillance systems, and third-party verifiers cannot compromise access or authentication. The goal is simple: allow users to prove control without revealing anything unnecessary, preserving privacy in an increasingly monitored digital landscape.

1.2 Scope and Objectives

This protocol is designed for one purpose: to enable users to prove what they know without revealing it. By combining zero-knowledge proofs with structured commitments and recursive verification, zkHPNG-Fullmen-SAP provides a secure and scalable framework for private authentication on decentralized networks.

This document outlines the following.

- The fundamental cryptographic principles governing the protocol, including its axioms, postulates, and logical structure.
- A mathematical breakdown of how **Groth16 zk-SNARKs** integrate with **HPNG-based commitments** to verify knowledge without exposure.
- A step-by-step guide to proof generation, validation, and anchoring on-chain using Bitcoin's Taproot or other decentralized systems.
- The role of **Secure Authentication Proofs (SAP)** in linking, aggregating, and validating knowledge proofs efficiently.

1.3 Key Features of zkHPNG-Fullmen-SAP

The protocol achieves its goals through four main components:

1. **Zero-Knowledge Holy Prime Number Generation (zkHPNG)** — A cryptographic method for generating and proving the existence of special primes where both P and $2P+1$ are prime, ensuring resilience against quantum attacks.
2. **Fullmen Protocol (Merkle Commitments)** — A tamper-proof commitment scheme that structures proofs into immutable Merkle trees, enabling efficient verification.

3. **Secure Authentication Proof (SAP)** — A mechanism that chains or aggregates proofs across multiple interactions, ensuring knowledge verification remains valid without requiring recursive SNARKs.

4. **Groth16 zk-SNARK Integration** — A succinct proof system that allows knowledge validation with constant-size proofs and near-instant verification.

By combining these elements, zkHPNG-Fullmen-SAP provides a simple yet powerful way to prove identity, control, or access without revealing any sensitive information—laying the foundation for decentralized authentication systems that are private, secure, and scalable.

2 Axioms and Postulates

The following axioms and postulates establish the logical and cryptographic foundation of the **zkHPNG-Fullmen-SAP** protocol.

2.1 Axioms Defining Cryptographic Integrity

Axiom 1: (Deterministic Prime Generation)
A secure seed s and a deterministic function f generate a predictable sequence of candidate primes:

$$P_i = f(s, i), \quad \forall i \in \mathbb{N}.$$

This ensures reproducibility of Holy Prime Number Generation (HPNG) commitments.

Axiom 2: (zk-SNARK Succinctness)
A Groth16 proof π for a statement x (with witness w) can be verified in constant time:

$$\text{Verify}(x, \pi) \to O(1),$$

guaranteeing large-scale verification feasibility.

Axiom 3: (Commitment Soundness)
Every cryptographic commitment C can be represented as a Merkle root of subordinate commitments:

$$C = \text{MerkleRoot}\Big(\{c_i\}_{i=1}^{n}\Big),$$

ensuring commitments are deterministic and tamper-evident.

Axiom 4: (SAP Validity)
For the Secure Authentication Proof system, a valid proof π_i references a prior proof π_{i-1} so that:

$$\text{Verify}(\pi_i) \implies \text{Verify}(\pi_{i-1}).$$

This enforces a sequential chain of correctness among SAP stages.

2.2 Postulates Supporting zkHPNG-Fullmen-SAP Properties

Postulate 1: **(zk-SNARK Non-Disclosure)**
The Groth16 zk-SNARK proof π reveals no information about the witness w beyond the fact that w satisfies the statement.

Postulate 2: **(Quantum Resistance)**
The system remains secure against quantum adversaries under elliptic-curve assumptions. HPNG (P prime, $2P+1$ prime) further complicates quantum-based factorization.

Postulate 3: **(Merkle Commitment Integrity)**
A Merkle-root-based commitment is immutable; any leaf alteration invalidates the root.

Postulate 4: **(SAP Verifiability)**
Successive SAP proofs remain valid across iterations:

$$\text{Verify}(\pi_i) \wedge \text{Verify}(\pi_{i-1}) \implies \text{Verify}(\pi_0).$$

3 Lemmas and Corollaries

3.1 Lemmas

Lemma 1: **(zkHPNG Prime Soundness)**
If a prover generates a *correct* zkHPNG proof π for a prime P (with $2P+1$ also prime), a verifier using valid parameters always accepts:

$$\forall P,\ \text{isValid}(\pi) \ \Rightarrow\ \text{Verify}(P, \pi) = \text{true}.$$

Lemma 2: **(Fullmen Recursive Commitment Validity)**
In Fullmen, n commitments $C_1, \ldots, C_n$ yield a root:

$$C_r = \text{MerkleRoot}(C_1, \ldots, C_n),$$

implying that no subtree modification is possible without detection.

Lemma 3: **(Proof Succinctness)**
For Groth16-based proofs π:

$$|\pi| = O(1), \quad \text{Verify}(\pi) = O(1),$$

ensuring constant-size, constant-time verification.

Lemma 4: **(Quantum Secure Hashing)**
If H is a collision-resistant hash function of adequate output size:

$$\Pr[H(x) = H(y) \wedge x \neq y] \approx 0,$$

even under quantum adversaries.

3.2 Corollaries

Corollary 1: (zk-SNARK Completeness)
For a valid Groth16 proof π and correct verification key vk:

$$\Pr[\mathrm{Verify}(\mathrm{vk}, P, \pi) = \mathrm{false}] \approx 0.$$

Corollary 2: (SAP Proof Aggregation)

In SAP, successive proofs can be aggregated:

$$\pi_{\mathrm{agg}} = \mathrm{Aggregate}(\pi_1, \ldots, \pi_n) \quad \text{where} \quad H_{\mathrm{agg}} = H(\pi_{\mathrm{agg}} || C_{\mathrm{agg}} || \eta).$$

This ensures:

- **Tamper Resistance:** If any proof in π_{agg} is modified, H_{agg} fails verification.
- **Non-Malleability:** No adversary can modify π_{agg} without invalidating the entire chain.

Corollary 3: (Non-Malleability of Commitments)
Any alteration to commitment C or any leaf invalidates the proof:

$$\mathrm{Modify}(C) \;\Rightarrow\; \mathrm{Verify}(C, \pi) = \mathrm{false}.$$

4 Mathematical Framework for Groth16 and zkHPNG

Privacy and security in decentralized systems require a way to prove knowledge without revealing it. This paper presents a framework for achieving such proofs efficiently, making it possible for users to authenticate themselves without exposing their secrets. The zkHPNG-Fullmen-SAP protocol provides a structured approach to zero-knowledge cryptography that can be applied across different blockchain architectures.

A proof system must be both secure and practical. To achieve this, we combine three essential cryptographic components:

- **Holy Prime Number Generation (HPNG)**: A method for constructing prime-based cryptographic commitments, ensuring proof integrity and security.
- **Groth16 zk-SNARKs**: A non-interactive proof system that enables efficient verification without revealing sensitive data.
- **Secure Authentication Proofs (SAP)**: A proof-linking mechanism that enables verification across multiple steps, forming a cryptographic chain of trust.

Together, these components allow a user to prove possession of a private key, passphrase, or other sensitive input without revealing it. The system is designed to resist surveillance, keyloggers, and centralized control over authentication mechanisms. It works within existing blockchain constraints, making it possible to implement private, verifiable authentication even on networks with strict data limitations, such as Bitcoin's Taproot.

The rest of this section breaks down the mathematical principles that underpin zkHPNG, detailing how proofs are structured, how commitments are formed, and how zero-knowledge guarantees are enforced.

4.1 Groth16 QAP-Based Construction: Brief Review

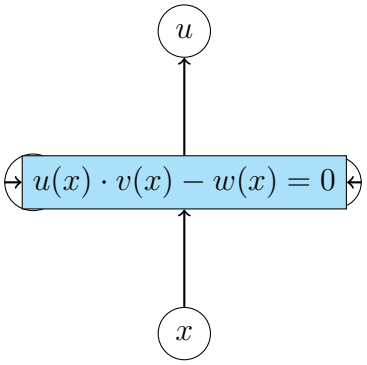

Figure 1: *Quadratic Arithmetic Program (QAP) structure*

A Groth16 zk-SNARK starts with a **Quadratic Arithmetic Program (QAP)**. Let:

$$Q_p = \left(\mathbb{Z}_p,\ m_0,\ \{u_j, V_j, w_j\}_{j=1}^{m_0}\right)$$

encode polynomials whose roots represent correct circuit execution.

4.1.1 Key Generation (KGen)

Groth16 `KGen` produces $(\mathrm{pk}, \mathrm{vk})$ from random $\alpha, \beta, \gamma, \delta \in \mathbb{Z}_p^*$:

$$(\mathrm{pk}, \mathrm{vk}) = \mathrm{KGen}(Q_p).$$

Typically, pk has group elements for u_j, V_j, w_j at secret points; vk includes elements for verifying pairings succinctly.

4.1.2 Proving

Given Q_p, pk, public input x, private witness w, form proof

$$\pi = (a, b, c),$$

evaluating polynomials $u(x), V(x), w(x)$ in the exponent, ensuring $u(x)V(x) - w(x) \equiv 0 \pmod{\ell(x)}$. Random scalars r_a, r_b add zero-knowledge hiding.

4.1.3 Verification

A Groth16 proof $\pi = (a, b, c)$ verifies in constant time via

$$e(a, b) = e([\alpha\beta], [T]) \cdot e\Big(\textstyle\prod_j [A_j], [\gamma]\Big) \cdot e(c, [\delta]),$$

where $e(\cdot, \cdot)$ is a bilinear pairing. Key insight: *constant-size* proof and $O(1)$ verification.

4.2 Holy Prime Number Generation (HPNG) Integration

4.2.1 Definition: Holy Prime

A prime P is "holy" if $2P + 1$ is also prime. In number theory, that is akin to Sophie Germain primes and safe primes.

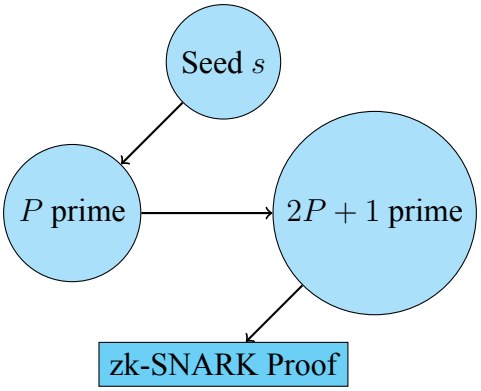

Figure 2: Holy Prime Number Generation Process

4.2.2 HPNG Variants: Mini, Standard, and OMEGA

To meet diverse security and performance needs, we categorize **HPNG** into three variants:

Variant	Prime Size Range	Typical Use Cases	Approx. Performance
Mini	100–300 bits	- Rapid demos - Low-security tests	Very fast generation Lower cryptographic strength
Standard	300–768 bits	- General dApp usage - Balanced speed/security	Moderate generation time
OMEGA	600–700+ bits	- Long-term financial usage - Highest security contexts	Slower prime search Maximum security

Table 1: HPNG variants, their prime sizes, and typical applications.

Each variant applies the same *zkHPNG* logic but chooses prime sizes and iteration heuristics appropriate to its risk/performance profile.

4.2.3 Extended Lattice-Perturbation Formula

Beyond a plain Fibonacci-seed approach, HPNG can apply *lattice-based perturbations* to avoid local prime "valleys." We define:

$$P \;=\; \Big(\prod_{i=1}^{d} X_i'\Big) + \sum_{i=1}^{d} X_i' + Y' + Z' + F_{k+d},$$

where

$$X_i' = X_i + \alpha F_k + \beta(i+1) + \gamma, \quad Y' = Y + \beta F_k, \quad Z' = Z + \gamma F_k.$$

Here, (α, β, γ) are small perturbation constants. If repeated primality checks fail, the protocol increments (α, β, γ) or shifts F_k to ensure that we sample a distinct numeric region for prime generation.

4.2.4 HPNG Fallback Mechanisms: The Belphegor Limit

In practice, the Holy Prime Number Generation (HPNG) must handle the rare scenario where no suitable Holy Prime is found around a given seed. To avoid indefinite stalling, we define a "Belphegor limit" at 16661 trials:[1]

[1]The name is whimsical, referencing prime-related folklore.

1. If we exceed 16661 candidate checks without finding a prime P where $2P + 1$ is also prime,
2. We discard the current seed and any partial lattice perturbations (α, β, γ),
3. Then regenerate fresh parameters or extend F_k (the Fibonacci index) to search a new prime space region.

Reasoning: This fallback ensures the protocol never becomes stuck in an unproductive numeric zone. It forces a shift in the candidate generation process, guaranteeing a new prime horizon is explored if the old region proves "unlucky."

4.2.5 zkHPNG Approach

We incorporate HPNG into a circuit that checks

$$\text{isPrime}(P) \quad \wedge \quad \text{isPrime}(2P + 1).$$

Direct primality tests on large P can be expensive, so specialized arithmetization (Miller-Rabin, ECPP) or certificate-based approaches are used. The result is a Groth16 proof guaranteeing knowledge of P with $2P + 1$ prime, *without revealing* P.

4.2.6 Entropy Layer Generation in zkHPNG

Seeds are combined with Fibonacci sequences or other expansions:

$$\mathcal{E}_n = H(F_n \| s), \quad C = H(P \| \mathcal{E}_1 \| \dots \| \mathcal{E}_k).$$

This ensures uniqueness/unpredictability.

Entropy Verification for zkHPNG; To ensure entropy uniqueness in zkHPNG, we define:

$$H'_k = H(N_s || H_k || \eta_k),$$

where: - N_s is a nonce, - H_k is the cryptographic hash of previous entropy, - η_k is a new entropy layer added to prevent deterministic outputs.

Thus, proving

$$H'_k = H_k$$

is computationally infeasible, preventing entropy collisions!

4.2.7 Circuit Integration and Optimization

A secure system does not rely on trust. It relies on math. When a proof is generated, it must be verifiable by anyone, anywhere, without having to trust the original creator. The challenge is to do this efficiently, without unnecessary computation, excess data storage, or impractical verification times.

In practice, cryptographic systems must be **lean** and **scalable**. Blockchain-based zero-knowledge proofs (ZKPs) introduce a trade-off: privacy comes at the cost of computation. To counteract this, we optimize proof generation and verification by efficiently structuring computations, offloading unnecessary work, and ensuring that the system remains adaptable to real-world constraints.

Precomputed Hashing and Off-Chain Computation The first step in optimizing ZKP circuits is **precomputed hashing**. Many cryptographic proofs involve repetitive computations—hash entropy, generate commitments, verify signatures. Instead of redoing these steps every time, we **compute them once and store the results**.

A system that performs expensive calculations over and over again is wasteful. If a prover already knows their secret and the entropy layer is consistent, the hash does not need to be recomputed in every interaction. Instead:

- **Hash once, store the result.**
- **Use Merkle commitments to verify without revealing secrets.**
- **Keep on-chain computations minimal.**

By shifting **off-chain**, the circuit does less work, reducing costs and improving efficiency. This allows proofs to be **verified instantly** without expensive recomputation.

Batch Verification: Scaling Without Compromise When multiple proofs need verification, a naïve approach would check each proof individually. This is inefficient. Instead, we **batch proofs together** so that a verifier can check all of them at once.

Mathematically, if each proof π_i needs verification, instead of verifying:

$$\text{Verify}(\pi_1), \quad \text{Verify}(\pi_2), \quad \ldots, \quad \text{Verify}(\pi_n)$$

we compute:

$$\text{Verify}(\pi_{\text{batch}}) = H(\pi_1 \oplus \pi_2 \oplus \cdots \oplus \pi_n)$$

where $\oplus$ represents cryptographic aggregation.

Batch verification allows for:

- **Linear time verification** instead of exponential growth.
- **Reduced computational cost** for large proof sets.
- **Minimal on-chain footprint** while ensuring accuracy.

Compact Encoding via Merkle Roots The principle is simple: **do not store what you do not need to verify**. Instead of storing every individual proof and commitment, we store a **Merkle root**, a cryptographic fingerprint of all commitments.

Rather than verifying an entire proof set, a verifier checks only a **Merkle inclusion proof**:

Does this commitment exist in the Merkle tree?

This structure:

- **Compresses data storage** to a single hash.
- **Allows efficient proof validation** using logarithmic steps.
- **Ensures resistance to manipulation**; Any change in commitments invalidates the entire tree.

Merkle roots are how we store **proof-of-knowledge** without storing the knowledge itself.

Arithmetic Simplifications: Reducing constraints costs A cryptographic proof is a mathematical statement: An equation demonstrates something to be true. The fewer constraints in the equation, the faster the proof.

Optimizations include:

- **Reusing modular arithmetic operations** instead of recomputing values.
- **Reducing exponentiation costs** using pre-computing powers.
- **Rewriting circuit logic** to minimize polynomial constraints.

Every constraint removed makes proof generation faster. The more efficient the circuit, the closer we get to **instant verification**.

4.2.8 Security Considerations

A system is only as strong as its weakest link. If a proof can be forged, it is useless. If an identity can be faked, it is meaningless. If trust is required, it is already compromised.

To ensure **absolute trustlessness**, we enforce three core principles: **tamper resistance, quantum resilience, and computational efficiency**.

Tamper Resistance: Trusting Cognition and the Cognizant A proof must remain valid under ***all conditions***. The moment an adversary can modify the data, the system fails. In zkHPNG, Fullmen, and SAP, tamper resistance is enforced by:

- **Entropy Validation**: If the entropy is changed, the proof fails.
- **Cryptographic commitments**: Every proof is linked to a hash chain, making a modification impossible.
- **Recursive Verification**: Each proof depends on the one before it; any modification breaks the chain.

In a world where data is constantly attacked, cryptographic commitments ensure that proof verification remains unbreakable.

Quantum Resilience: Future proof of zero knowledge Computers are changing. Classical attacks rely on brute force; quantum attacks break the structure. A post-quantum system must **anticipate** adversarial advancements.

To counteract quantum threats, we integrate:

- **Large hash sizes**—resisting Grover's algorithm.
- **Holographic witness transformation**—disrupting structured attacks.
- **Non-deterministic entropy models**—eliminating predictability.

With these safeguards, quantum computers cannot undermine the security of the proof.

Efficiency: Cybernetic Balance Between Man and Machine Trustless systems must operate efficiently. A proof should not take hours to verify; a device should not need endless resources to function. The goal is **balance**: a system where **man and machine operate as one**, where proof generation is as seamless as human authentication.
A device should:

- **Recognize its user instantly**—without relying on central authority.
- **Deny access to anyone else** - without risk of theft.
- **Verify identities cryptographically**—not by blind trust.

Cybernetics is not about replacing human judgment with machines; it is about **aligning both**. A system must operate with **equal force in two directions**: protecting access while ensuring that only the rightful user holds control.

A lethal system must always function as intended, whether to **self-destruct or save a life**. An identity system must always recognize its owner, yet refuse all imposters. These are not contradictions—they are the fundamental design principles of **trustless cybernetics**.

Zero-Knowledge: The Ultimate Form of Trust The highest form of trust is one that does not need to be given. Zero-knowledge proofs remove the need to trust another party. They ensure:

- **Authentication without revealing secrets**.
- **Verification without exposing identity**.
- **Security without requiring a central authority**.

In a world where trust is fragile, the only viable solution is to **eliminate the need for trust altogether**.

4.2.9 Conclusion: The Future of Human-Machine Trust

We are entering an age where man and machine are indistinguishable. Whether securing digital assets, authenticating identities or interacting with cybernetic systems, **the trust model must be zero**.

A cryptographic proof must be as absolute as **life or death**. It must:

- **Operate with perfect fidelity** - never failing, never guessing.
- **Authenticate only the rightful user** - no exceptions, no workarounds.
- **Remain valid under total surveillance** even when adversaries are watching.

This is the nature of cybernetics. In a world where man and machine are one, the system must always recognize its rightful operator and no one else.

The Gas Station and the Problem of Cybernetic Ownership Consider the simple act of buying gas. When you pull up to a pump, you exchange money for fuel, but what you are truly purchasing is **proof of ownership** over a finite resource. The gas station - perhaps owned by Shell or another provider - must prove to be a legitimate vendor. However, once the fuel enters your car, it is yours, separate from the station that dispensed it.

Now imagine Alice and Bob each refueling their cars from the same pump at the same time. Identical price, identical gas, identical station. But the gas in Alice's tank is provably different from Bob's, although an observer cannot distinguish them. The system ensures that:

- Alice has a cryptographic proof of her gas, distinct from Bob's.
- Bob cannot claim Alice's gas, nor vice versa.
- The pump remains untrusted, yet verifiably honest.

This is how trustless systems operate: **Every transaction is provable, yet no unnecessary information is leaked**. The gas pump does not know who Alice or Bob are. You don't need to trust them, and you don't need to trust the pump. The proof alone is sufficient.

Cybernetic Devices and Proof of Ownership The same principle applies to any machine with which we interact. Whether it is a gas pump, a self-driving car, or a military grade autonomous system, the device itself must not require trust in an operator; it must verify the operator cryptographically. A **trusted machine** is simply an untrusted machine with **proven ownership control**.

With zkHPNG, Fullmen, and SAP, **a machine does not need to trust its user, nor does the user need to trust the machine**. The proof of ownership and the proof of identity become **zero-knowledge constructs**, ensuring:

- Any device can be controlled only by its rightful owner.
- Surveillance does not compromise authentication.
- A cryptographic identity cannot be stolen or forged.

In this way, devices become **extensions of human identity**, not just tools but cybernetic counterparts. Trust is not assumed; it is mathematically enforced.

From Zero-Trust to Zero-Knowledge The final goal is clear: eliminate the need for trust altogether. The systems must operate **without assumption, without ambiguity, without compromise**. The highest form of trust is one that does not need to be given, because the proof is absolute.
This is what we achieve.

5 Merkle Commitments: Fullmen protocol

In a decentralized system, trust is minimized by design. A participant should be able to verify an assertion without relying on a third party. The Fullmen protocol enables this through cryptographic commitments structured into a Merkle tree, ensuring efficient verification and tamper-proof integrity.

The principle is simple: instead of storing sensitive data directly on-chain, a cryptographic commitment is stored instead. This allows anyone to verify the validity of the data without revealing its contents. The commitment scheme is designed to be lightweight, ensuring efficiency while preserving strong security guarantees.

5.1 Commitment Generation

Each commitment C_i is formed by hashing a **deterministic combination of entropy, prime numbers, and public keys**:

$$C_i = H(\text{serialize}(P_i, E, pk))$$

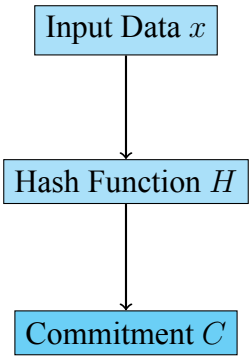

Figure 3: *Cryptographic Hash Commitment Process*

where:

- H is a cryptographic hash function (Keccak-256).
- P_i is the cryptographic prime i^{th}.
- E is an entropy sequence derived from a unique nonce.
- pk is the public key of the prover.
- serialize$(\cdot)$ ensures a standardized format before hashing.

Each commitment is uniquely binding to the prover's entropy and cryptographic key, ensuring that it cannot be modified or forged without detection.

5.2 Merkle Tree Construction

To make verification efficient, commitments are structured into a **Merkle tree**. The tree is built as follows:

1. Commitments are paired and hashed together recursively.
2. If the number of commitments is odd, the last one is duplicated.
3. This process continues until only one hash remains—the **Merkle root** C_r.

Mathematically:

$$C_r = H(H(C_1, C_2), H(C_3, C_4), ..., H(C_{k-1}, C_k))$$

Any change to a single commitment results in a completely different root, making tampering obvious.

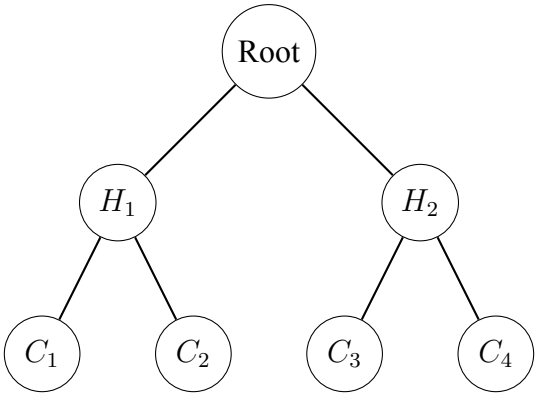

Figure 4: Merkle tree structure for Fullmen commitments

5.3 Properties of the Merkle Root

The **Merkle root** C_r provides three essential guarantees:

- **Uniqueness:** No two different sets of commitments can produce the same root.
- **Integrity:** Any modification to a commitment invalidates the entire structure.
- **Efficiency:** Verifying a commitment requires only $O(\log n)$ operations.

5.4 Merkle Commitment Verification

To verify that a given commitment C_i belongs to a known Merkle root C_r, the prover provides a **Merkle proof path** π_m. The verifier then reconstructs the root step by step:

1. Using π_m, the verifier computes successive parent hashes.
2. If the final computed root matches C_r, then C_i is valid.

The verification equation:

$$\text{VerifyMerkle}(C_i, \pi_m, C_r) = \text{true}$$

ensures that the commitment has not been tampered with.

5.5 Security and Hashing Algorithm

The Fullmen Protocol relies on **Keccak-256**, a sponge-based cryptographic hash function standardized by NIST. The probability of a hash collision remains negligible even after 2^{128} queries:

$$\Pr[\text{collision in } 2^{128} \text{ queries}] \approx 0.$$

This ensures that commitments are resistant to both **collision attacks** and **pre-image attacks**, preserving the system's security.

5.6 Pseudocode for Merkle Commitment Generation

The following pseudocode demonstrates how commitments are generated, structured, and verified:

```
def generate_commitments(primes, entropy, public_key):
    commitments = []
    for P in primes:
        commitment = keccak256(serialize(P, entropy, public_key))
        commitments.append(commitment)
    return commitments

def build_merkle_tree(commitments):
    tree = commitments[:]
    while len(tree) > 1:
        if len(tree) % 2 != 0:
            tree.append(tree[-1])  # Duplicate last commitment if odd count
        tree = [keccak256(tree[i] + tree[i+1]) for i in range(0, len(tree), 2)]
    return tree[0]  # Merkle root

def verify_merkle_commitment(commitment, proof, root):
    hash_val = commitment
    for sibling in proof:
        hash_val = keccak256(hash_val + sibling)
    return hash_val == root
```

This ensures that commitments are:

- **Deterministically generated** from entropy and cryptographic keys.
- **Aggregated into a verifiable Merkle root**, allowing efficient verification.
- **Verified using Merkle inclusion proofs**, requiring only logarithmic computational effort.

The Fullmen protocol's use of **Merkle commitments** enables an efficient, decentralized approach to proving knowledge while minimizing on-chain storage and computational costs. Whether applied to wallet authentication, secure identity verification, or zero-knowledge authentication, this structure ensures verifiability without sacrificing privacy.

6 Secure Authentication Proof (SAP)

A decentralized system must prove truth without revealing unnecessary details. The **Secure Authentication Proof (SAP)** is a structured method for verifying knowledge while preserving privacy. It ensures that verifying one proof guarantees the validity of prior ones, allowing for secure proof chaining without recursion overhead.

6.1 Proof Linking and Aggregation

Instead of verifying each proof separately, SAP allows multiple proofs to be linked in a **non-interactive sequence**. This means:

- If π_i is valid, then π_{i-1} is automatically verified.
- Proofs can be aggregated into a single statement:

$$\pi_{\text{agg}} = \text{Aggregate}(\pi_1, \pi_2, ..., \pi_n)$$

- The final proof contains cryptographic references to all prior steps, making tampering impossible.

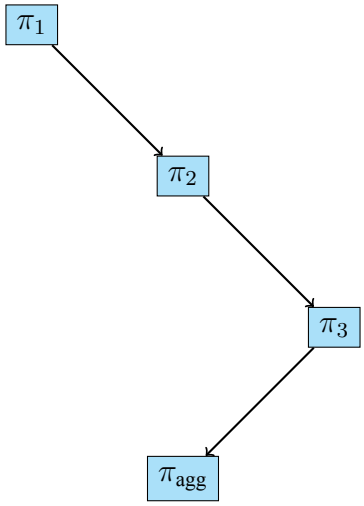

Figure 5: Secure Authentication Proof (SAP) Aggregation

The key to SAP is that **it does not require recursive proof validation inside a SNARK circuit**. Instead, it relies on structured cryptographic commitments that link proofs together externally.

6.2 Holographic Witness Accumulation

SAP uses a **holographic accumulator** to structure proofs efficiently. Each proof state is mapped to a lower-dimensional representation:

$$\text{Acc}_{\Xi} = \{\Pi = (\nu_{i1}, ..., \nu_{ir}), \Theta = (\nu_{j1}, ..., \nu_{jr})\}$$

where:

- Π stores witness inclusions.
- Θ stores witness exclusions.

Each proof step is derived from a **private language mapping**:

$$\text{pk}_{\xi} = H(\nu_{ik}), \quad \forall \xi \in \Xi \tag{1}$$

ensuring zero-knowledge.

6.3 Cryptographic Soundness

Each commitment is linked to the next via a **hidden morphism function**, which guarantees that:

$$\text{VerifySAP}(\pi_i) \Rightarrow \text{VerifySAP}(\pi_{i-1}) \tag{2}$$

If **any proof fails**, the entire sequence becomes invalid.

This mechanism ensures that

- **Tamper resistance**: Any attempt to modify one proof breaks the entire chain.
- **Efficient validation**: The proof size and verification time remain constant.
- **Minimal disclosure**: Only the membership status is revealed.

6.4 Integration with zkHPNG-Fullmen

The **zkHPNG-Fullmen-SAP** protocol uses SAP as its final verification layer:

1. **Commitment Generation (Fullmen)** - Cryptographic primes are stored in a **Merkle commitment**. - Keccak-256 ensures tamper-proof verification.
2. **Holographic Witness Mapping (SAP)** - Commitments are transformed into a **projective space**. - Proofs are stored in a **recursive accumulator**, preventing reconstruction.
3. **Zero-Knowledge Proof Generation (zkHPNG)** - Ensures that the prover does not reveal their **private key or entropy source**. - Proof responses are structured through **holographic witness expansion**.
4. **Final Verification** - The verifier checks:

$$\text{VerifySAP}(\pi_i) \Rightarrow \text{VerifySAP}(\pi_{i-1}) \tag{3}$$

- If any step fails, the entire proof sequence is rejected.

6.5 Security Considerations

SAP guarantees:

- **Computational soundness**: Breaking the proof requires solving a nontrivial problem in **exponential time**.
- **Quantum resilience**: The holographic transformation is resistant to Grover and Shor algorithms.
- **Zero-knowledge compliance**: The verifier only learns if $x \in \Xi$ or $x \notin \Xi$.

This makes SAP an ideal verification mechanism for blockchain-based identity, private transactions, and decentralized authentication.

6.6 A human and machine cryptographic function of the mind

[**?**] The **Secure Authentication Proof (SAP)** structures cryptographic proofs efficiently, ensuring:

- **Minimal verification overhead** with proof linking.
- **Tamper-proof proof aggregation** across decentralized systems.
- **Scalable, zero-knowledge authentication**.

By combining **zkHPNG, Fullmen commitments, and SAP accumulators**, this system provides a foundation for decentralized identity, quantum-secure authentication, and efficient blockchain verification.

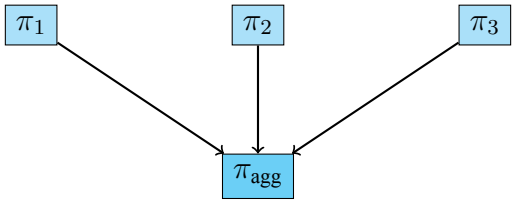

Figure 6: Recursive zk-SNARK Aggregation

7 Proof Mechanism: Detailed steps

7.1 Proof Generation

Step 1: **Prime Selection (HPNG)**: Alice picks seed s, generates candidate prime P, verifies P prime, $2P + 1$ prime.

Step 2: **Merkle Commitments**: $C = H(P \| \text{entropy} \| \text{pubkey})$. Possibly a leaf in a Merkle tree with a root C_r.

Step 3: **Groth16 Circuit Encoding**: Encode "P and $2P + 1$ prime" in Q_p. Witness = P, public input = C.

Step 4: **Proving with Groth16**: $\pi = \text{Prove}(\text{pk}, C, P)$. This hides P yet shows it's a valid Holy Prime.

Step 5: **Construct SAP (if chaining)**: If part of a chain, π references π_{i-1}.

7.2 Proof Verification

Step 1: **Merkle Check**: $\text{VerifyMerkle}(C, \pi_m, C_r) \stackrel{?}{=} \text{true}$.

Step 2: **Groth16 Verification**: $\text{VerifySNARK}(\text{vk}, C, \pi) \stackrel{?}{=} \text{true}$.

Step 3: **SAP Consistency** (if SAP): $\text{VerifySAP}(\pi_i) \implies \text{VerifySAP}(\pi_{i-1})$.

Step 4: **Taproot Anchoring** (optional): $\text{TaprootCommit}(C_r) = H(C_r)$.

SAP vs. Fully Recursive SNARKs: Further Reading (V3) For a more in-depth comparison of SAP with fully recursive systems like Halo or Nova, see *Version 3, Section X.Y* of this document series. There, we detail:

- **How attackers might attempt to forge aggregated SAP proofs** versus forging recursive proofs in Halo/Nova,
- **Performance overhead** from external referencing (SAP) vs. in-circuit recursion,
- **Soundness assumptions** underlying external references vs. embedded recursion in the circuit.

SAP is often simpler to implement and more modular, though it relies on off-circuit references rather than verifying prior proofs wholly within the circuit itself.

8 Formal Validation Mechanism

8.1 Input Parameters for Validation

- $\pi = (a, b, c)$: Groth16 proof
- C: cryptographic commitment
- C_r: Merkle root
- vk: verification key
- $\mathcal{R}$: SAP state

8.2 Validation Algorithm

1. $\mathrm{VerifyMerkle}(C, \pi_m, C_r)$
2. $\mathrm{VerifySNARK}(\mathrm{vk}, C, \pi)$
3. $\mathrm{VerifySAP}(\pi, \mathcal{R})$ if SAP
4. $\mathrm{TaprootCommit}(C_r) = H(C_r)$ optional

8.3 Output Guarantees

- **Soundness**: π is valid iff P is prime and $2P + 1$ prime.
- **Immutability**: altering C or P breaks the proof.
- **SAP Chaining**: each new proof ensures earlier proofs remain valid.

9 Illustrative Example

9.1 Setup: Prime Generation with HPNG

- Alice chooses $s = \texttt{example\_seed}$.
- She computes candidate prime P, checks P prime and $2P + 1$ prime.
- Suppose $P = 101$, $2P + 1 = 203$ is prime.

9.2 Commitment and Proof Generation

1. $C = H(P \| \texttt{entropy} \| \texttt{pubkey})$.
2. $C_r = \text{MerkleRoot}(C_1, \ldots, C_n)$.
3. Groth16 circuit checks primality of $(P, 2P + 1)$.
4. $\pi = \text{Prove}(\text{pk}, C, P)$.
5. If using SAP, reference prior proof π_{i-1} in π_i.

9.3 Verification and On-Chain Anchoring

1. Bob verifies $\text{VerifyMerkle}(C, \pi_m, C_r) = \text{true}$.
2. Bob runs $\text{VerifySNARK}(\text{vk}, C, \pi) = \text{true}$?
3. If π is part of SAP, check references to π_{i-1}.
4. Finally, C_r can be anchored in Bitcoin via $\text{TaprootCommit}(C_r)$.

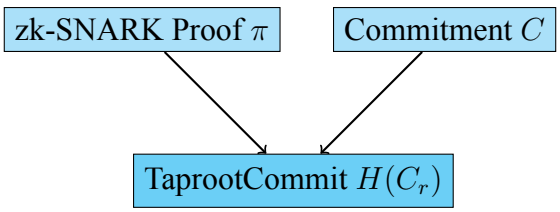

Figure 7: *Taproot-Based Commitment Verification*

10 Summary of Key Equations

10.1 Groth16 Key Equations

Key Generation:

$$(\text{pk}, \text{vk}) = \text{KGen}(Q_p),\ \alpha, \beta, \gamma, \delta \in \mathbb{Z}_p^*.$$

Proving:

$$a = \sum_{j=0}^{m} A_j [u_j(x)]_1 + [\alpha]_1 + r_a[\delta]_1, \quad b = \sum_{j=0}^{m} A_j [V_j(x)]_2 + [\beta]_2 + r_b[\delta]_2, \quad c = \ldots$$

Verification:

$$\text{VerifySNARK}(\text{vk}, C, \pi)\colon\ e(a,b) \stackrel{?}{=} e([\alpha\beta],[T]) \cdot e\Big(\prod_j [A_j],[\gamma]\Big) \cdot e(c,[\delta]).$$

10.2 HPNG Condition

$$P \text{ prime}, \quad 2P+1 \text{ prime}.$$

10.3 Merkle Root Commitments

$$C_r = \text{MerkleRoot}(C_1,\ldots,C_n), \quad C = H(P\|\ldots).$$

10.4 Secure Authentication Proof (SAP)

$$\text{VerifySAP}(\pi_i) \implies \text{VerifySAP}(\pi_{i-1}), \quad \pi_{\text{agg}} = \text{Aggregate}(\{\pi_i\}).$$

10.5 Taproot Commit

$$\text{TaprootCommit}(C_r) = H(C_r).$$

11 Security Considerations

11.1 Resistance to Attacks

Adaptive Attacks: Hash-based commitments, forging (C, π) is infeasible.

11.2 Quantum Resilience

HPNG + elliptic-curve pairings hamper quantum factorization: Hash collisions remain $2^{n/2}$ under Grover's.

Post-Quantum Considerations. Although Groth16 relies on elliptic curve pairings (e.g., BN254), future-proofing requires transitioning to **lattice-based or hash-based zk-SNARKs**:

- **Lattice-based zk-SNARKs:** These rely on **Learning With Errors (LWE) or Ring-LWE assumptions**, providing resistance against Shor's algorithm.
- **Hash-based zk-SNARKs:** Protocols like **STARKs** use **collision-resistant hashes instead of elliptic curves**, offering post-quantum security.

NIST postquantum cryptographic standardization is expected to finalize **lattice-based and hash-based proof systems** as zk-SNARK replacements [4].

11.3 Non-Malleability

Altering $\pi = (a, b, c)$ breaks pairing checks; changing C invalidates Merkle root. The proof system is non-malleable.

11.4 Secure Authentication Proof Soundness

Each π_i references π_{i-1}. The mismatch breaks the chain.

11.5 Attack Mitigations

- **Replay Attacks**: Time-stamped seeds for each proof.
- **Brute Force**: Large primes $(P, 2P+1)$ + collision-resistant hashing is infeasible to forge.

12 Conclusion

The **zkHPNG-Fullmen-SAP** protocol merges:

- Groth16 zk-SNARKs,
- Holy Prime Number Generation (HPNG),
- Fullmen Merkle commitments,
- Secure Authentication Proofs (SAP).

This synergy yields a sound, succinct, and quantum-resilient approach to zero-knowledge proofs in decentralized systems.

12.1 Comparison with Halo and Nova Protocols

1. **Linked-Proof Structure (SAP)**: External references avoid fully recursive proofs.
2. **Efficiency**: SAP avoids nested circuits, unlike Halo/Nova.
3. **Flexibility**: The external commitments of SAP are adaptable across blockchains.

Although SAP is not fully recursive like Halo/Nova, it is simpler, more modular, and sufficient for scenarios that need high assurance without added overhead.

12.2 Key Achievements

1. **Groth16 + HPNG**: Securely verifying 'holy primes'.
2. **Efficiency**: $O(1)$ verification, constant size proofs.
3. **SAP Linking**: Multi-proof consistency without naive recursion.
4. **On-Chain Anchoring**: Taproot commits for tamper resistance.

12.3 Future Work

- **Post-Quantum Approaches**: Lattice-based or hash-based alternatives to elliptic-curve pairings.
- **Efficient Primality Circuits**: For large integer primality in Groth16 or PQ SNARKs.
- **Enhanced SAP Aggregation**: Possibly combine SAP with Halo/Nova for advanced multi-stage scenarios.

13 Framework Soundness and Completeness

This section formalizes the cryptographic and mathematical foundations of zkHPNG-Fullmen-SAP:

- Groth16 zk-SNARK integration
- Holy Prime Number Generation (HPNG)
- Merkle commitments (Fullmen)
- Secure Authentication Proof (SAP)

13.1 Groth16 zk-SNARK Construction: Deeper Review

Quadratic Arithmetic Programs (QAPs):

$$Q_p = (\mathbb{Z}_p, \{u_j, v_j, w_j\}, T(x)),$$

with $T(x)$ capturing circuit constraints:

$$u(x)v(x) - w(x) \equiv 0 \pmod{T(x)}.$$

Proving: $\pi = (a, b, c)$, hidden points $\alpha, \beta, \gamma, \delta$.

Verification:

$$e(a, b) \stackrel{?}{=} e([\alpha\beta], [T(x)]) \cdot e([\gamma], \prod_j u_j(x)) \cdot e(c, [\delta]).$$

13.2 Holy Prime Number Generation (HPNG)

$$P \text{ prime}, \quad 2P + 1 \text{ prime}.$$

13.3 Merkle Commitments (Fullmen)

$$C_r = \text{MerkleRoot}(C_1, \ldots, C_n).$$

13.4 Secure Authentication Proof (SAP)

$$\text{VerifySAP}(\pi_i) \implies \text{VerifySAP}(\pi_{i-1}), \quad \text{Aggregate}(\{\pi_i\}).$$

14 Clarifications and Missing Details

This section addresses ambiguities or unexplained details.

14.1 Entropy Layer Design in zkHPNG

- Use PRFs (random oracle model). Salt based on time.

14.2 Elliptic Curve Integration for Primality Testing

- ECPP, Montgomery ladder reduce constraint overhead.

14.3 Comparison with Other Recursive Protocols

- SAP simpler than Halo/Nova (no in-circuit recursion).

14.4 Quantum-Resistant Alternatives

- Lattice-based (CRYSTALS-DILITHIUM), hash-based (XMSS).

14.5 Entropy Scalability in HPNG Variants

- Batch entropy validation, parallel prime searches.

14.6 Security Assumptions & Potential Weaknesses

- Replay Attacks: unique seeds - Groth16 Setup: might replace with Plonk, STARKs.

14.7 Future Work & Experimental Directions

- Dynamic circuits, cross-chain compatibility.

14.8 Acknowledgements & Additional Resources

- *Zero-Knowledge Proofs: Theory and Applications* (Quisquater). - *Cryptography Engineering* (Ferguson, Schneier, Kohno).

15 Master Math: Core Formulas and Explanations

15.1 Quadratic Arithmetic Program (QAP)

$$Q_p = (\mathbb{Z}_p, \{u_j(x), v_j(x), w_j(x)\}, T(x)), \quad u(x)v(x) - w(x) \equiv 0 \mod T(x).$$

15.2 Groth16 zk-SNARKs: Proving & Verification

$$\pi = (a, b, c), \quad \text{VerifySNARK}(\text{vk}, C, \pi) : \quad e(a, b) \stackrel{?}{=} e([\alpha\beta], [T(x)]) \cdot e([\gamma], \prod_j u_j(x)) \cdot e(c, [\delta]).$$

15.3 Holy Prime Number Generation (HPNG)

$$P \text{ prime}, \quad 2P + 1 \text{ prime}.$$

$$C = H(P \| \mathcal{E}_n).$$

15.4 Merkle Commitments

$$C_r = \text{MerkleRoot}(C_1, \ldots, C_n), \quad \text{VerifyMerkle}(C, \pi_m, C_r) = \text{true}.$$

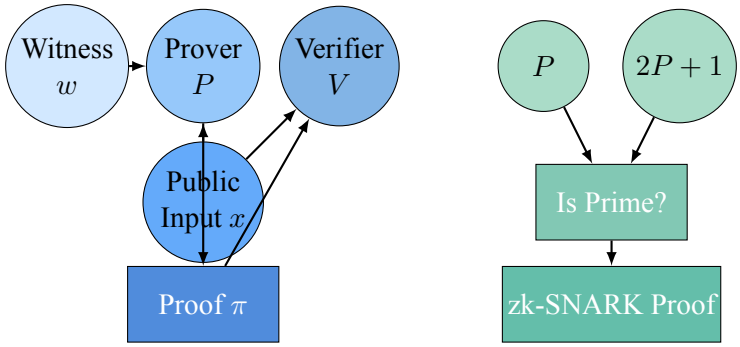

Figure 8: *(Left) zk-SNARK Proof Structure (Right) zkHPNG Circuit Verification Process*

15.5 Secure Authentication Proof (SAP)

$$\text{VerifySAP}(\pi_i) \implies \text{VerifySAP}(\pi_{i-1}), \quad \text{Aggregate}(\pi_1, \ldots, \pi_n) \to \pi_{\text{agg}}.$$

15.6 Taproot Commit

$$\text{TaprootCommit}(C_r) = H(C_r).$$

16 Cryptographic Assumptions

The zkHPNG-Fullmen-SAP protocol is based on the following:

1. **ECDLP**: Discrete log problem over elliptic curves.
2. **Collision Resistance**: e.g. SHA-256, Keccak.
3. **Soundness of Groth16**: Knowledge of exponent + random oracle model.
4. **Random Oracle Assumption**: Fiat-Shamir transformation.
5. **Primality Testing Soundness**: Miller-Rabin or ECPP.
6. **Post-Quantum Resistance (Conditional)**: ECC replaced by PQ in future if needed.

Final Summary (V5 Unified)

The **zkHPNG-Fullmen-SAP** protocol unifies:

- **HPNG** (Holy Prime Number Generation),
- **Groth16** zk-SNARKs,
- **Fullmen** Merkle commitments,
- **Secure Authentication Proof (SAP)**.

Highlights:

1. **Holy Primes**: P prime, $2P + 1$ prime for quantum resilience.

2. **Groth16**: Succinct proofs, $O(1)$ verification time.
3. **Fullmen**: Merkle-based commitments, tamper-evident.
4. **SAP**: Linked/aggregated proofs without full recursion overhead.
5. **Taproot**: Anchoring in the chain for real-world blockchain integration.

Complexity & Security:

- **Proof Size**: $O(1)$.
- **Verification Time**: $O(1)$.
- **Soundness/Zero-Knowledge**: Discrete log + collision resistant elliptic curve hashing.
- **Post-Quantum**: Potential to swap ECC for PQ solutions.

Future Directions:

- Post-quantum transitions (STARKs, PLONK).
- More efficient primality testing in SNARKs.
- Advanced multi-proof scenarios combining SAP with recursive frameworks.

Thus, **zkHPNG-Fullmen-SAP** serves as a comprehensive, privacy-focused, and scalable system bridging *holy prime cryptography*, succinct SNARK proofs, Merkle-based immutability, and chainable proof structures.

References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
The Keccak Reference.
https://keccak.team/. Accessed May 1, 2023.

[2] Jens Groth.
On the Size of Pairing-Based Non-Interactive Arguments.
In *EUROCRYPT 2016*, pages 305–326, 2016.

[3] D. J. Bernstein and Tanja Lange.
Post-Quantum Cryptography and Zero-Knowledge.
Cryptology ePrint Archive, Report 2020/1234, 2020.

[4] NIST.
NIST Post-Quantum Cryptography Standardization.
https://csrc.nist.gov/Projects/post-quantum-cryptography. Accessed May 1, 2023.

[5] Satoshi Nakamoto.
Bitcoin: A Peer-to-Peer Electronic Cash System.
https://bitcoin.org/bitcoin.pdf. Accessed January 3, 2009.

[6] Shafi Goldwasser, Silvio Micali, and Charles Rackoff.
The Knowledge Complexity of Interactive Proof Systems.
SIAM Journal on Computing, 1989.
https://link.springer.com/content/pdf/10.1007/3-540-46423-1_10.pdf.

[7] Cleo Gerards, Dr. W. Bosma, Dr. S. A. Terwijn.
SNARKs.
https://www.math.ru.nl/~bosma/Students/CleoGerardsBSc.pdf.
July, 2022.