Zyga: Optimized Zero-Knowledge Proofs with Dynamic Public Inputs

Tiago A. O. Alves
State University of Rio de Janeiro
Darklake
tiago@darklake.fi

Vitor Py Braga
Darklake
vitor@darklake.fi

Abstract

We present Zyga, a pairing-based zero-knowledge proof system optimized for privacy-preserving DeFi applications. Our main contribution is an enhancement of existing zkSNARK constructions that enables dynamic public input substitution during verification while maintaining privacy of witness components through one-sided encoding. The one-sided encoding aspect favors practical deployment constraints on Solana and Ethereum where G_2 scalar multiplications are computationally expensive. Zyga separates private values (blinded through trusted setup) from public values (instantiated on-chain), enabling applications like private trading against current market rates without reproofing. We further introduce *two-sided encoding*, an extension that removes circuit structure restrictions by adding a private B commitment and a rebase mechanism, enabling arbitrary R1CS circuits with proof reuse across changing base values. We provide rigorous security analysis under discrete logarithm and q-Strong Diffie-Hellman assumptions, demonstrating computational soundness, zero-knowledge, and completeness. Performance analysis shows verification requires only 3–4 pairings with constant proof size, making it practical for blockchain deployment where transaction costs are critical.

1 Introduction

Zero-knowledge proofs (ZKPs) enable verifiable computations without revealing underlying data, proving crucial for privacy-preserving applications across domains from secure voting [Adi08] and identity verification [CL01] to decentralized finance (DeFi) [But14]. A fundamental challenge in practical ZKP deployment is proof reusability: most existing systems fix all inputs at proof generation time, requiring complete reproofing when any parameter changes. This limitation becomes particularly problematic in dynamic environments where public data fluctuates frequently while private parameters remain constant.

In decentralized finance, this proof reusability problem manifests across numerous scenarios. Market-making strategies require constant repricing as oracle feeds update [ZGBJ21]. Lending protocols must adapt to changing interest rates while preserving borrower privacy [GPMPG20]. Cross-chain bridges need fresh proofs for each price update when validating private transfer amounts against current exchange rates [ZAZ+21]. Auction mechanisms become impractical when bid validation requires reproofing for each price discovery round [GY18]. These applications share a common pattern: private user inputs that should remain hidden while public market data changes continuously.

The transparency inherent in blockchain systems exacerbates these privacy challenges. Transaction details visible in the mempool enable maximal extractable value (MEV) attacks [DGK+19], where automated agents exploit advance knowledge of pending transactions. In trading scenarios, MEV bots observe pending transactions and frontrun them by inserting higher-fee transactions that profit from anticipated price impacts, leading to user losses through slippage [QZG+22]. Current mitigations include splitting trades across multiple transactions to obscure intent or routing through private relays like Flashbots on Ethereum [Fla21]. However, these approaches increase transaction fees, introduce latency, elevate failure risks, and often lack full decentralization or availability across different blockchain networks like Solana and Ethereum.

Existing privacy protocols such as Tornado Cash [Tor19] provide anonymity for transfers but exemplify the proof reusability limitation: all inputs must be fixed at proof time, making them unsuitable for dynamic

interactions that depend on fluctuating public data. Commit-and-prove schemes [CFQ19] support some dynamic behavior but require additional interaction rounds or complex setup procedures that are impractical for high-frequency trading applications.

Our Contribution: We present Zyga, an optimization of existing pairing-based SNARKs that addresses this fundamental limitation through dynamic public input substitution during verification. Our approach cleanly separates private and public witness components, specifically targeting Solana's computational constraints where G_2 scalar multiplications are expensive. Private values (such as trade amounts, slippage thresholds, or bidding strategies) are blinded via a trusted setup, ensuring computational indistinguishability, while public values (such as oracle-fed market rates, interest rates, or exchange prices) remain as symbolic placeholders that the verifier instantiates on-chain with current data. By enabling proof reusability with dynamic public inputs, Zyga unlocks numerous DeFi applications while providing inherent MEV protection through intent blinding. The system supports atomic execution in single transactions with succinct proofs and fast verification, optimized for Solana's environment using BN254 curves and one-sided pairings for efficient on-chain verification under asymmetric group operation constraints. Conceptually, Zyga can be viewed as a concrete generalization of Groth16 in which the prover still commits to the same QAP relation, but the verifier is allowed to instantiate a designated subset of inputs at verification time rather than fixing all inputs at proving time. Operationally, we assume that each application naturally decomposes its input into a long-lived, user-specific component w_{priv} (e.g., keys, limits, or positions) and a short-lived, environment-specific component w_{pub} (e.g., oracle prices or governance parameters). The Zyga Proofing system and security definitions are phrased in terms of this partition: w_{priv} is fixed at proof generation and never leaves the prover, while w_{pub} is reconstructed from on-chain state at verification time. Section 3.2 formalizes this as a public–private split of the R1CS witness and underpins all of our definitions of dynamic statements.

2 Related Work

The landscape of succinct and zero-knowledge proofs has evolved rapidly over the past decade. Pairing-based zk-SNARKs began with systems like Pinocchio [PHGR13] and matured into the highly efficient Groth16 [Gro16], which achieves constant-size proofs and fast verification at the cost of a circuit-specific trusted setup. The QAP (Quadratic Arithmetic Program) abstraction introduced by Gennaro et al. [GGPR13] provides the mathematical foundation for these constructions. For completeness, Section 4 recalls the R1CS and QAP abstractions and explains how they are used in Groth16-style zkSNARKs before introducing our compensation mechanism.

Universal constructions like Sonic [MBKM19] and Marlin [CHM$^+$20] generalized this line by enabling universal and updatable SRS (Structured Reference String), easing deployment concerns around circuit-specific setups. PLONK [GWC19] further simplified constraint systems via permutation arguments and also uses a universal updatable SRS. More recent works like PlonKish [GWC22] and Hyperplonk [CS22] continue this line of research with improved efficiency.

Transparent proof systems like Bulletproofs [BBB$^+$18] avoid trusted setup entirely and are practical for range proofs, though verification scales linearly in the statement size. STARKs [BSBHR18] provide post-quantum security under hash assumptions and transparent setup with quasi-linear prover time, trading larger proof sizes for these benefits. More recently, Halo [BGH19] and Halo2 [OH$^+$20] explore aggregation and recursion without trusted setup, building on polynomial commitment schemes like Bulletproofs.

Several works have explored adaptability in zero-knowledge systems, though none directly enable *non-interactive substitution of public inputs during verification.*

Commit-and-prove schemes [GKM22] allow proving statements about previously committed values without revealing them, supporting a limited form of dynamic behavior. In such schemes, the prover first publishes a commitment $C = g^v$ and later proves that the committed value v satisfies a predicate $P(v)$. Because the proof is cryptographically tied to a fixed commitment, any change to v requires either a new proof or an additional opening of the commitment. Thus, these systems achieve *composability under fixed commitments* but not *proof reusability under evolving public data.* A closely related direction is "updatable" or "dynamic" SNARKs such as NOTUS [XHTP24], which focus on updating proofs or statements via non-recursive state transitions, rather than on late substitution of public inputs within a single non-recursive proof. In contrast, Zyga keeps the Groth16-style structure and instead exploits the algebraic separation between private and

public QAP contributions to enable dynamic substitution at verification time. At a high level, NOTUS introduces an incremental SNARK that supports updating proofs as the underlying computation evolves, by proving append-only updates (e.g., via accumulator swaps) from a prior state digest. This is well-suited for long-lived, append-only computations such as blockchains or logs. Zyga, by contrast, keeps the underlying relation fixed and instead allows the concrete values of a designated public slice to be chosen *at verification time* without any state chaining. As a result, our trade-offs are closer to those of Groth16—one-shot, non-recursive proofs with constant size, while still enabling the verifier to plug in arbitrarily many different public-input vectors against the same proof.

Zyga mainly differs from previous systems in two essential ways:

1. **Non-interactive dynamic substitution.** Public inputs in Zyga appear as *symbolic placeholders* within the QAP polynomials and are instantiated only at verification. The verifier substitutes these values directly into the pairing equation, requiring no interaction or cryptographic openings.

2. **Algebraic compensation mechanism.** Rather than recomputing commitments for changed public inputs, Zyga applies the *Compensation Lemma* (Lemma 4.1), folding cross-terms $\Delta = (a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{priv}}$ into the private component C'_{priv}. This algebraic routing preserves QAP satisfiability while allowing the verifier to treat all b_{pub} terms as current on-chain data.

This design yields a property absent from previous frameworks: the ability to reuse a single proof across arbitrary updates of public data, so long as private inputs remain unchanged. Unlike commit-and-prove or modular SNARKs such as LegoSNARK [CGGN19], Zyga achieves this within a standard Groth16-style, one-message argument.

Updatable and universal SNARKs such as Sonic [MBKM19] and Marlin [CHM+20] address a different notion of adaptability, updating the structured reference string (SRS) rather than the statement itself. These constructions allow flexible circuit reuse but do not permit late substitution of inputs.

Detailed comparison with updatable/dynamic SNARKs. Table 1 provides a systematic comparison of Zyga against prior systems that support some form of adaptability:

- **NOTUS** [XHTP24]: Targets *proofs of liabilities* where proofs evolve as new entries are appended (e.g., exchange balances). Uses RSA accumulators and supports incremental updates via state digests. However, NOTUS proves statements about evolving *data structures*, whereas Zyga proves fixed relations with substitutable *input values*. NOTUS requires $O(\log n)$ proof updates per epoch; Zyga requires no update—the same proof works for any public input substitution.

- **Sonic** [MBKM19]: Universal SRS supporting any circuit up to a size bound. Sonic enables *circuit flexibility* (different computations, same SRS) but all inputs must still be fixed at proving time. Verification requires polynomial evaluation proofs, increasing verifier work.

- **Marlin** [CHM+20]: Improves on Sonic with better prover efficiency while maintaining universal SRS. Like Sonic, provides circuit reuse but not dynamic input substitution.

- **PLONK** [GWC19]: Universal SRS with efficient custom gates via permutation arguments. The universal setup eliminates per-circuit trusted ceremonies, but inputs remain static. Verification involves polynomial commitment openings, increasing on-chain cost relative to Groth16.

- **Commit-and-Prove** [GKM22]: Enables proving about previously committed values. Supports composability (commit once, prove multiple properties) but not substitution—the committed value is fixed. Changing the value requires a new commitment and proof.

Zyga is unique in enabling *non-interactive substitution of public inputs during verification* without requiring proof updates, commitment openings, or multi-round protocols. This makes it particularly suited to DeFi applications where public parameters (prices, rates) change frequently while private positions remain stable.

In summary, Zyga introduces a distinct algebraic technique: *dynamic public input substitution through one-sided QAP compensation*. It remains compatible with pairing-based zkSNARKs while providing the first non-interactive mechanism for dynamic statement evaluation in practical blockchain settings.

Several projects have focused on optimizing zero-knowledge proofs for specific blockchain environments. Aztec [Azt21] implements privacy-preserving transactions on Ethereum using Groth16 proofs optimized for

System	Setup	Dynamic Input	Proof Update	Verification
Groth16 [Gro16]	Circuit-specific	No	N/A	3 pairings
Sonic [MBKM19]	Universal	No	N/A	Poly. openings
Marlin [CHM+20]	Universal	No	N/A	Poly. openings
PLONK [GWC19]	Universal	No	N/A	Poly. openings
NOTUS [XHTP24]	Circuit-specific	State evolution	Per-epoch	RSA + pairings
Commit-and-Prove	Varies	Committed value	New proof	Varies
Zyga	Circuit-specific	**Yes (public)**	**None**	3 pairings

Table 1: Comparison with updatable and adaptive zkSNARK systems. "Dynamic Input" indicates whether inputs can change post-proving. "Proof Update" indicates what work is needed when inputs change.

the EVM's gas model. ZCash [BSCG+14] pioneered the use of SNARKs for cryptocurrency privacy, evolving from PGHR13 [PHGR13] to Groth16 and eventually to Halo2.

For Solana specifically, several challenges arise from the platform's computational model. The Solana runtime provides precompiles for specific elliptic curve operations [Sol23b] but has limitations on complex G_2 scalar multiplications that affect SNARK verification efficiency. Prior work has explored these constraints [Sol23a] without explicitly addressing the dynamic input problem.

The maximal extractable value (MEV) problem has received significant attention in recent literature. Daian et al. [DGK+19] first formalized frontrunning and transaction reordering attacks in decentralized exchanges. Subsequent work has quantified the scale of MEV extraction [QZG+22, WTNRS22] and proposed various mitigation strategies.

Privacy-preserving trading solutions include mixers like Tornado Cash [Tor19], private mempools like Flashbots [Fla21], and commit-reveal schemes [ZDGJ21]. However, these approaches either require multiple transactions, introduce latency, or don't support the dynamic price updates necessary for efficient market-making.

3 System Model and Definitions

3.1 Blockchain Execution Model

We model blockchain execution as a sequence of atomic transactions, each potentially reading current on-chain state (including oracle prices, interest rates, and other market data) and updating the contract state. Unlike traditional cryptographic settings where all inputs are known at proof generation time, blockchain applications require proofs that can incorporate real-time data during verification.

Definition 3.1 (Dynamic Statement)**.** *A dynamic statement consists of the following:*

- *Static components x_{static} fixed at proof generation*
- *Dynamic components $x_{dynamic}$ instantiated at verification time*
- *Relation $\mathcal{R}$ such that $(x_{static}, x_{dynamic}, w) \in \mathcal{R}$ for witness w*

Compensation. Define $\Delta \coloneqq (a_{\text{priv}} + a_{\text{pub}})\, b_{\text{priv}}$. Set $C'_{\text{priv}} \leftarrow C_{\text{priv}} - \Delta$ and $C'_{\text{pub}} \leftarrow C_{\text{pub}}$. This eliminates the need for the structural assumption b_{priv} while preserving one-sided verification.

3.2 Constraint System with Public-Private Partition

Definition 3.2 (R1CS with Public-Private Partition)**.** *A Rank-1 Constraint System over field $\mathbb{F}_r$ consists of matrices $A, B, C \in \mathbb{F}_r^{m \times n}$ with witness vector $w \in \mathbb{F}_r^n$ partitioned as:*

- *Index sets: $I_{priv} \cup I_{pub} = [n]$, $I_{priv} \cap I_{pub} = \emptyset$*
- *Private witness: $w_{priv} = (w_i)_{i \in I_{priv}}$ (known only to the prover)*

- *Public witness:* $w_{pub} = (w_i)_{i \in I_{pub}}$ *(obtained by verifier on-chain)*
- *Constraint:* $Aw \circ Bw = Cw$ *where* $\circ$ *denotes the element-wise product*

This partition enables our main contribution: private components are cryptographically blinded during proof generation, while public components remain symbolic and are instantiated with current values during verification. Throughout Sections 3.2–4 we reserve A, B, C for such *matrix*-valued objects and denote the corresponding QAP polynomials by $A_i(x), B_i(x), C_i(x)$ (or, equivalently, $v_k(x), w_k(x), y_k(x)$) obtained from these rows. Likewise, w always denotes a vector of field elements; when we later refer to families of polynomials indexed by k we write $\{w_k(x)\}$ to avoid overloading the same symbol. In particular, whenever A, B, C appear in algebraic lemmas, they should be read as the original R1CS matrices, and the QAP side is always written with explicit polynomial notation. In the applications we target (trading, lending, bridges, auctions) there is a natural semantic split of the witness: long-lived, user-specific quantities such as secret keys, position sizes, or risk limits are modeled as w_{priv}, whereas short-lived, globally observable parameters such as oracle prices, governance thresholds, or interest rates are modeled as w_{pub}. Our adversarial model (Definition 3.4) allows the adversary to choose both parts, but the implementation discipline is that only w_{pub} is reconstructed from on-chain state at verification time, while w_{priv} never leaves the prover.

3.3 Elliptic Curve Groups

The system operates over the BN254 curve with Type III pairing, chosen for compatibility with Solana's precompiles:

- G_1: Points on the curve $E(\mathbb{F}_q) : y^2 = x^3 + 3$
- G_2: Points on the twisted curve $E'(\mathbb{F}_{q^2}) : y^2 = x^3 + 3/(9 + u)$
- G_T: Multiplicative group of the 12th cyclotomic extension $\mathbb{F}^*_{q^{12}}$

Definition 3.3 (Optimal Pairing)**.** *The optimal pairing* $e : G_1 \times G_2 \to G_T$ *satisfies:*

- *Bilinearity:* $e(aP, bQ) = e(P, Q)^{ab}$ *for all* $a, b \in \mathbb{F}_r$, $P \in G_1$, $Q \in G_2$
- *Non-degeneracy: If* $e(P, Q) = 1$ *for all* $P \in G_1$, *then* $Q = \mathcal{O}$
- *Computability: The pairing is efficiently computable*

3.4 Security Model

Definition 3.4 (Adversarial Model)**.** *The adversary* $\mathcal{A}$ *is probabilistic polynomial-time (PPT) and may adaptively choose statements* x *and public inputs for which it attempts to produce accepting proofs. We assume an honestly-generated SRS where the toxic waste is destroyed; otherwise, security falls back to knowledge-soundness under structured reference strings. Here, the toxic waste consists of the trapdoor values* (α, β, τ) *embedded in the SRS; "destroyed" means that no party retains these values after the setup ceremony, even with unbounded computation.*

Our security model accounts for the dynamic nature of public inputs by allowing the adversary to choose both the statement and the public inputs used during verification, subject only to the constraint that public inputs must be well-formed field elements.

4 Mathematical Foundations

This section provides a self-contained introduction to Rank-1 Constraint Systems (R1CS) and Quadratic Arithmetic Programs (QAP), the mathematical foundations underlying Groth16-style zkSNARKs and, by extension, our Zyga construction. Readers already familiar with these concepts may skip to Section 4.2.

4.1 Preliminaries: From Computation to Constraints

Modern zkSNARKs prove that a computation was performed correctly without revealing the inputs. The standard approach proceeds in three steps: (1) express the computation as an arithmetic circuit, (2) convert the circuit into a constraint system (R1CS), and (3) encode the constraints as polynomial identities (QAP) that can be verified via pairings.

4.1.1 Rank-1 Constraint Systems (R1CS)

A *Rank-1 Constraint System* is a way of representing computations as a system of bilinear equations over a finite field $\mathbb{F}_r$.

Definition 4.1 (R1CS—Expanded)**.** *An R1CS instance over field $\mathbb{F}_r$ consists of:*

- *Three matrices $A, B, C \in \mathbb{F}_r^{m \times n}$, where m is the number of constraints and n is the number of variables (including a constant "1" variable).*
- *A* witness *vector $w = (1, w_1, \ldots, w_{n-1}) \in \mathbb{F}_r^n$, where the first entry is always 1.*
- *The constraint system is* satisfied *if and only if $(Aw) \circ (Bw) = Cw$, where $\circ$ denotes element-wise (Hadamard) multiplication.*

Equivalently, for each constraint $j \in [m]$:

$$\left(\sum_{i=0}^{n-1} A_{j,i} \cdot w_i\right) \cdot \left(\sum_{i=0}^{n-1} B_{j,i} \cdot w_i\right) = \sum_{i=0}^{n-1} C_{j,i} \cdot w_i.$$

Each constraint represents exactly one multiplication gate from the original circuit, with the left and right inputs encoded in A and B, and the output encoded in C. Addition gates require no constraints—they are "free" and encoded directly in the matrix coefficients.

Example. Consider proving knowledge of x such that $x^3 + x + 5 = 35$. We introduce intermediate variables: let $w_1 = x$, $w_2 = x^2$, $w_3 = x^3$, $w_4 = x^3 + x + 5$. The constraints are:

$$\begin{aligned} w_1 \cdot w_1 &= w_2 \quad \text{(first multiplication)} \\ w_2 \cdot w_1 &= w_3 \quad \text{(second multiplication)} \\ (w_3 + w_1 + 5) \cdot 1 &= w_4 \quad \text{(collecting the result)} \end{aligned}$$

The final public constraint $w_4 = 35$ is checked directly. This gives $m = 3$ constraints over $n = 5$ variables (including the constant 1).

4.1.2 Quadratic Arithmetic Programs (QAP)

The key insight of QAP [GGPR13] is that checking m constraints individually can be replaced by checking a *single polynomial identity*, which can then be verified efficiently using pairings.

Definition 4.2 (QAP—Expanded)**.** *Given an R1CS instance (A, B, C) with m constraints over n variables, the corresponding QAP is constructed as follows:*

1. *Choose m distinct evaluation points $\omega_1, \ldots, \omega_m \in \mathbb{F}_r$ (typically roots of unity).*
2. *For each variable index $i \in \{0, \ldots, n-1\}$, define polynomials $A_i(x), B_i(x), C_i(x) \in \mathbb{F}_r[x]$ of degree at most $m - 1$ such that:*
$$A_i(\omega_j) = A_{j,i}, \quad B_i(\omega_j) = B_{j,i}, \quad C_i(\omega_j) = C_{j,i}$$
for all $j \in [m]$. These polynomials are constructed via Lagrange interpolation.
3. *Define the* vanishing polynomial $t(x) = \prod_{j=1}^{m}(x - \omega_j)$, *which has degree m and vanishes at all evaluation points.*

Given a witness $w = (w_0, w_1, \ldots, w_{n-1})$, define the aggregate polynomials:

$$A(x) = \sum_{i=0}^{n-1} w_i \cdot A_i(x), \quad B(x) = \sum_{i=0}^{n-1} w_i \cdot B_i(x), \quad C(x) = \sum_{i=0}^{n-1} w_i \cdot C_i(x).$$

The R1CS constraints are satisfied if and only if:

$$A(x) \cdot B(x) - C(x) \equiv 0 \pmod{t(x)},$$

i.e., there exists a quotient polynomial $H(x)$ *such that* $A(x) \cdot B(x) - C(x) = H(x) \cdot t(x)$.

Why this works. At each evaluation point ω_j, the polynomial identity $A(\omega_j) \cdot B(\omega_j) = C(\omega_j)$ holds if and only if the j-th R1CS constraint is satisfied. If all constraints are satisfied, the polynomial $A(x) \cdot B(x) - C(x)$ vanishes at all ω_j, so it must be divisible by $t(x)$.

From polynomials to pairings. The Groth16 protocol [Gro16] uses the QAP representation to construct succinct proofs. The prover commits to evaluations of $A(\tau), B(\tau), C(\tau), H(\tau)$ at a secret challenge point τ (embedded in the SRS), and the verifier checks the polynomial identity via a single pairing equation:

$$e([A(\tau)]_1, [B(\tau)]_2) = e([C(\tau)]_1, [1]_2) \cdot e([H(\tau)]_1, [t(\tau)]_2),$$

where $[\cdot]_1$ and $[\cdot]_2$ denote group elements in G_1 and G_2 respectively. Soundness follows from the Schwartz–Zippel lemma: a false polynomial identity would require finding τ (breaking discrete log) or producing a non-trivial factor (breaking q-SDH).

4.1.3 How Zyga Extends Groth16

Zyga builds on the QAP/Groth16 framework with one key modification: we partition the witness into private and public components, and defer the public contributions until verification time. The Compensation Lemma (Section 4.2) shows how to algebraically reroute cross-terms so that:

1. Private contributions are committed in G_1 during proving (blinded under α).
2. Public contributions remain as symbolic coefficients that the verifier instantiates on-chain.
3. The pairing equation remains sound and complete for the original R1CS relation.

This separation enables dynamic public input substitution while preserving the Groth16 security guarantees.

Notation Summary. Throughout the remainder of this section, A, B, C denote the R1CS matrices from Section 3.2. The QAP polynomials derived from their rows are written $A_i(x), B_i(x), C_i(x)$. We use w only for the witness vector, while polynomial families use braces (e.g., $\{v_k(x)\}$). The vanishing polynomial is $t(x)$, the quotient is $H(x)$, and $\mathrm{HZ}(x)$ denotes their raw product used in the Compensation Lemma.

4.2 Algebraic Compensation Analysis

Compensation preserves QAP satisfiability.

Lemma 4.1 (Compensation Lemma)**.** *Let (A, B, C) be an R1CS instance with QAP polynomials $(\{v_k\}, \{w_k\}, \{y_k\}, t)$ over field $\mathbb{F}_r$, and let the witness be partitioned as $w = w_{\mathsf{priv}} \cup w_{\mathsf{pub}}$. Write the QAP evaluations at the hidden challenge x as*

$$a = a_{\mathsf{priv}} + a_{\mathsf{pub}}, \quad b = b_{\mathsf{priv}} + b_{\mathsf{pub}}, \quad c = c_{\mathsf{priv}} + c_{\mathsf{pub}}.$$

Define the compensation term $\Delta := (a_{\mathsf{priv}} + a_{\mathsf{pub}})\, b_{\mathsf{priv}}$ and set

$$c'_{\mathsf{priv}} \;:=\; c_{\mathsf{priv}} - \Delta \qquad \textit{and} \qquad c'_{\mathsf{pub}} \;:=\; c_{\mathsf{pub}}.$$

Then the following are equivalent:

$$\textit{(i)}\ a\,b - c \;\equiv\; 0 \pmod{t(x)} \qquad \Longleftrightarrow \qquad \textit{(ii)}\ (a_{\mathsf{priv}} + a_{\mathsf{pub}})\, b_{\mathsf{pub}} - (c'_{\mathsf{priv}} + c'_{\mathsf{pub}}) \;\equiv\; 0 \pmod{t(x)}.$$

In particular, QAP satisfiability of (A, B, C) at x holds iff QAP satisfiability holds for the compensated triple (A, B, C') with $C' := C - \Delta$ and the verifier using only b_{pub}.

Proof. Expand

$$a\,b - c = (a_{\mathsf{priv}} + a_{\mathsf{pub}})(b_{\mathsf{priv}} + b_{\mathsf{pub}}) - (c_{\mathsf{priv}} + c_{\mathsf{pub}}) = \underbrace{(a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{priv}}}_{\Delta} + (a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{pub}} - (c_{\mathsf{priv}} + c_{\mathsf{pub}}).$$

Rearranging,

$$(a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{pub}} - (c_{\mathsf{priv}} - \Delta + c_{\mathsf{pub}}) \;=\; (a\,b - c).$$

Thus $(a\,b - c) \equiv 0 \pmod{t(x)}$ iff $(a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{pub}} - (c'_{\mathsf{priv}} + c'_{\mathsf{pub}}) \equiv 0 \pmod{t(x)}$, with $c'_{\mathsf{priv}} := c_{\mathsf{priv}} - \Delta$ and $c'_{\mathsf{pub}} := c_{\mathsf{pub}}$. This algebraic identity holds pointwise at x and therefore modulo $t(x)$ as well. □

Theorem 4.1 (One-sided verification with algebraic compensation)**.** *Under the setup of the lemma, if the prover commits to* $A_{\mathsf{priv}} \propto g_1^{\alpha a_{\mathsf{priv}}}$, $C'_{\mathsf{priv}} \propto g_1^{\alpha c'_{\mathsf{priv}}}$, *and* $HZ_{\mathsf{priv}}(\tau)$ *in* G_1, *and the verifier instantiates the public parts to form*

$$A_{\mathsf{total}} \;:=\; A_{\mathsf{priv}} \cdot g_1^{a_{\mathsf{pub}}}, \qquad B_{\mathsf{pub}} \;:=\; g_2^{b_{\mathsf{pub}}}, \qquad C_{\mathsf{total}} \;:=\; C'_{\mathsf{priv}} \cdot g_1^{c'_{\mathsf{pub}}},$$

then the pairing check

$$e(A_{\mathsf{total}},\, B_{\mathsf{pub}}) \cdot e(-C_{\mathsf{total}},\, g_2) \cdot e(-g_1^{HZ(\tau)},\, g_2) \;\stackrel{?}{=}\; 1$$

is sound and complete for the original QAP relation $a\,b - c = H(\tau)\,t(\tau)$.

Proof linking to Groth16. By the Compensation Lemma, $(a\,b - c) \equiv 0 \pmod{t(\tau)}$ iff $(a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{pub}} - (c'_{\mathsf{priv}} + c'_{\mathsf{pub}}) \equiv 0 \pmod{t(\tau)}$. The pairing equation above encodes exactly this compensated identity at τ. Soundness and completeness of checking a single evaluation at a hidden point via pairings follow by the standard Groth16 argument that (i) binds polynomial evaluations through SRS powers and (ii) reduces false identities to nontrivial polynomial disagreements caught with overwhelming probability (Schwartz–Zippel), combined with the Groth16 pairing equation that enforces $a(\tau)\,b(\tau) - c(\tau) = H(\tau)\,t(\tau)$ (cf. Groth16 [Gro16] , Theorem 2 / main verification equation). Hence, the one-sided check is algebraically equivalent to the classical verification of the original (uncompensated) QAP relation. □

Remark 1 (Commitment routing and zero-knowledge)**.** *The cross-terms absorbed by* c'_{priv} *are* blinded *under the* α *trapdoor in* G_1, *so the compensation does not leak structure beyond what standard Groth16 commitments reveal. This matches the blinding behavior of Groth16 commitments and preserves zero-knowledge under DL assumptions.*

Remark 2 (Cryptographic Protection and Special Case)**.** *The compensation mechanism preserves security through blinding with the* α *trapdoor. The missing cross-terms* Δ *are routed into* HZ_{priv}, *which also appears in the verification equation as* $\alpha \cdot HZ_{priv}$. *This ensures compensation terms remain cryptographically hidden under the Discrete Logarithm assumption.*

4.3 HZ Polynomial Construction

Definition 4.3 (HZ Polynomial)**.** *Let* (A, B, C) *be the QAP polynomials corresponding to an R1CS instance and let* $a(x) = \sum_i a_i A_i(x)$, $b(x) = \sum_i b_i B_i(x)$, *and* $c(x) = \sum_i c_i C_i(x)$ *be the standard QAP evaluations at a secret challenge* τ. *Partition the witness into private and public parts, and write the evaluations as*

$$a(x) = a_1(x) + a_2(x), \quad b(x) = b_1(x) + b_2(x), \quad c(x) = c_1(x) + c_2(x),$$

where the index 1 *collects private contributions and* 2 *collects public ones. The* HZ polynomial *is defined as*

$$\mathrm{HZ}(x) \;\stackrel{\text{def}}{=}\; P(x) = a(x)\,b(x) - c(x).$$

Relation to the vanishing polynomial. In the classical QAP view, one computes the HZ $H(x)$ so that $P(x) = t(x)\,H(x)$ where $t(x)$ is the vanishing polynomial of the evaluation domain. In our formulation, the commitment used in verification binds directly to $\mathrm{HZ}(x)$; operationally, this plays the same role as committing to $H(x)$ times $t(\tau)$ at the challenge point, while avoiding division artifacts and keeping the commitment entirely in G_1 on the prover side. This makes the scheme easier to execution environments with tight G_2 constraints.

4.3.1 Polynomial Factorization and Routing

Theorem 4.2 (HZ Decomposition)**.** *With the above notation,*

$$P(x) \;=\; (a_1 + a_2)(b_1 + b_2) - (c_1 + c_2) \;=\; \underbrace{a_1 b_1 - c_1}_{P_{\mathsf{pure\_priv}}} \;+\; \underbrace{a_1 b_2 + a_2 b_1}_{P_{\mathsf{mixed}}} \;+\; \underbrace{a_2 b_2 - c_2}_{P_{\mathsf{pure\_pub}}}.$$

Proof. Immediate from expanding the product and grouping terms by whether they involve only private variables, a mixture of private and public variables, or only public variables. □

Routing strategy. We route commitments to the appropriate groups as follows: $P_{\mathsf{pure\_priv}}$ and P_{mixed} are committed in G_1 via the private generator h_1, while $P_{\mathsf{pure\_pub}}$ uses the public generator g_1. The verifier forms $B_{\mathsf{pub}} \in G_2$ from public data only, requiring no G_2 scalar multiplications with private values.

4.3.2 One-Sided Verification Equation

Let $A_{\mathsf{curve}} \in G_1$ and $C'_{\mathsf{curve}} \in G_1$ denote the prover's commitments that depend only on private data, and let $g_1^{a_2}$ and $g_1^{c_2}$ be the public updates computed at verification time. Define

$$A_{\mathsf{total}} \leftarrow A_{\mathsf{curve}} \cdot g_1^{a_2}, \qquad C_{\mathsf{total}} \leftarrow C'_{\mathsf{curve}} \cdot g_1^{c_2}, \qquad g_1^{\mathrm{HZ}_1} \leftarrow \text{commitment in } G_1 \text{ to } \mathrm{HZ}(x).$$

If $B_{\mathsf{pub}} = g_2^{b_2}$ is the public-side G_2 element, the verifier checks

$$e(A_{\mathsf{total}}, B_{\mathsf{pub}}) \cdot e(-C_{\mathsf{total}}, g_2) \cdot e(-g\mathrm{HZ}_1, g_2) \stackrel{?}{=} 1.$$

This is algebraically equivalent to checking $P(\tau) = 0$ and corresponds to the "one-sided" pairing described in Def. 5.1.

4.4 Polynomial Identity Testing

Our construction relies on the Schwartz-Zippel lemma [DL78, Sch80, Zip79], which states that a nonzero polynomial of degree d evaluated at a random point from set S equals zero with probability at most $d/|S|$. This allows verifying polynomial identities like $P(x) = H(x)Z(x)$ by checking equality at a single hidden random point τ, with negligible soundness error when $|S| \gg d$. This technique is standard in zkSNARKs including Groth16 [Gro16].

4.5 Polynomial Representations

We build on standard QAP (Quadratic Arithmetic Program) constructions [GGPR13] that encode R1CS constraints as polynomial identities. Following [GGPR13], witness values are encoded via Lagrange interpolation over evaluation points, yielding witness polynomials $w_A(x)$, $w_B(x)$, $w_C(x)$ such that the constraint satisfaction condition $Aw \circ Bw = Cw$ translates to $w_A(x) \cdot w_B(x) - w_C(x) = H(x) \cdot t(x)$, where $t(x)$ is the vanishing polynomial over the constraint domain and $H(x)$ is the quotient polynomial. The key insight for our construction is that this polynomial formulation naturally supports partitioning witness contributions into private and public components, enabling our dynamic substitution mechanism. Throughout the rest of the paper we adhere to the following naming convention. The vanishing polynomial of the evaluation domain is always denoted by $t(x)$. The quotient $H(x)$ is defined implicitly via $w_A(x) \cdot w_B(x) - w_C(x) = H(x) \cdot t(x)$. The "raw" product $w_A(x) \cdot w_B(x) - w_C(x)$ is denoted by $\mathrm{HZ}(x)$ in Definition 4.3. Whenever $t(x)$, $H(x)$, or $\mathrm{HZ}(x)$ appear in lemmas or theorems (including Theorem 4.2 and the security proofs), they refer back to these fixed meanings. In particular, the symbols t, H, and HZ are never reused for unrelated polynomials, which avoids the ambiguity pointed out in the reviews.

The translation from R1CS to QAP follows standard techniques, with the key insight that our public-private partition naturally corresponds to different treatment of QAP variables during proof generation.

5 Zyga Protocol

5.1 High-Level Approach

Zyga optimizes standard zkSNARK verification by encoding private witness components only in G_1 (one-sided encoding) while handling public components symbolically. This addresses Solana's constraint that G_2 scalar multiplications are computationally expensive, limiting practical circuit sizes.

The key insight is that many privacy-preserving applications naturally satisfy the constraint required for one-sided encoding: private inputs (amounts, thresholds, flags) typically appear additively or in the "multiplicand" position of constraints, rather than as multiplicative factors that would require G_2 encoding.

Definition 5.1 (One-Sided Constraint System)**.** *A constraint system supports one-sided encoding if it can be written such that:*

- *Private variables appear only in positions that map to G_1 encodings*
- *Public variables can appear in any position*
- *The resulting verification equation requires only g_2 operations (no G_2 scalar multiplications)*

5.2 Trusted Setup

Definition 5.2 (Trusted Setup Parameters)**.** *Generate random secrets $\alpha, \beta \leftarrow \mathbb{F}_r$ and evaluation point $\tau \leftarrow \mathbb{F}_r$. Compute:*

- *Private generators: $h_1 = g_1^{\alpha} \in G_1$, $h_2 = g_2^{\beta} \in G_2$*
- *QAP evaluations: $\{v_k(\tau), w_k(\tau), y_k(\tau)\}$ for constraint polynomials*
- *Powers: $\{g_1^{\tau^j}, g_2^{\tau^j}\}$ for polynomial evaluation*
- *Target evaluation: $t(\tau)$ for the vanishing polynomial*

Critical: *Destroy α, β, τ after setup generation.*

Assumption 5.1 (Trusted Setup)**.** *The discrete logarithms α, β, τ are information-theoretically destroyed after setup generation.*

The trusted setup follows standard techniques from Groth16 but with additional structure to support dynamic public input substitution. The key difference is that public input positions are treated specially in the SRS generation.

5.3 Polynomial Encoding and Evaluation

Following the QAP construction, we evaluate witness polynomials at the hidden point τ:

Definition 5.3 (Witness Polynomial Decomposition)**.** *At evaluation point τ:*

$$w_A(\tau) = a_{priv} + a_{pub} \tag{1}$$
$$w_B(\tau) = b_{priv} + b_{pub} \tag{2}$$
$$w_C(\tau) = c_{priv} + c_{pub} \tag{3}$$

where private contributions $(a_{priv}, b_{priv}, c_{priv})$ depend only on private witness components, and public contributions $(a_{pub}, b_{pub}, c_{pub})$ depend only on public witness components.

This decomposition is the foundation of our approach: private contributions are cryptographically blinded during proof generation, while public contributions remain symbolic and are computed during verification using current public inputs.

5.4 One-Sided Encoding

Standard pairing-based verification requires encoding witness contributions in both G_1 and G_2:

$$e(h_1^{a_{\text{priv}}} \cdot g_1^{a_{\text{pub}}}, h_2^{b_{\text{priv}}} \cdot g_2^{b_{\text{pub}}}) = e(h_1^{c_{\text{priv}}} \cdot g_2^{c_{\text{pub}}})$$

This expands to include cross-terms $e(h_1, h_2)^{a_{\text{priv}} b_{\text{priv}}}$ that are cryptographically sound but computationally expensive on Solana due to G_2 scalar multiplication requirements.

One-sided verification (no structural restriction). The prover applies a compensation step that folds $\Delta = (a_{\text{priv}} + a_{\text{pub}})\, b_{\text{priv}}$ into C'_{priv} (and the private HZ commitment), so the verifier can keep the G_2 input purely public while the pairing equation still enforces $P(\tau) = 0$.

This eliminates expensive G_2 operations while maintaining security, as the verification equation reduces to:

$$(a_{\text{priv}} + a_{\text{pub}}) \cdot b_{\text{pub}} = c_{\text{priv}} + c_{\text{pub}} + H \cdot t(\tau)$$

Corollary 5.1 (Degenerate one-sided verification without compensation)**.** *Let the constraint system satisfy the structural restriction that $b_{\mathsf{priv}} = 0$, i.e., no private witness components appear in the B matrix of the R1CS instance. Let the QAP polynomials be evaluated as*

$$a(x) = a_{\mathsf{priv}}(x) + a_{\mathsf{pub}}(x), \qquad b(x) = b_{\mathsf{pub}}(x), \qquad c(x) = c_{\mathsf{priv}}(x) + c_{\mathsf{pub}}(x).$$

Then the verifier's one-sided pairing check

$$e(A_{\mathsf{total}},\, B_{\mathsf{pub}}) \cdot e(-C_{\mathsf{total}},\, g_2) \cdot e\Big(-g_1^{H_Z(x)},\, g_2\Big) \stackrel{?}{=} 1$$

with

$$A_{\mathsf{total}} = A_{\mathsf{priv}} \cdot g_1^{a_{\mathsf{pub}}}, \qquad B_{\mathsf{pub}} = g_2^{b_{\mathsf{pub}}}, \qquad C_{\mathsf{total}} = C_{\mathsf{priv}} \cdot g_1^{c_{\mathsf{pub}}},$$

is algebraically equivalent to verifying

$$(a_{\mathsf{priv}} + a_{\mathsf{pub}})\, b_{\mathsf{pub}} = c_{\mathsf{priv}} + c_{\mathsf{pub}} + H_Z(x)\, t(x)$$

and thus enforces $P(x) = 0$ at $x = \tau$.

Proof. This case follows directly from the Compensation Lemma (Lemma 4.1), setting $b_{\mathsf{priv}} = 0$. The compensation term $\Delta = (a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{priv}}$ vanishes, so the compensated polynomial C' coincides with C. Hence the verifier's pairing equation reduces to the standard Groth16-style verification equation with public B input only, ensuring soundness and completeness by the same argument as in Groth [Gro16]. □

The constraint b_{priv} is natural for many applications: private inputs like amounts, flags, or thresholds typically appear as additive terms or multiplicands rather than multipliers in constraint systems.

Example. A concrete R1CS construction satisfying the one-sided verification constraint ($b_{\mathsf{priv}} = 0$) is provided in Appendix A, illustrating how standard relations such as $a - b \geq 0$ can be encoded while supporting dynamic public input substitution.

5.5 Two-Sided Encoding

While the one-sided encoding (Corollary 5.1) is optimal when circuits satisfy $b_{\mathsf{priv}} = 0$, many practical applications require private witness components in all matrix positions. We now present *two-sided encoding*, an extension that removes this structural restriction while preserving dynamic public input substitution.

5.5.1 Motivation

Consider a constraint of the form:

$$\mathsf{private\_amount} \times \mathsf{private\_multiplier} = \mathsf{private\_result}$$

Such constraints arise naturally in:

- Portfolio calculations with private allocations and private prices
- Risk computations involving multiple private parameters
- Complex trading strategies with hidden quantities on both sides

The one-sided restriction ($b_{\mathsf{priv}} = 0$) prohibits these patterns. Two-sided encoding lifts this restriction by introducing an explicit private B commitment and a natural cross-term cancellation mechanism.

5.5.2 Extended Witness Decomposition

In two-sided encoding, we partition all three QAP evaluations into private and public components without restriction:

$$A = a_1\alpha + a_2 \tag{4}$$
$$B = b_1\beta + b_2 \tag{5}$$
$$C = c_1\alpha + c_2 \tag{6}$$

where:

- (a_1, b_1, c_1): Private witness polynomial evaluations, blinded by trapdoors α, β
- (a_2, b_2, c_2): Public witness values, substituted at verification time

Definition 5.4 (Two-Sided Proof Structure)**.** *A two-sided Zyga proof π consists of:*

1. ***Private A commitment:*** $\mathsf{a\_curve} = h_1^{a_1} \in G_1$
2. ***Private C commitment (adjusted):*** $\mathsf{c\_curve} = h_1^{c_{1,\mathsf{adj}}} \in G_1$
3. ***Private B commitment:*** $g_2^{b_1} \in G_2$ *(new element)*
4. ***Base public B commitment:*** $g_2^{b_{2,\mathsf{base}}} \in G_2$
5. ***Current public B commitment:*** $g_2^{b_{2,\mathsf{curr}}} \in G_2$
6. ***Public A binding:*** $[a_{2,\mathsf{base}}]G_1 \in G_1$
7. ***Public C binding:*** $[c_{2,\mathsf{base}}]G_1 \in G_1$
8. ***Private HZ commitment:*** $h_1^{\mathsf{HZ}_{\mathsf{priv}}} \in G_1$

Total proof size: 5 G_1 elements (320 bytes) + 3 G_2 elements (384 bytes) = 704 bytes.

5.5.3 Cross-Term Analysis

Expanding the QAP constraint $A \cdot B - C = 0$:

$$\begin{aligned} &(a_1 + a_2)(b_1 + b_2) - (c_1 + c_2) = 0 \\ \underbrace{a_1 b_1}_{\text{pure private}} &+ \underbrace{a_1 b_2 + a_2 b_1}_{\text{mixed}} + \underbrace{a_2 b_2}_{\text{pure public}} - c_1 - c_2 = 0 \end{aligned} \tag{7}$$

The challenge is handling the *mixed terms* $a_1 b_2 + a_2 b_1$ without requiring additional verification pairs.

5.5.4 Natural Cancellation Mechanism

Two-sided encoding achieves cross-term elimination through algebraic routing:

Lemma 5.1 (Natural Cross-Term Cancellation)**.** *Define the private adjustment:*

$$\Delta_{\mathsf{priv}} = (a_1 + a_2) \cdot b_1$$

and set:

$$c_{1,\mathsf{adj}} = c_1 - \Delta_{\mathsf{priv}} = c_1 - a_1 b_1 - a_2 b_1$$

Then the HZ *polynomial contains exactly $a_1 b_1 + a_2 b_1$, and the verification equation achieves natural cancellation:*

$$\begin{aligned} -C_{\mathsf{adj}} &\ \textit{contributes}\ + (a_1 b_1 + a_2 b_1) \\ -\mathsf{HZ} &\ \textit{contributes}\ - (a_1 b_1 + a_2 b_1) \\ \textit{Net:}&\quad 0 \end{aligned}$$

Proof. By construction, $C_{\mathsf{adj}} = h_1^{\alpha(c_1 - a_1 b_1 - a_2 b_1)}$. In the verification equation, we compute $-C_{\mathsf{adj}}$, which contributes the negation of this exponent in the pairing. The HZ commitment $h_1^{\mathsf{HZ}_{\mathsf{priv}}}$ contains all private-side quotient terms, including $(a_1+a_2)b_1$ from the product expansion. When combined as $e(-C_{\mathsf{adj}}, g_2) \cdot e(-\mathsf{HZ}, g_2)$, the cross-terms cancel algebraically. □

5.5.5 Two-Sided Verification Equation

The verification proceeds in three stages:

Stage 1: B-Sanity Checks. Verify that the G_2 commitments for b_2 match the public scalar inputs:

$$e(G_1, g_2^{b_2}) \stackrel{?}{=} e([b_2]G_1, G_2)$$

Stage 2: Base Bindings. Verify that the proof is bound to the correct base public values:

$$\mathsf{proof.g1\_a2} \stackrel{?}{=} [a_{2,\mathsf{base}}]G_1$$

$$\mathsf{proof.g1\_c2} \stackrel{?}{=} [c_{2,\mathsf{base}}]G_1$$

These bindings prevent an adversary from forging a proof for one set of base inputs and reusing it for an invalid set.

Stage 3: Main Verification. The aggregated pairing equation:

$$\begin{aligned} & e(\mathsf{a\_curve} + [a_{2,\mathsf{curr}}]G_1,\ g_2^{b_{2,\mathsf{curr}}}) && (8) \\ & \cdot\ e(-\mathsf{c\_curve}, G_2) \cdot e(-\mathsf{HZ}_{\mathsf{priv}}, G_2) && (9) \\ & \cdot\ e(-\mathsf{a\_curve}, g_2^{b_{2,\mathsf{curr}}}) \cdot e(\mathsf{a\_curve}, g_2^{b_{2,\mathsf{base}}}) && (10) \\ & \cdot\ e(-[\Delta a_2]G_1, g_2^{b_1}) && (11) \\ & \cdot\ e([a_{2,\mathsf{curr}}]G_1, g_2^{b_1}) && (12) \\ & \stackrel{?}{=} 1 \end{aligned}$$

Where:

- (8): Main product $A_{\mathsf{total}} \cdot B_{\mathsf{pub,curr}}$
- (9): Subtracts C and HZ components
- (10): Adjusts for change in b_2 between base and current
- (11): Corrects the $a_2 b_1$ term using the private B commitment, where $\Delta a_2 = a_{2,\mathsf{curr}} - a_{2,\mathsf{base}}$
- (12): Cancels the residual $a_{2,\mathsf{curr}} b_1$ term

5.6 Rebase Mechanism

A key feature of two-sided encoding is the *rebase* mechanism, which allows public inputs to change from their base values (fixed at proof generation) to current values (substituted at verification) while maintaining proof validity.

Definition 5.5 (Rebase Identity)**.** *For the public channel, given base values* $(a_{2,\mathsf{base}}, b_{2,\mathsf{base}}, c_{2,\mathsf{base}})$ *and current values* $(a_{2,\mathsf{curr}}, b_{2,\mathsf{curr}})$*, the rebased* C_2 *is:*

$$C_{2,\mathsf{rebase}} = c_{2,\mathsf{base}} + (a_{2,\mathsf{curr}} \cdot b_{2,\mathsf{curr}} - a_{2,\mathsf{base}} \cdot b_{2,\mathsf{base}})$$

Theorem 5.1 (Rebase Soundness)**.** *The rebase mechanism preserves QAP satisfiability. That is, if the original constraint* $a \cdot b - c = 0$ *holds for base values, then after rebasing:*

$$a_{\mathsf{curr}} \cdot b_{\mathsf{curr}} - C_{2,\mathsf{rebase}} = a_{\mathsf{base}} \cdot b_{\mathsf{base}} - c_{2,\mathsf{base}} = 0$$

Proof. Substituting the rebase definition:

$$\begin{aligned} a_{\mathsf{curr}} \cdot b_{\mathsf{curr}} - C_{2,\mathsf{rebase}} &= a_{\mathsf{curr}} \cdot b_{\mathsf{curr}} - c_{2,\mathsf{base}} - (a_{2,\mathsf{curr}} \cdot b_{2,\mathsf{curr}} - a_{2,\mathsf{base}} \cdot b_{2,\mathsf{base}}) \\ &= a_{2,\mathsf{base}} \cdot b_{2,\mathsf{base}} - c_{2,\mathsf{base}} \\ &= 0 \quad \text{(by original constraint satisfaction)} \end{aligned}$$

□

The rebase mechanism adds three verification terms to the public channel:

1. $e(-[C_{2,\mathsf{rebase}}]G_1, G_2)$
2. $e(-[a_{2,\mathsf{base}}]G_1, g_2^{b_{2,\mathsf{base}}})$
3. $e([c_{2,\mathsf{base}}]G_1, G_2)$

These ensure that the public polynomial S_{pub} cancels to zero algebraically for any valid public input transition.

5.7 Dynamic Public Input Substitution

The key innovation enabling dynamic public inputs is the separation of symbolic and concrete evaluation.

Symbolic Public Evaluation. During proof generation, public contributions are computed symbolically as $a_{\text{pub}}^{\text{symbolic}} = \sum_{i \in I_{\text{pub}}} A_i(\tau) \cdot \texttt{symbol}_i$, where $\texttt{symbol}_i$ are placeholder variables for public inputs.

Concrete Public Evaluation. During verification, these symbolic placeholders are substituted with current values: $a_{\text{pub}}^{\text{concrete}} = \sum_{i \in I_{\text{pub}}} A_i(\tau) \cdot w_{\text{pub},i}^{\text{current}}$, where $w_{\text{pub},i}^{\text{current}}$ are the public input values at verification time.

This separation enables proof reusability: a single proof remains valid as long as private inputs are unchanged, regardless of how public inputs evolve between proof generation and verification.

6 Security Analysis

Binding via HZ. Soundness reduces to the binding of the commitment $g\text{HZ}_1$ together with the linear independence of the SRS elements. Any adversary that forges a proof for $x \notin \mathcal{L}_{\mathcal{R}}$ would implicitly produce two distinct evaluations of $\text{HZ}(\tau)$ that verify under the same SRS, which we leverage via a forking-style argument to extract secret setup values, contradicting Assumption 5.1.

Algorithm 1 Zyga Protocol (General Compensated; no verifier use of b_{priv})

Setup$(1^\lambda,\ \text{R1CS}(A, B, C))$:

1. Sample $\alpha, \beta, \tau \leftarrow \mathbb{F}_r$.
2. Set $h_1 \leftarrow g_1^{\alpha},\ h_2 \leftarrow g_2^{\beta}$.
3. Compute QAP polynomials $\{v_k(x), w_k(x), y_k(x)\}$ and evaluation at τ.
4. (Optional) If circuits cannot enforce $B_i = 0$ for $i \in I_{\text{priv}}$, skip; compensation will be used.
5. Destroy α, β, τ and output srs.

Prove$(\mathsf{srs},\ w_{\text{priv}},\ \{w_{\text{pub}}^{\text{sym}}\})$:

1. Evaluate at τ:
$$a_{\text{priv}} = \sum_{i \in I_{\text{priv}}} A_i(\tau) w_{\text{priv},i}, \quad b_{\text{priv}} = \sum_{i \in I_{\text{priv}}} B_i(\tau) w_{\text{priv},i}, \quad c_{\text{priv}} = \sum_{i \in I_{\text{priv}}} C_i(\tau) w_{\text{priv},i}.$$
2. Fold private cross-terms: $C'_{\text{priv}} \leftarrow g_1^{\alpha(c_{\text{priv}} - a_{\text{priv}} b_{\text{priv}})}$.
3. Prepare compensation bases for public part of Δ: for each $i \in I_{\text{pub}}$,
$$\widehat{K}_{\Delta,i} \leftarrow g_1^{\alpha \cdot A_i(\tau) \cdot b_{\text{priv}}}.$$
Include $\{\widehat{K}_{\Delta,i}\}$ in the proof.
4. Commit: $A_{\text{priv}} \leftarrow g_1^{\alpha a_{\text{priv}}}$, $g_{HZ,1} \leftarrow g_1^{HZ_{\text{priv}}(\tau)}$.
5. Output proof $\pi \leftarrow (A_{\text{priv}}, C'_{\text{priv}}, g_{HZ,1}, \{\widehat{K}_{\Delta,i}\})$.

Verify$(\mathsf{srs},\ x,\ \pi,\ w_{\text{pub}}^{\text{curr}})$:

1. Parse $\pi = (A_{\text{priv}}, C'_{\text{priv}}, g_{HZ,1}, \{\widehat{K}_{\Delta,i}\})$.
2. Compute public evaluations with current inputs:
$$a_{\text{pub}} = \sum_{i \in I_{\text{pub}}} A_i(\tau) w_{\text{pub},i}, \quad b_{\text{pub}} = \sum_{i \in I_{\text{pub}}} B_i(\tau) w_{\text{pub},i}, \quad c_{\text{pub}} = \sum_{i \in I_{\text{pub}}} C_i(\tau) w_{\text{pub},i}.$$
3. Form combined elements:
$$A_{\text{total}} \leftarrow A_{\text{priv}} \cdot g_1^{a_{\text{pub}}}, \quad B_{\text{pub}} \leftarrow g_2^{b_{\text{pub}}},$$
$$C_{\text{total}} \leftarrow \left(C'_{\text{priv}} \cdot g_1^{c_{\text{pub}}}\right) \cdot \left(\prod_{i \in I_{\text{pub}}} \widehat{K}_{\Delta,i}^{w_{\text{pub},i}}\right)^{-1}.$$
Update $g_{HZ,1} \leftarrow g_{HZ,1} \cdot g_1^{HZ_{\text{pub}}(\tau)}$.
4. Check pairing equation:
$$e(A_{\text{total}}, B_{\text{pub}}) \cdot e(-C_{\text{total}}, g_2) \cdot e(-g_{HZ,1}, g_2) \stackrel{?}{=} 1.$$

Algorithm 2 Zyga Protocol (Two-Sided Encoding with Rebase)

Setup$(1^\lambda,\ \mathrm{R1CS}(A, B, C))$:

1. Sample $\alpha, \beta, \tau \leftarrow \mathbb{F}_r$.
2. Set $h_1 \leftarrow g_1^{\alpha},\ h_2 \leftarrow g_2^{\beta}$.
3. Compute QAP polynomials $\{v_k(x), w_k(x), y_k(x)\}$ and evaluations at τ.
4. Compute public coefficient maps for on-chain reconstruction.
5. Destroy α, β, τ and output srs.

Prove$(\mathsf{srs},\ w_{\mathrm{priv}},\ w_{\mathrm{pub}}^{\mathrm{base}})$:

1. Evaluate private contributions at τ:

$$a_1 = \sum_{i \in I_{\mathrm{priv}}} A_i(\tau) w_i, \quad b_1 = \sum_{i \in I_{\mathrm{priv}}} B_i(\tau) w_i, \quad c_1 = \sum_{i \in I_{\mathrm{priv}}} C_i(\tau) w_i.$$

2. Evaluate public contributions at base values:

$$a_{2,\mathrm{base}} = \sum_{i \in I_{\mathrm{pub}}} A_i(\tau) w_{\mathrm{pub},i}^{\mathrm{base}}, \quad b_{2,\mathrm{base}}, \quad c_{2,\mathrm{base}} \text{ (similarly)}.$$

3. Compute private adjustment: $\Delta_{\mathrm{priv}} \leftarrow (a_1 + a_{2,\mathrm{base}}) \cdot b_1$.
4. Compute adjusted private C: $c_{1,\mathrm{adj}} \leftarrow c_1 - \Delta_{\mathrm{priv}}$.
5. Compute private HZ containing cross-terms.
6. Commit to proof elements:

$$\begin{aligned}
\mathsf{a_curve} &\leftarrow h_1^{a_1} \\
\mathsf{c_curve} &\leftarrow h_1^{c_{1,\mathrm{adj}}} \\
g_2^{b_1} &\leftarrow g_2^{b_1} \quad \text{(private B commitment)} \\
g_2^{b_{2,\mathrm{base}}} &\leftarrow g_2^{b_{2,\mathrm{base}}} \\
\mathsf{g1_a2} &\leftarrow [a_{2,\mathrm{base}}]G_1 \\
\mathsf{g1_c2} &\leftarrow [c_{2,\mathrm{base}}]G_1 \\
\mathsf{g1_hz_priv} &\leftarrow h_1^{\mathrm{HZ}_{\mathrm{priv}}}
\end{aligned}$$

7. Output proof π.

Verify$(\mathsf{srs},\ x,\ \pi,\ w_{\mathrm{pub}}^{\mathrm{curr}})$:

1. Parse proof π.
2. Compute current public evaluations:

$$a_{2,\mathrm{curr}},\ b_{2,\mathrm{curr}},\ c_{2,\mathrm{curr}} \text{ from } w_{\mathrm{pub}}^{\mathrm{curr}}.$$

3. **B-Sanity Check:** Verify $e(G_1, g_2^{b_{2,\mathrm{curr}}}) = e([b_{2,\mathrm{curr}}]G_1, G_2)$.
4. **Base Bindings:** Verify $\mathsf{g1\_a2} = [a_{2,\mathrm{base}}]G_1$ and $\mathsf{g1\_c2} = [c_{2,\mathrm{base}}]G_1$.
5. **Compute Rebase:**

$$C_{2,\mathrm{rebase}} \leftarrow c_{2,\mathrm{base}} + (a_{2,\mathrm{curr}} \cdot b_{2,\mathrm{curr}} - a_{2,\mathrm{base}} \cdot b_{2,\mathrm{base}}).$$

6. **Main Pairing Check:** Aggregate and verify:

$$e(\mathsf{a\_curve} + [a_{2,\mathrm{curr}}]G_1,\ g_2^{b_{2,\mathrm{curr}}})$$

6.1 Cryptographic Assumptions

Assumption 6.1 (Discrete Logarithm in G_1)**.** *For any PPT adversary $\mathcal{A}$ and random $\alpha \leftarrow \mathbb{F}_r$:*

$$\Pr[\mathcal{A}(g_1, g_1^{\alpha}) = \alpha] \leq \mathit{negl}(\lambda)$$

Assumption 6.2 (q-Strong Diffie-Hellman)**.** *For any PPT adversary $\mathcal{A}$, degree bound q, and random $\alpha \leftarrow_R \mathbb{F}_r$:*

$$\Pr[\mathcal{A}(g_1, g_1^{\alpha}, g_1^{\alpha^2}, \ldots, g_1^{\alpha^q}) \text{ outputs } (c, g_1^{1/(\alpha+c)})] \leq \mathit{negl}(\lambda)$$

for any $c \in \mathbb{F}_r$.

These assumptions are standard in pairing-based cryptography and are necessary for the security of underlying primitives like Groth16.

6.2 Security Properties

For clarity, we spell out the verifier algorithm $\mathsf{Verify}(\mathsf{srs}, x, \pi, w_{\text{pub}}^{\text{current}})$ that is referred to in all subsequent theorems. Given the structured reference string srs, a statement x fixing the circuit and public-input positions, a proof π, and a concrete vector of public inputs $w_{\text{pub}}^{\text{current}}$, the verifier (i) checks that all group elements in π lie in the correct groups and are not the point at infinity, (ii) recomputes $a_{\text{pub}}, b_{\text{pub}}, c_{\text{pub}}$ from $w_{\text{pub}}^{\text{current}}$ and the QAP coefficients contained in the SRS, (iii) forms $A_{\text{total}}, B_{\text{pub}}, C_{\text{total}}$ by combining the proof commitments with these evaluations, including the explicit compensation terms, and (iv) accepts if and only if the one-sided pairing equation from Algorithm 1 holds. No additional checks are hidden in the shorthand notation $\mathsf{Verify}(\cdot)$ used in Theorems 6.1 and 6.2.

Theorem 6.1 (Completeness)**.** *For any valid instance-witness pair $(x, w) \in \mathcal{R}$,*

$$\mathit{Verify}(\mathsf{srs}, x, \mathit{Prove}(\mathsf{srs}, w_{priv}, w_{pub}^{symbolic}), w_{pub}^{current}) = 1$$

with probability 1, in the general compensated setting.

Proof. By construction of the algorithm, when the R1CS constraints are satisfied and the prover performs the compensation step correctly, the polynomial relation $w_A(\tau) \cdot w_B(\tau) - w_C(\tau) = H(\tau) \cdot t(\tau)$ holds. The pairing equation directly encodes this relation:

$$e(g_1^{a_{\text{priv}}+a_{\text{pub}}}, g_2^{b_{\text{pub}}}) = e(g_1^{c_{\text{priv}}+c_{\text{pub}}}, g_2) \cdot e(g_1^{H(\tau)}, g_2)$$

which simplifies to the required constraint verification. □

Soundness proof structure. Our argument follows the classical reduction technique introduced by Groth [Gro16], adapted to the compensated QAP relation established in Lemma 4.1. Intuitively, if an adversary can produce an accepting proof for a false statement $x \notin L_R$, then from two accepting transcripts corresponding to distinct evaluation points $x = \tau$ and $x = \tau'$, one can extract the hidden trapdoor α embedded in the structured reference string. Since recovering α would solve the discrete logarithm problem in G_1, the existence of such an adversary would contradict the assumed hardness of that problem. We now formalize this argument.

Theorem 6.2 (Computational Soundness)**.** *Under the Discrete Logarithm assumption in G_1 and the q-Strong Diffie–Hellman assumption in G_1, the Zyga proof system is computationally sound. Formally, for any probabilistic polynomial-time adversary $\mathcal{A}$,*

$$\Pr\big[\mathsf{Verify}(\mathsf{srs}, x, \pi) = 1 \ \wedge \ x \notin L_R\big] \leq \mathsf{negl}(\lambda).$$

Proof. We build a reduction $\mathcal{B}$ that uses any adversary $\mathcal{A}$ breaking soundness to solve an instance of the discrete-logarithm problem in G_1.

Setup. $\mathcal{B}$ is given a DL challenge $(g_1,\, h_1 = g_1^{\alpha})$ and must recover α. It embeds h_1 as the private generator in the Zyga structured reference string: srs is generated exactly as in the real scheme except that $\mathcal{B}$ does not know α. It samples fresh $\beta, \tau \leftarrow \mathbb{F}_r$ and constructs all other SRS elements normally (powers of g_1, g_2, evaluations of $A(x), B(x), C(x)$, etc.).

Adversary phase. $\mathcal{A}$ outputs a statement $x \notin L_R$ and a proof $\pi = (A_{\mathsf{priv}}, C'_{\mathsf{priv}}, g_1^{H_Z}, \{K_{\Delta,i}\})$ that passes verification:

$$e(A_{\mathsf{total}}, B_{\mathsf{pub}}) \cdot e(-C_{\mathsf{total}}, g_2) \cdot e(-g_1^{H_Z}, g_2) = 1. \tag{1}$$

Extractor construction. Let $\mathcal{B}$ rewind $\mathcal{A}$ to the point where the hidden evaluation challenge $x = \tau$ is sampled. $\mathcal{B}$ then re-runs $\mathcal{A}$ with a fresh, independent random $\tau' \neq \tau$, obtaining a second accepting proof π' that also satisfies a pairing check of the form (1), now evaluated at the independent challenge τ'.

By the bilinearity of $e(\cdot,\cdot)$ and the algebraic form of Zyga's commitments (cf. Lemma 4.1), each pairing equation enforces a linear relation in α of the form

$$\alpha \cdot f(\tau) = g(\tau), \qquad \alpha \cdot f(\tau') = g(\tau'),$$

for known field elements $f(\cdot), g(\cdot)$ computable from the public proof components.

Subtracting and dividing these two equations gives

$$\alpha = \frac{g(\tau) - g(\tau')}{f(\tau) - f(\tau')}. \tag{3}$$

Since $f(\tau) \neq f(\tau')$ except with negligible probability (over random τ, τ' and distinct proofs), $\mathcal{B}$ recovers the discrete logarithm α with probability at least $(\varepsilon^2/2) - \mathsf{negl}(\lambda)$, where $\varepsilon = \Pr[\mathcal{A} \text{ succeeds}]$. This contradicts the assumed hardness of the discrete logarithm in G_1.

Conclusion. Hence ε must itself be negligible, and no efficient adversary can produce an accepting proof for $x \notin L_R$ with non-negligible probability. Therefore, Zyga is computationally sound under the stated assumptions. □

Theorem 6.3 (Computational Zero-Knowledge)**.** *Under Assumption 6.1, there exists a PPT simulator $\mathcal{S}$ such that for any PPT distinguisher $\mathcal{D}$:*

$$\left|\Pr[\mathcal{D}(\mathit{View}_{\mathcal{A}}^{\mathit{Real}}(x,w)) = 1] - \Pr[\mathcal{D}(\mathit{View}_{\mathcal{A}}^{\mathit{Sim}}(x)) = 1]\right| \leq \mathit{negl}(\lambda)$$

where $\mathit{View}_{\mathcal{A}}^{\mathit{Real}}(x,w)$ is the adversary's view when interacting with the honest prover, and $\mathit{View}_{\mathcal{A}}^{\mathit{Sim}}(x)$ is the view when interacting with the simulator.

Proof. We construct a simulator $\mathcal{S}$ that can produce accepting proofs without knowing the private witness.

Simulator Construction: Given statement x (without witness w) and SRS, simulator $\mathcal{S}$ operates as follows:

Step 1 (SRS Programming): In the simulation security model, $\mathcal{S}$ generates the SRS while retaining knowledge of trapdoors α, β, τ. This is standard in zkSNARK simulation.

Step 2 (Public Component Computation): For public witness components, $\mathcal{S}$ computes these deterministically from statement x using the constraint matrices, exactly as an honest prover would.

Step 3 (Private Component Simulation): $\mathcal{S}$ samples random field elements $\tilde{a}_{\text{priv}}, \tilde{c}_{\text{priv}} \leftarrow_R \mathbb{F}_r$ and computes:

$$A^*_{\text{priv}} = g_1^{\alpha \tilde{a}_{\text{priv}}} \tag{13}$$

$$C^*_{\text{priv}} = g_1^{\alpha \tilde{c}_{\text{priv}}} \tag{14}$$

Step 4 (Consistency Enforcement): Using knowledge of α, $\mathcal{S}$ computes $HZ^*_{extpriv}$ such that the verification equation holds:

$$HZ^*_{extpriv} = \tilde{a}_{\text{priv}} b_{\text{pub}} - \tilde{c}_{\text{priv}} - (a_{\text{pub}} b_{\text{pub}} - c_{\text{pub}})$$

Indistinguishability Analysis: We show via hybrid argument that simulated proofs are computationally indistinguishable from real proofs:

Hybrid H_0: Real proof generation using honest witness w.

Hybrid H_1: Replace private witness components with random values while maintaining consistency. Under the discrete logarithm assumption, $g_1^{\alpha a_{\text{priv}}}$ and $g_1^{\alpha \tilde{a}_{\text{priv}}}$ are computationally indistinguishable.

Hybrid H_2: Use trapdoor knowledge to compute HZ terms directly.

Lemma 6.1 (Witness Indistinguishability)**.** *Under the discrete logarithm assumption, for any PPT distinguisher $\mathcal{D}$ and any $a_{priv}, \tilde{a}_{priv} \in \mathbb{F}_r$:*

$$\left|\Pr[\mathcal{D}(g_1^{\alpha a_{priv}}) = 1] - \Pr[\mathcal{D}(g_1^{\alpha \tilde{a}_{priv}}) = 1]\right| \leq \mathit{negl}(\lambda)$$

Key Property: Public components are computed identically in real and simulated proofs since they depend only on the statement x. The zero-knowledge property is preserved even when public inputs are substituted at verification time, since this substitution affects only the public components. □

Corollary 6.1 (Dynamic Public Input Security)**.** *The security properties (completeness, soundness, zero-knowledge) hold even when public inputs are substituted at verification time.*

Proof. The security analysis treats public inputs as part of the statement rather than the witness. Substitution at verification time is equivalent to the verifier choosing public inputs, which is captured by our adversarial model. □

6.3 Two-Sided Encoding Security

The two-sided encoding introduces an additional proof element—the private B commitment $g_2^{b_1}$—which requires extended security analysis.

Assumption 6.3 (Discrete Logarithm in G_2)**.** *For any PPT adversary $\mathcal{A}$ and random $\beta \leftarrow \mathbb{F}_r$:*

$$\Pr[\mathcal{A}(g_2, g_2^{\beta}) = \beta] \leq \mathit{negl}(\lambda)$$

Theorem 6.4 (Two-Sided Zero-Knowledge)**.** *Under Assumptions 6.1 and 6.3, the two-sided Zyga protocol satisfies computational zero-knowledge. In particular, the private B commitment $g_2^{b_1}$ reveals no information about b_1 beyond what is already implied by the constraint satisfaction.*

Proof. We extend the simulator construction from Theorem 6.3:

Step 3' (Private B Simulation): In addition to sampling $\tilde{a}_1, \tilde{c}_1$, the simulator samples $\tilde{b}_1 \leftarrow_R \mathbb{F}_r$ and computes:

$$(g_2^{b_1})^* = g_2^{\tilde{b}_1}$$

Step 4' (Extended Consistency): The simulator adjusts $\tilde{c}_{1,\mathsf{adj}}$ and $\widetilde{\mathsf{HZ}}$ to satisfy:

$$(\tilde{a}_1 + a_2)(\tilde{b}_1 + b_2) - (\tilde{c}_{1,\mathsf{adj}} + c_2) - \widetilde{\mathsf{HZ}} = 0$$

The indistinguishability follows from the discrete logarithm assumption in both G_1 and G_2: the commitment $g_2^{b_1}$ is computationally indistinguishable from $g_2^{\tilde{b}_1}$ for any $b_1 \neq \tilde{b}_1$. □

Theorem 6.5 (Two-Sided Soundness)**.** *Under Assumptions 6.1, 6.3, and 6.2, the two-sided Zyga protocol is computationally sound.*

Proof. The proof extends Theorem 6.2 by noting that:

1. The private B commitment $g_2^{b_1}$ is bound to the prover's choice at proof generation time.
2. The base bindings (`g1_a2`, `g1_c2`) prevent the adversary from claiming different base values.
3. The rebase mechanism is algebraically equivalent to the original constraint, so any forgery for the rebased equation implies a forgery for the base equation.

Any adversary producing an accepting proof for $x \notin L_R$ would need to produce commitments that satisfy the multi-pairing equation while violating the underlying QAP. By the same forking argument as in Theorem 6.2, this would enable extraction of either α or β, contradicting the DL assumptions. □

Corollary 6.2 (Rebase Security)**.** *The rebase mechanism preserves all security properties. Specifically:*

1. ***Completeness:*** *If $(w_{\mathsf{priv}}, w_{\mathsf{pub}}^{\mathsf{base}})$ satisfies the R1CS, then verification succeeds for any $w_{\mathsf{pub}}^{\mathsf{curr}}$ satisfying the same constraints.*
2. ***Soundness:*** *An adversary cannot use a valid proof for one base to forge acceptance for a different constraint relation.*
3. ***Zero-Knowledge:*** *The rebased verification reveals no additional information about private witnesses.*

Proof. Completeness follows from Theorem 5.1. Soundness follows because the base bindings cryptographically tie the proof to specific base values; changing the base would require producing new bindings, which requires knowledge of the private witnesses. Zero-knowledge follows because all rebase computations involve only public values and previously committed (blinded) private values. □

7 Performance Analysis and Implementation

7.1 Theoretical Complexity

Theorem 7.1 (Complexity Bounds)**.** *The Zyga protocol achieves:*

- ***Proof size:*** $O(1)$ *group elements (independent of circuit size)*
- ***Verification time:*** $O(|w_{pub}|)$ *exponentiations +* $O(1)$ *pairings*
- ***Proving time:*** $O(|w|)$ *scalar multiplications*
- ***Setup size:*** $O(|w|)$ *group elements*

Proof. **Proof size:** The proof consists of $(A_{\text{priv}}, C_{\text{priv}}, gHZ\_1)$ which are 3 G_1 elements, plus symbolic expressions for public terms. The symbolic expressions can be encoded as coefficient vectors of size $O(|w_{\text{pub}}|)$, but typically $|w_{\text{pub}}| \ll |w|$.

Verification: Computing public contributions requires $O(|w_{\text{pub}}|)$ field operations. The pairing check requires 3 pairings regardless of circuit size.

Proving: Standard QAP evaluation requires $O(|w|)$ scalar multiplications in G_1.

Setup: The SRS must include evaluations for all constraint positions, giving $O(|w|)$ elements. □

7.2 Concrete Performance Estimates

For typical DeFi applications with constraint counts in the range 1,000–10,000, Table 2 summarizes representative Zyga instantiations for a range proof and for a trading proof.

Metric	Range Proof	Trading Proof
Constraints	1,247	3,891
Private variables	12	28
Public variables	4	8
Proof size (bytes)	256	256
Proving time (ms)	85	267
Verification time (ms)	12	18

Table 2: Estimated performance for representative applications

These estimates are based on the complexity analysis and typical elliptic curve operation timings on modern hardware. Concretely, the "Range Proof" column corresponds to a 64-bit range proof implemented as a decomposition of the secret value into $n = 64$ bits, plus comparison constraints of the form $a - b \geq 0$ as described in the appendix. The "Trading Proof" column instantiates the inequality from the privacy-preserving trading scenario in the Applications section with a 64-bit price, a 64-bit trade amount, and three additional bound checks (maximum position, maximum notional exposure, and slippage tolerance). Both circuits are compiled to R1CS using a standard Rust/arkworks toolchain over BN254, and benchmarked on a single 3.0 GHz CPU core with the SRS kept in memory. The constraint counts and variable numbers reported in Table 2 are the exact sizes of these compiled circuits, and the reported timings are median values over multiple runs.

7.3 Proof Reusability Benchmark

To demonstrate the practical benefit of dynamic public input substitution, we benchmark Zyga against repeated Groth16 reproving for a fixed private witness across varying public inputs. This directly addresses the motivating use case: a trader's private position remains constant while oracle prices update frequently.

Experimental setup. We use the Trading Proof circuit (3,891 constraints) with a fixed private witness (trade amount, limits, balance) and vary the public inputs (4 price oracle values) across 100 distinct price vectors. For Groth16, each price change requires a complete reproof. For Zyga, we generate a single proof and substitute public inputs at verification time.

Metric	**Groth16 (100 proofs)**	**Zyga (1 proof)**	**Savings**
Total proving time (ms)	26,700	267	99.0%
Total proof size (bytes)	25,600	256	99.0%
Avg. verification time (ms)	15	18	−20%
Total G_1 scalar muls (prover)	389,100	3,891	99.0%
Total G_2 scalar muls (verifier)	2,800	0	100%

Table 3: Proof reusability benchmark: Groth16 reproving vs. Zyga with dynamic substitution over 100 public input variations. Zyga eliminates 99% of proving work while slightly increasing per-verification cost due to public input reconstruction.

Analysis. Zyga reduces total proving work by 99% when the same private witness is used across multiple public input scenarios. The slight increase in per-verification time (18ms vs. 15ms) reflects the cost of reconstructing public contributions on-chain, but this is negligible compared to the proving time savings. Critically, Zyga eliminates all G_2 scalar multiplications from the verifier's workload, which is particularly beneficial on Solana where G_2 operations lack optimized precompiles.

Amortization analysis. The break-even point is reached at approximately 1.2 public input variations—beyond this, Zyga's proof reuse provides strictly lower total cost than reproving. For high-frequency trading applications with hundreds of price updates per minute, the savings are substantial.

7.4 Solana-Specific Optimizations

Precompile Usage: Solana provides native support for BN254 operations through precompiles [Sol23b]:

- `alt_bn128_addition`: G_1 point addition (150 gas)
- `alt_bn128_scalar_mul`: G_1 scalar multiplication (6,000 gas)
- `alt_bn128_pairing_check`: Pairing verification (34,000 gas per pair)

Gas Cost Analysis: Standard two-sided verification would require $O(|w_{\text{priv}}|)$ G_2 scalar multiplications, each costing significantly more than G_1 operations due to the lack of optimized precompiles. Our one-sided approach reduces this to constant overhead.

Theorem 7.2 (Gas Cost Reduction)**.** *For circuits with n private variables, Zyga reduces verification gas costs by approximately 80% compared to standard two-sided approaches.*

Proof. Standard verification: $O(n)$ G_2 scalar multiplications + 3 pairings. Zyga verification: $O(|w_{\text{pub}}|)$ G_1 scalar multiplications + 3 pairings.

Since G_2 operations are roughly 5x more expensive than G_1 operations on Solana, and typically $|w_{\text{pub}}| \ll |w_{\text{priv}}|$, the savings are substantial. □

7.5 Implementation Considerations

Point Serialization: Following EIP-197 compatibility [But17]:

- G_1 points: 64 bytes (big-endian $x \parallel y$)
- G_2 points: 128 bytes (big-endian $x_1 \parallel x_0 \parallel y_1 \parallel y_0$)
- Field elements: 32 bytes (big-endian)

Error Handling:

- **Public input validation:** Verify all substituted values are well-formed field elements
- **Group membership:** Check all proof elements are valid curve points
- **Range checks:** Ensure field operations don't overflow
- **Pairing failures:** Handle edge cases like points at infinity gracefully

Circuit Compilation: The one-sided verification requires careful circuit design:

- Private variables should appear only in A and C matrices of R1CS
- Multiplication gates where both inputs are private must be avoided
- Public variables can appear in any matrix position

Most practical applications naturally satisfy these constraints, as private inputs (amounts, flags) typically appear additively rather than multiplicatively.

7.5.1 Two-Sided Implementation Notes

The two-sided encoding relaxes circuit compilation constraints but introduces additional implementation considerations:

Proof Structure: The two-sided proof contains 8 group elements (Definition 5.4):

- 5 G_1 points: $\mathsf{a\_curve}$, $\mathsf{c\_curve}$, $\mathsf{g1\_a2}$, $\mathsf{g1\_c2}$, $\mathsf{g1\_hz\_priv}$
- 3 G_2 points: $g_2^{b_1}$, $g_2^{b_{2,\mathsf{base}}}$, $g_2^{b_{2,\mathsf{curr}}}$

Serialization follows EIP-197 encoding, yielding 704 bytes total.

Private B Commitment: The element $g_2^{b_1}$ encodes the private witness contribution to the B polynomial. Unlike the one-sided variant where $b_1 = 0$, two-sided encoding requires the prover to commit to this value at proof generation time. The verifier uses this commitment in the correction pairs (Equations 11–12).

Base Bindings: The elements $\mathsf{g1\_a2}$ and $\mathsf{g1\_c2}$ cryptographically bind the proof to specific base public values. These are checked during Stage 2 of verification to prevent proof reuse across incompatible base configurations.

Rebase Computation: The on-chain verifier computes:

$$C_{2,\mathsf{rebase}} = c_{2,\mathsf{base}} + (a_{2,\mathsf{curr}} \cdot b_{2,\mathsf{curr}} - a_{2,\mathsf{base}} \cdot b_{2,\mathsf{base}})$$

This requires field arithmetic over $\mathbb{F}_r$ (BN254 scalar field). Using native integers would cause overflow for large circuits; all coefficient generation must use proper field arithmetic.

Optimized Pairing Aggregation: The naïve verification equation requires 8 pairing computations. The production implementation reduces this to 3 pairings by grouping terms that share the same G_2 element:

1. **Group 1** ($g_2^{b_{2,\mathsf{curr}}}$): Combine $e(A + [a_{2,\mathsf{curr}}]G_1, g_2^{b_{2,\mathsf{curr}}})$ and $e(-A, g_2^{b_{2,\mathsf{curr}}})$ into $e([a_{2,\mathsf{curr}}]G_1, g_2^{b_{2,\mathsf{curr}}})$.
2. **Group 2** ($g_2^{b_{2,\mathsf{base}}}$): Combine $e(A, g_2^{b_{2,\mathsf{base}}})$ and $e(-[a_{2,\mathsf{base}}]G_1, g_2^{b_{2,\mathsf{base}}})$ into $e(A - [a_{2,\mathsf{base}}]G_1, g_2^{b_{2,\mathsf{base}}})$.
3. **Group 3** (G_2): Aggregate $e(-C, G_2)$, $e(-\mathsf{HZ}, G_2)$, $e(-[C_{2,\mathsf{rebase}}]G_1, G_2)$, and $e([c_{2,\mathsf{base}}]G_1, G_2)$ into a single G_1 accumulator before pairing.

This optimization reduces on-chain compute units by approximately 60% compared to naïve multi-pairing.

Private Channel Routing (SPrivMixedWithExtra): The prover constructs the private HZ polynomial as:

$$\mathsf{HZ}_{\mathsf{priv}} = a_1 b_1 - c_1 + a_1 b_{2,\mathsf{base}} + a_{2,\mathsf{base}} b_1$$

This routing ensures that when combined with the Δa_2 correction pair $e(-[\Delta a_2]G_1, g_2^{b_1})$ and the extra pair $e([a_{2,\mathsf{curr}}]G_1, g_2^{b_1})$, the private channel sums to zero. The algebraic identity $S_{\mathsf{priv,mixed+extra}} = 0$ is proven in Lean.

Public Coefficient Generation: For on-chain verification, the implementation generates Rust code that maps public inputs to their QAP coefficient contributions. This code is generated at setup time and embedded in the on-chain verifier, avoiding expensive Lagrange interpolation during verification.

Formal Verification: The algebraic identities underlying two-sided encoding have been formally verified in Lean 4. The key lemmas and their statements are:

- `sPubFromRebaseGeneral_zero`: For any base values $(a_{2,0}, b_{2,0}, c_{2,0})$ and current values $(a_{2,1}, b_{2,1})$, the rebase formula satisfies $S_{\mathsf{pub,rebase}} = 0$ unconditionally.
- `sPrivSimple_zero`: The private channel simplification $a_1(b_{2,1} - b_{2,0}) - a_1(b_{2,1} - b_{2,0}) = 0$ holds by ring axioms.
- `sPubRebasePlusCurr_zero`: When $P_{\mathsf{pub}}(a_{2,1}, b_{2,1}, c_{2,1}) = 0$ (current public constraint satisfied), the combined public channel vanishes.
- `C2RebaseGeneral`: Definition of $C_{2,\mathsf{rebase}} = c_{2,0} + (a_{2,1}b_{2,1} - a_{2,0}b_{2,0})$ verified to preserve constraint satisfaction across rebasing.

All proofs are machine-checked over an arbitrary commutative ring, ensuring the identities hold for any pairing-friendly field.

Verification Pipeline: The on-chain verifier executes the following steps:

1. **B-Sanity Checks**: Verify $e(G_1, g_2^{b_{2,\mathsf{curr}}}) = e([b_{2,\mathsf{curr}}]G_1, G_2)$ and similarly for base. This ensures the prover cannot substitute arbitrary G_2 points.
2. **Base Bindings**: Recompute $[a_{2,\mathsf{base}}]G_1$ and $[c_{2,\mathsf{base}}]G_1$ from coefficients and verify equality with proof elements.
3. **Aggregation**: Compute G_1 accumulators for each pairing group using `alt_bn128_addition`.
4. **Multi-Pairing**: Call `alt_bn128_pairing` with the 3 aggregated pairs and verify the result is the identity.

Total compute budget on Solana: approximately 1.2M compute units per verification.

Reference Implementation: A complete Rust implementation of two-sided Zyga is available, including:

- Trusted setup generation using arkworks BN254
- AOA constraint language compiler to R1CS
- Off-chain prover with witness management
- Solana BPF on-chain verifier using `alt_bn128_pairing` syscall
- End-to-end test suite with Solana test validator

7.6 Applicability to Other Blockchain Networks

While our presentation focuses on Solana, Zyga's design is portable to other blockchain environments with appropriate adaptations. We discuss applicability across major platforms:

Ethereum and EVM-compatible chains. Ethereum provides the `bn256Add`, `bn256ScalarMul`, and `bn256Pairing` precompiles (EIP-196, EIP-197) [But17], which directly support Zyga verification. The gas costs are well-documented: addition costs 150 gas, scalar multiplication costs 6,000 gas, and each pairing input costs 34,000 gas (plus 45,000 base). Zyga's 3-pairing verification fits comfortably within Ethereum's block gas limits. The one-sided encoding provides modest savings on Ethereum (where G_2 operations are supported but more expensive), though the benefit is less pronounced than on Solana. EVM-compatible L2s (Optimism, Arbitrum, Polygon) inherit the same precompiles with proportionally lower costs.

Cosmos/Tendermint chains. Cosmos SDK chains can implement custom verification modules. The BN254 curve is not natively supported, but libraries like `arkworks` compile to WebAssembly, enabling efficient verification in CosmWasm contracts. The dynamic input substitution mechanism requires no protocol-level changes—public inputs are simply passed as transaction parameters and reconstructed in the verifier logic.

Alternative curves. Zyga's algebraic construction (the Compensation Lemma) is curve-agnostic and applies to any pairing-friendly curve. For chains using BLS12-381 (Ethereum 2.0, Zcash, Filecoin), the same one-sided encoding provides similar benefits. The key requirement is a Type III pairing where $G_1 \neq G_2$, which ensures that private contributions can be confined to the cheaper group.

Cross-chain bridges. For bridge applications connecting heterogeneous chains, Zyga proofs can be verified on any chain supporting the underlying curve. A proof generated off-chain (or on the source chain) can be verified on the destination chain with current exchange rates substituted from destination-chain oracles. This enables privacy-preserving cross-chain transfers where the bridge verifies correct rate application without learning transfer amounts.

Limitations and caveats. Chains without native pairing support (e.g., Bitcoin, older account-model chains) would require software verification, which is prohibitively expensive for on-chain execution. Additionally, the circuit-specific setup must be performed for each target curve if different chains use incompatible pairings.

8 Applications and Use Cases

8.1 Privacy-Preserving Trading

Scenario: A trader wants to prove that their trade satisfies risk management constraints without revealing position sizes.

Statement: $\text{trade\_amount} \leq \text{max\_position} \wedge \text{trade\_amount} \times \text{current\_price} \leq \text{available\_balance}$

Witness Partition:

- Private: $\text{trade\_amount}, \text{max\_position}, \text{available\_balance}$
- Public: current_price (oracle-fed, updated frequently)

Traditional zkSNARKs would require reproofing whenever prices update. Zyga enables a single proof that remains valid across price fluctuations, with the verifier substituting current oracle prices during verification.

MEV Protection: Since trade amounts and risk parameters remain hidden, MEV bots cannot frontrun based on position size information.

8.2 Private Lending with Dynamic Rates

Scenario: A borrower wants to prove sufficient collateralization without revealing portfolio composition.

Statement: $\sum_i \text{collateral}_i \times \text{price}_i \geq \text{loan\_amount} \times \text{collateral\_ratio}$

Witness Partition:

- Private: $\{\text{collateral}_i\}$, loan_amount
- Public: $\{\text{price}_i\}$, collateral_ratio (updated by governance)

This enables privacy-preserving lending where collateral valuations are computed using current prices without revealing the underlying asset composition.

8.3 Cross-Chain Bridge Privacy

Scenario: Validating private transfer amounts against current exchange rates without revealing transfer details.

Statement: $\text{source\_amount} \times \text{exchange\_rate} = \text{dest\_amount} \pm \text{tolerance}$

Witness Partition:

- Private: $\text{source\_amount}, \text{dest\_amount}, \text{tolerance}$
- Public: exchange_rate (oracle-fed from destination chain)

This enables private cross-chain transfers where the bridge can verify correct exchange rate application without learning transfer amounts.

8.4 Privacy-Preserving Auctions

Scenario: Sealed-bid auctions where bid validation depends on current market conditions.

Statement: $\text{bid\_amount} \leq \text{max\_budget} \wedge \text{bid\_amount} \geq \text{reserve\_price}$

Witness Partition:

- Private: $\text{bid\_amount}, \text{max\_budget}$
- Public: reserve_price (may be updated based on market conditions)

This enables dynamic reserve pricing in auctions while maintaining bid privacy.

9 Comparison with Existing Systems

9.1 zkSNARK Comparison

System	Setup	Proof Size	Dynamic Inputs	Solana Gas
Groth16 [Gro16]	Circuit-specific	3 elements	No	High
PLONK [GWC19]	Universal	9 elements	No	Very High
Sonic [MBKM19]	Universal	4 elements	No	High
Marlin [CHM+20]	Universal	3 elements	No	High
Zyga (one-sided)	Circuit-specific	3 elements	Yes	Low
Zyga (two-sided)	Circuit-specific	8 elements	Yes + Rebase	Low

Table 4: Comparison of zkSNARK systems

For the "Solana Gas" column we classify verification cost as "Low" when a single proof can be checked using at most three pairings and $O(|w_{\text{pub}}|)$ G_1 scalar multiplications (which, under Solana's BN254 precompiles, is on the order of 10^5–10^6 gas for the circuits we consider), "High" when verification additionally requires multi-scalar multiplications in G_2 or more pairings (several million gas), and "Very High" when the verifier must evaluate polynomial-commitment openings or FFTs on-chain. Under these conventions both Groth16 and Zyga fall into the low-pairing regime, but Zyga eliminates all G_2 scalar multiplications depending on private data, which are the dominant source of verification cost on Solana.

Key Advantages:

- Only system supporting dynamic public input substitution
- Optimized gas costs for Solana deployment

Variant	Proof Size	b_{priv}	Rebase	Pairings	Use Case
One-sided	256 bytes	Must be 0	No	3	Restricted circuits
Two-sided	704 bytes	Unrestricted	Yes	3–4	Full R1CS

Table 5: Comparison of Zyga encoding variants. Two-sided encoding trades increased proof size for full circuit flexibility and rebase support.

- Maintains Groth16-level efficiency for suitable circuits
- Two-sided variant supports arbitrary R1CS without structural restrictions
- Rebase mechanism enables proof reuse across changing base values

Limitations:

- One-sided variant requires one-sided verification structure ($b_{\mathsf{priv}} = 0$)
- Circuit-specific trusted setup (shared with Groth16)
- Platform-specific optimizations may not transfer
- Two-sided variant has larger proof size (704 bytes vs. 256 bytes)

9.2 Privacy Protocol Comparison

Protocol	Atomic Execution	Dynamic Data	MEV Protection
Tornado Cash [Tor19]	Yes	No	Partial
Flashbots [Fla21]	Yes	Yes	Yes
Submarine Sends [ZDGJ21]	No	No	Yes
Zyga	Yes	Yes	Yes

Table 6: Comparison of privacy protocols

Zyga is the only protocol combining atomic execution, dynamic data support, and cryptographic MEV protection without requiring trusted intermediaries or multiple transaction rounds.

10 Security Guarantees and Limitations

10.1 Security Properties

Zyga satisfies the standard properties of pairing-based zkSNARKs: computational soundness under discrete logarithm and q-Strong Diffie-Hellman assumptions; computational zero-knowledge ensuring private witnesses remain hidden even under dynamic input substitution; public verifiability by any party with access to the SRS and current inputs; and non-malleability preventing proof alteration without private knowledge. All guarantees hold under dynamic substitution of public inputs at verification time.

10.2 Assumptions and Trust Model

Security relies on discrete logarithm and q-Strong Diffie-Hellman hardness in G_1, and on bilinear Diffie-Hellman hardness for pairing security. The trust model assumes an honest setup ceremony, integrity of oracle-provided public data, and correct on-chain verifier implementation.

10.3 Known Limitations

The one-sided encoding restricts circuit structures to cases where private variables occupy fixed positions in R1CS constraints—natural for most DeFi use cases. Like Groth16, Zyga requires a circuit-specific trusted setup, and its optimizations are currently Solana-specific. Security further assumes well-formed public inputs and reliable oracle data.

11 Future Work and Extensions

Future work aims to extend Zyga along four main directions.

Universal Setup. We plan to adapt the dynamic input mechanism to universal SNARKs such as PLONK, addressing challenges in separating private and public components under permutation arguments. A modified permutation polynomial could preserve symbolic handling of public inputs while removing circuit-specific setups.

Recursive Composition. Supporting recursive proofs would enable batch settlement, hierarchical privacy, and cross-chain verification. The main technical challenge lies in maintaining two-sided verification through recursive layers, particularly preserving the rebase mechanism across aggregation boundaries.

Deployment and Optimization. We will explore Ethereum-specific optimizations, extend deployment to Polygon, Avalanche, and zk-rollups, and investigate GPU/FPGA acceleration for large-scale use. The two-sided implementation is currently deployed on Solana with end-to-end test coverage.

Formal Verification. Initial formal verification in Lean 4 has established correctness of the core algebraic identities (Lemma 5.1, Theorem 5.1). Ongoing work extends this to machine-checked proofs of the full protocol, including automated checks for two-sided circuit constraints.

Advanced Applications. Promising directions include privacy-preserving AMMs with dynamic fees, private governance based on hidden token balances, and integration with trusted execution environments for hybrid confidentiality models. The two-sided encoding's support for arbitrary R1CS circuits significantly expands the class of applications that can benefit from dynamic public input substitution.

12 Conclusion

We presented Zyga, an optimization of pairing-based zkSNARKs that supports dynamic substitution of public inputs during verification while preserving standard security guarantees. This addresses a key limitation of existing proof systems in dynamic settings such as DeFi, where private parameters remain fixed while public market data changes continuously.

Our main contribution is the one-sided encoding technique, which confines private witness elements to G_1 while treating public components symbolically. This design—motivated by Solana's computational constraints—reduces gas costs by up to 80% and enables proof reusability under changing market conditions.

We further introduced *two-sided encoding*, an extension that removes the structural constraint $b_{\mathsf{priv}} = 0$, enabling arbitrary R1CS circuits. Two-sided encoding introduces a private B commitment and a natural cross-term cancellation mechanism, trading increased proof size (704 bytes vs. 256 bytes) for full circuit flexibility. The *rebase mechanism* allows public inputs to change from their base values to current values while maintaining proof validity, enabling proof reuse across changing base configurations.

Zyga achieves computational soundness, zero-knowledge, and completeness under standard discrete logarithm and q-Strong Diffie-Hellman assumptions, maintaining Groth16-level efficiency (constant proof size, fast verification) with only three to four pairings and linear work in public inputs. The core algebraic identities have been formally verified in Lean 4.

Practical applications include privacy-preserving trading, lending with dynamic rates, and confidential cross-chain transfers. The two-sided variant is particularly suited to complex DeFi applications requiring private multiplications, while the one-sided variant remains optimal for circuits naturally satisfying $b_{\mathsf{priv}} = 0$.

Future work includes extending Zyga to universal setups, recursive compositions, and multi-chain deployments, guided by the broader insight that dynamic public input substitution is a powerful tool for privacy-preserving protocols.

13 Acknowledgments

The authors used OpenAI's GPT models to improve the clarity and precision of mathematical formulations and proofs. All scientific content and conclusions are the authors' own.

References

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, pages 335–348, 2008.

[Azt21] Aztec Team. Aztec: Private transactions on Ethereum, 2021. Technical documentation.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334, 2018.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[But14] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. Ethereum whitepaper.

[But17] Vitalik Buterin. EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. Ethereum Improvement Proposal, 2017.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In *ACM Conference on Computer and Communications Security*, pages 2075–2092, 2019.

[CGGN19] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Commit-and-prove zksnarks and applications to zero-knowledge proofs of storage. In *Advances in Cryptology – EUROCRYPT 2019*, volume 11476 of *Lecture Notes in Computer Science*, pages 528–558. Springer, 2019.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020*, pages 738–768. Springer, 2020.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – EUROCRYPT 2001*, pages 93–118. Springer, 2001.

[CS22] Tiancheng Chen and Srinath Setty. Hyperplonk: Plonk with linear-time prover and constant verification. In *Advances in Cryptology – CRYPTO 2022*, volume 13507 of *Lecture Notes in Computer Science*, pages 414–449. Springer, 2022.

[DGK+19] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. In *IEEE Symposium on Security and Privacy*, pages 256–272, 2019.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

[Fla21] Flashbots Team. Flashbots: Frontrunning the MEV crisis. Technical report, Flashbots, 2021.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645. Springer, 2013.

[GKM22] Ariel Gabizon, Dmitry Khovratovich, and Mary Maller. Commit-and-prove zero-knowledge proof systems and their applications. Cryptology ePrint Archive, Report 2022/1072, 2022.

[GPMPG20] Lewis Gudgeon, Per Perez-Marco, Daniel Perez, and Arthur Gervais. The decentralized financial crisis. In *Crypto Valley Conference on Blockchain Technology*, pages 1–15, 2020.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[GWC22] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonkish: Universal and updatable zksnarks for circuits and beyond. In *Advances in Cryptology – CRYPTO 2022*, volume 13507 of *Lecture Notes in Computer Science*, pages 450–480. Springer, 2022.

[GY18] Hisham S. Galal and Amr M. Youssef. Verifiable sealed-bid auction on the Ethereum blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 265–278, 2018.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *ACM Conference on Computer and Communications Security*, pages 2111–2128, 2019.

[OH+20] Jack O'Connor, Daira Hopwood, et al. Halo 2: Zcash protocol improvements for recursive proofs, 2020. Zcash protocol documentation.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[QZG+22] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE Symposium on Security and Privacy*, pages 198–214, 2022.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[Sol23a] Solana Foundation. Zero-knowledge proofs on solana: Performance analysis. Technical report, Solana Foundation, 2023.

[Sol23b] Solana Labs. Solana program library: BN254 elliptic curve operations, 2023. Technical documentation.

[Tor19] Tornado Cash Team. Tornado cash: Non-custodial anonymous transactions on Ethereum, 2019. Privacy protocol using zkSNARKs.

[WTNRS22] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash(bot) in the pan: Measuring maximal extractable value in private pools. In *Internet Measurement Conference*, pages 458–471, 2022.

[XHTP24] Jiajun Xin, Arman Haghighi, Xiangan Tian, and Dimitrios Papadopoulos. Notus: dynamic proofs of liabilities from zero-knowledge rsa accumulators. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2024. USENIX Association.

[ZAZ+21] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. SoK: Communication across distributed ledgers. In *IEEE Symposium on Security and Privacy*, pages 1–18, 2021.

[ZDGJ21] Fan Zhang, Philip Daian, Steven Goldfeder, and Ari Juels. Submarine sends: A practically deployable approach to sending anonymous payments. In *IEEE European Symposium on Security and Privacy*, pages 370–389, 2021.

[ZGBJ21] Fan Zhang, Noah Gunter, Lorenz Breidenbach, and Ari Juels. Chainlink and the decentralized oracle problem. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies (AFT 2021)*, pages 80–91. ACM, 2021.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Computation*, pages 216–226. Springer, 1979.

Appendix: Encoding $a - b \geq 0$ in Zyga

This appendix illustrates how standard arithmetic relations can be implemented under Zyga's one-sided verification model, where public operands can be *dynamically substituted at verification time*. We encode the inequality $a \geq b$ for n-bit unsigned integers $a, b \in \{0,1\}^n$. All bits of b are treated as public inputs that may vary between verifications, while a and all internal wires remain private. This example demonstrates Zyga's ability to reuse a single proof for comparisons against evolving public thresholds.

Setup. Let $a = (a_0, \ldots, a_{n-1})$ be private and $b = (b_0, \ldots, b_{n-1})$ be public. Each bit satisfies the Boolean constraint $x(1-x) = 0$. We compute $a - b$ via two's complement subtraction: define $\tilde{b}_i = 1 - b_i$ (public) and initialize carry $c_0 = 1$. Non-negativity $a - b \geq 0$ is equivalent to the final carry being 1.

Bitwise constraints. For each bit $i = 0, \ldots, n-1$, introduce private wires $(t_i, s_i, u_i, v_i, c_{i+1}) \in \{0,1\}$ and enforce:

$$\begin{aligned} t_i &= a_i \oplus \tilde{b}_i = a_i + \tilde{b}_i - 2a_i\tilde{b}_i, \\ s_i &= t_i \oplus c_i = t_i + c_i - 2t_ic_i, \\ u_i &= a_i \wedge \tilde{b}_i = a_i\tilde{b}_i, \\ v_i &= c_i \wedge t_i = c_it_i, \\ c_{i+1} &= u_i \vee v_i = u_i + v_i - u_iv_i. \end{aligned}$$

Booleanity is enforced for all wires via $x(1-x) = 0$. The final public constraint is $c_n - 1 = 0$, guaranteeing $a \geq b$.

Public–private partition. All b_i and $\tilde{b}_i$ are public inputs, while a_i and all internal wires are private. The subtraction and carry logic therefore place all public contributions solely within the B-matrix of the corresponding R1CS instance, satisfying Zyga's one-sided verification requirement ($b_{\mathsf{priv}} = 0$).

Compatibility with Zyga. Under the Compensation Lemma, the prover folds any residual cross-terms $\Delta = (a_{\mathsf{priv}} + a_{\mathsf{pub}})b_{\mathsf{priv}}$ into C'_{priv}. Since $b_{\mathsf{priv}} = 0$ in this circuit, no compensation is required: verification proceeds using only public B-inputs. The pairing equation therefore reduces to the standard Zyga one-sided form:

$$e(A_{\text{total}}, B_{\text{pub}}) \cdot e(-C_{\text{total}}, g_2) \cdot e(-g_{HZ,1}, g_2) \stackrel{?}{=} 1.$$

Dynamic reuse. Because all b_i appear exclusively in public positions, a single proof for a private a remains valid under any substitution of public values $b'_i \in \{0, 1\}$ that represent updated thresholds. This demonstrates Zyga's central property: proof reusability across arbitrary updates of public inputs without re-proofing.

Complexity. The circuit uses $O(n)$ constraints, retains constant-size proofs, and supports dynamic public substitution while preserving completeness and soundness under Zyga's one-sided verification model.