Bitcoin PIPEs v2

Covenants and ZKPs on Bitcoin via Witness Encryption

Michel Abdalla Brent Carmer Muhammed El Gebali Handan Kilinc-Alper
Mikhail Komarov Yaroslav Rebenko Lev Soukhanov Erkan Tairi Elena Tatuzova
Patrick Towa

[[alloc] init]

February 5, 2026

Abstract

Covenants and ZKP verification directly on Bitcoin L1 have long been regarded as infeasible due to the limited expressiveness of Bitcoin Script and the absence of covenant-enabling opcodes such as `OP_CAT`*,* `OP_CTV`*,* `OP_VAULT` *or* `OP_CSFS`*. These limitations have prevented the realization of zkRollups, trustless bridges, and programmable vaults natively on Bitcoin.*

This work introduces Bitcoin PIPEs *v2, an upgrade to the original Bitcoin PIPEs approach focusing on emulating missing covenant functionality practically without requiring a soft fork. At its core, a PIPE v2 uses a* witness encryption (WE) *scheme to lock a Bitcoin private key under an NP statement. The key (and thus the ability to spend the associated coins) can be recovered only by a participant who provides a valid witness (e.g., a SNARK proof) satisfying that statement. Once unlocked, the mechanism outputs a standard Schnorr signature indistinguishable from any other Bitcoin signature. From Bitcoin's perspective, transactions appear entirely ordinary; yet they are cryptographically guaranteed to enforce arbitrary off-chain logic.*

We formalize how PIPEs v2 enable arbitrary spending conditions on Bitcoin by enforcing predicates on signatures through cryptography, without requiring any consensus changes. We introduce a new primitive, the ***Witness Signature (WS)****, which captures conditional signing under hard relations. We show that a PIPE instantiated with a WE scheme and a standard digital signature scheme enables programmable covenants and SNARK-verifiable conditions on Bitcoin—entirely without soft forks, trusted parties, or interactive fraud-proof mechanisms such as those used in BitVM constructions.*

Finally, we explore ***Arithmetic Affine Determinant Program*** *(AADP)-based witness encryption as a concrete and promising research direction for realizing PIPEs. AADPs provide an explicit arithmetic framework for enforcing SNARK-verifiable NP predicates within the PIPE architecture.*

This work presents a new, second-generation construction of PIPEs (PIPEs v2) for Bitcoin, extending and replacing the earlier formulation proposed in [Kom24].

1 Introduction

Bitcoin Script lacks native covenant mechanisms, thereby preventing users from expressing spending conditions that depend on future transaction structure or external cryptographic proofs. Existing opcode introduction proposals such as `OP_CAT`, `OP_CTV`, and `OP_CSFS` would enable covenants, but all require protocol changes that have not been adopted. As a consequence, core functionalities such as programmable vaults, template-restricted transactions and trustless bridges along with add-ons such as zkRollups cannot be realized directly on Bitcoin.

This work presents an alternative cryptographic approach to covenants that operates entirely within Bitcoin's existing consensus rules. We introduce PIPEs (PIPE v2) [1], a mechanism that enforces arbitrary NP statements as spending conditions by shifting covenant logic from Bitcoin Script into cryptography.

[1] While the name originally stood for *Polynomial Inner Product Encryption* to reflect an earlier functional-encryption-based construction [Kom24], it is now retained simply as *PIPE* in recognition of that original version rather than as an acronym.

Intuitively, instead of asking Bitcoin Script to verify whether a spending transaction satisfies some rule, a PIPE makes it *cryptographically impossible* to produce a valid signature unless the rule holds. The private key controlling the locked coins is encrypted inside a mechanism that can "release" it only when a specified computation—such as verifying a SNARK proof or checking a transaction template—succeeds. Once the condition is met, the mechanism recovers the key and outputs an ordinary Schnorr signature indistinguishable from any other Bitcoin signature. From the network's perspective, the transaction appears completely normal; yet behind the scenes, the signature could only have been generated because the specified condition was satisfied. In this way, PIPEs emulate covenant-like, programmable spending policies purely through cryptography, without requiring any changes to Bitcoin's consensus rules.

We design PIPEs by deploying a *Witness Encryption (WE)* [GGSW13] scheme, a cryptographic primitive that allows information to be encrypted in such a way that it can be decrypted only by someone who knows a valid *witness* to a statement being true. In cryptographic terms, a statement x represents a computational claim for example, "this transaction satisfies a specific condition" and a witness w is concrete data proving that claim, such as a SNARK proof, a valid preimage, or a signed message satisfying the rule. In the context of Bitcoin, this means that the private key controlling a UTXO can be encrypted under a statement x describing the intended spending condition, such as "this transaction includes a valid proof" or "the outputs of this transaction match a precommitted template." Only a participant who can produce a corresponding witness w such that $(x, w) \in R$ can decrypt the witness encryption and obtain the signing key. In this way, PIPEs provide a general and composable method for enforcing arbitrary spend rules.

We additionally formalize the security model of *PIPEs* and introduce a new cryptographic primitive that we call the ***Witness Signature*** (WS). This primitive captures the essence of conditional signing under hard relations, providing a general framework for enforcing arbitrary spending conditions through cryptography. We define the syntax and security properties of witness signatures—such as correctness, unforgeability, verifiability, and soundness—and show that our PIPE construction instantiated with a secure witness encryption scheme and a standard digital signature scheme is a secure witness signature. This formalization not only clarifies the theoretical foundations of PIPEs but also establishes confidence that PIPEs can be used securely for our intended purposes on Bitcoin.

A natural question is how to construct a *feasible* WE scheme suitable for the PIPE setting. In this work, we focus on a construction based on *Arithmetic Affine Determinant Programs* (AADPs) as a concrete and illustrative research direction. Our new AADP-based WE design [aad26] follows the general affine determinant program framework [BIJ+20] for WE, while operating natively over arithmetic constraint systems, making it particularly well suited for enforcing SNARK-verifiable predicates. This approach provides an explicit algebraic mechanism for tying the encryption of a Bitcoin signing key to satisfaction of an NP statement, while keeping all verification logic off-chain.

1.1 Differences between PIPE v2 and PIPE v1 [Kom24].

In this section, we summarize the conceptual and functional differences between the PIPE v1 [Kom24] and the PIPE v2. Although both approaches aim to enforce cryptographically defined spending rules on Bitcoin, their underlying mechanisms yield fundamentally different covenant capabilities.

Distinction in terms of cryptography. In the earlier version [Kom24] of this paper (PIPE v1), we investigated several cryptographic candidates to instantiate witness signatures, including general-purpose *Functional Encryption (FE)*, *Quadratic Functional Encryption (QFE)*, and even *Indistinguishability Obfuscation (iO)*. The original idea was that by encrypting the Bitcoin signing key under an FE or QFE scheme, one could directly compute a signature as part of the decryption process, thereby enforcing covenant conditions through functional evaluation. However, this approach proved infeasible in practice: expressing elliptic-curve operations required for Schnorr signatures as low-degree polynomials exceeded the computational depth supported by QFE, and general-purpose FE schemes that could handle such operations require iO or graded encodings, which are currently far from practical. These primitives also entail large ciphertexts, complex trusted setups, and inefficient decryption procedures, making them unsuitable for any real-world deployment on Bitcoin. For these reasons, we abandoned the FE- and iO-based instantiations in favor of a simpler, yet powerful, WE-based approach. A WE-based PIPE (PIPE v2) does not attempt to compute the signature algorithm directly; instead, it cryptographically binds the release of a private key to the satisfaction of an NP statement as we discussed above. This shift preserves

the core vision of PIPEs, i.e., cryptographically enforced covenants on Bitcoin, while grounding it in assumptions and constructions that are more feasible, modular, and directly compatible with existing Bitcoin primitives.

Distinction in terms of covenant capabilities. PIPE v1 [Kom24] and the WE-based construction in PIPE v2 differ not only in their underlying primitives, but also in how they constrain signing. In the FE-based design in PIPE v1, the functional key implements a map of the form

$$F_{P,\mathsf{sk}}(m_1 \| m_2) = \begin{cases} \mathsf{Sign}_{\mathsf{sk}}(m_1) & \text{if } P(m_2) = 1, \\ \bot & \text{otherwise,} \end{cases}$$

where $P(m) = c(f'(m))$ is a fixed policy. The holder of the functional key never learns sk directly and can obtain signatures only by supplying inputs (m_1, m_2) such that $P(m_2) = 1$. This creates a *per-signature policy gate*: each time a signature is requested, the FE mechanism checks the policy again before producing it. The enforcement occurs *inside* the signing algorithm itself. This gives rise to *message-binding* behavior in the cryptographic sense: signatures are only produced on messages that appear as inputs to a policy-satisfying invocation of $F_{P,\mathsf{sk}}$, even if the signed component (m_1) and the checked component (m_2) are not identical. In particular, FE-based PIPEscan enforce spending rules at every signature request, rather than only once.

In the WE-based PIPE considered this version, the situation is fundamentally different. Witness encryption is used to protect the signing key sk under an NP statement x, and a valid witness w with $(x, w) \in R$ allows sk to be recovered exactly once. After decryption, the user holds an ordinary secret key and can generate Schnorr signatures on *arbitrary* messages without any further interaction with the PIPE mechanism. Consequently, WE-based PIPEs do not provide per-signature enforcement as in the FE setting; instead, they realize a *one-time, witness-gated key release*. This matches many covenant-like constructions on Bitcoin where spendability depends on proving a single, possibly complex, condition (e.g., validity proofs, exit proofs, fraud proofs, or template-consistency proofs) before the funds are handed over to a normal key holder. However, WE-based PIPEs do not emulate covenant mechanisms requiring persistent, transaction-by-transaction policy checks or opcode-level transaction binding (such as `OP_CTV`), because once sk is released the signer is free to construct any transaction.

Definition 1.1 (Binary NP Covenant)**.** *Let R be an NP relation with associated language $L_R = \{x \mid \exists w : (x, w) \in R\}$. A* binary NP covenant *is a spending condition whose authorization rule is specified by a predicate*

$$P_R(x, w) = \begin{cases} 1 & \textit{if } (x, w) \in R, \\ 0 & \textit{otherwise,} \end{cases}$$

that is, a single-bit decision procedure for membership in L_R. A transaction spending from such a covenant is permitted if and only if the spender provides a witness w certifying $P_R(x, w) = 1$. No additional constraints are imposed on the transaction message beyond this one-bit authorization decision.

PIPE v2 with WE naturally enforces precisely these binary NP covenants. The statement x is hard-coded into the PIPE instance, and the signing key is recoverable only when the spender supplies a witness w with $(x, w) \in R$. Thus, PIPE v2 enforces a *single* authorization bit—unlock versus no-unlock. Once unlocked, the signer obtains an ordinary secret key and may sign arbitrary messages; hence multi-step or message-binding covenant semantics are not achieved. Nevertheless, any spending rule reducible to a binary NP predicate fits directly into this definition. Examples include:

- spending gated by a zk-proof,
- witness-verified predicates about the intended spending transaction[2],
- fraud proofs (See § 6) or exit proofs,
- validity proofs for state-transition systems.

[2]Unlike consensus-level mechanisms such as `OP_CTV`, PIPE v2 does not enforce that the on-chain transaction matches a specific template; it only enforces that the spender shows knowledge of a witness attesting that a desired off-chain predicate holds.

1.2 Outline of the Paper

§ 1.3 reviews prior work on Bitcoin covenants, programmable contracts, and witness encryption. § 1.4 gives an intuitive overview of the PIPE framework and its on-chain workflow. In § 3, we formalize the notion of Witness Signatures, define their security properties, and show in § 4 how our PIPE construction realizes a WS scheme when instantiated with a witness encryption scheme. § 5 introduces our AADP-based WE construction. Finally, § 6 shows how PIPE can be deployed within the BitVM framework.

1.3 Related Work

Covenants and Opcode Proposals. Several proposals aim to extend Bitcoin Script with *covenants*, which restrict how coins can be spent in the future. Here are some notable examples:

- **CheckTemplateVerify (CTV).** `OP_CTV`, proposed in 2020 and specified in BIP-119, allows a user to commit to a specific transaction by enforcing that only a transaction whose hash matches a predefined template can spend the output. This enables vaults, congestion control, and other advanced use cases, but activating `OP_CTV` would require a soft fork.
- **Concatenation (CAT).** `OP_CAT`, proposed in BIP-347, concatenates two elements on the stack and can be used to express more complex spending conditions. The absence of concatenation-induced covenants (via `OP_CAT` opcode)(first introduced by Andrew Poelstra in [Poe21]) in Bitcoin has made certain operations cumbersome or impossible. Having CAT is essential for efficient Merkle tree operations for verifying commitments. Without CAT, simulating its functionality requires cumbersome workarounds that are often impractical. Re-enabling `OP_CAT` would make it possible to build efficient Merkle-tree and commitment-based covenants directly in Script. However, like `OP_CTV`, it also requires a protocol change.
- **Vault Opcode (`OP_VAULT`).** `OP_VAULT`, specified in BIP-345 is a covenant-style opcode designed to enforce staged spending paths for UTXOs. In a typical vault construction, funds are first moved into a restricted intermediate state, from which they can only be spent after a delay or via a predefined recovery path. This enables theft-resistant custody, delayed withdrawals, and recovery mechanisms without requiring continuous online monitoring. While vault-like behavior can be approximated using existing Script features and timelocks, these constructions are often complex and fragile. A dedicated `OP_VAULT` opcode would allow vault semantics to be expressed directly and efficiently, but its deployment would require a soft fork and careful consideration of covenant expressiveness.
- **CheckSigFromStack (`OP_CSFS`).** `OP_CSFS`, also known as `OP_CHECKSIGFROMSTACK` and specified in BIP-348, allows signature verification over arbitrary data provided on the stack, rather than over a fixed transaction digest. This enables powerful forms of introspection and authorization, including script-level verification of commitments, adaptor-style constructions, and advanced covenant designs. In particular, `OP_CSFS` makes it possible to verify signatures on application-defined messages inside Script, which is difficult or inefficient with existing opcodes. However, this increased expressiveness raises concerns about complexity and potential misuse, and like previous ones, introducing `OP_CSFS` would require a consensus change via a soft fork.

In summary, these covenant proposals along with others not listed here show the community's interest in more expressive on-chain spending rules. Our work takes a different approach by achieving similar functionality through pure cryptography without modifying Bitcoin's consensus rules.

Programmable Bitcoin. Recent work on enhancing Bitcoin's scripting capabilities has revived interest in disabled opcodes like `OP_CAT` or `OP_MUL` as key building blocks for advanced cryptographic protocols. `OP_CAT`'s ability to concatenate arbitrary data elements makes it a natural candidate for constructing sophisticated commitment schemes that underpin zero-knowledge protocols.

Early studies in Bitcoin privacy, such as Zerocash and Zerocoin, demonstrated the power of ZK proofs in achieving confidential transactions, though these systems generally operate off-chain or require modifications to consensus rules [BCG+14, MGGR13]. In contrast, proposals leveraging `OP_CAT` aim to integrate ZK proofs directly into Bitcoin's scripting environment. By re-enabling `OP_CAT`, researchers have explored how to build covenants that enforce spending conditions while concealing sensitive transaction data through on-chain ZK verification.

The Lightning Network demonstrated that off-chain protocols could securely and efficiently settle transactions on Bitcoin while limiting on-chain footprint [JP16]. The Lightning Network promises instantaneous Bitcoin transactions but requires substantial capital lockup in order for its channels to operate effectively along with a lot of interactivity from users (e.g. requiring for a user to be online to receive payments). While Lightning focuses on payment channels with limited programmability, its design principles inspire BitVM's approach to off-chain contract execution.

BitVM [AAL+24] introduced an optimistic computation model for Bitcoin, enabling users to execute programmable logic enforced by deposits. Its design relies on garbled circuits, where one participant (the Prover) encodes a computation as a Boolean circuit and another (the Verifier) checks its correctness through interactive challenges. If the Prover produces an incorrect result, the Verifier can reveal a minimal subset of the circuit — specifically, the gate where the computation diverges — to construct an on-chain fraud proof. This mechanism allows arbitrary computation to be conditionally enforced on Bitcoin without modifying consensus rules. However, the model is highly interactive and economically heavy: honest users must often wait up to three months before withdrawing funds unless liquidity guarantees are provided by external operators, which would require tens of billions of dollars in capital to scale.

BitVM2 [LAA+25] improves upon this design by restructuring how computation is verified. Instead of expressing logic as a garbled Boolean circuit and resolving disputes gate by gate, BitVM2 models computation as a program - a sequence of high-level execution steps. Each step represents a coherent portion of the computation, and only the segment containing an inconsistency must be challenged on-chain. Verification is carried out through a few on-chain transactions that publish intermediate results, which may then be contested if discrepancies are detected. Any participant can act as a challenger by identifying a segment where an inconsistency occurs. This approach dramatically reduces interactivity and on-chain footprint while preserving verifiability of arbitrary computation.

BitVM3 [FLB25] further refines the original BitVM model by replacing interactive search of the fraud gate with succinct zero-knowledge proofs. In this version, Prover (acting as the Garbler) produces a zk-proof attesting that the output label hashes correctly correspond to the input label hashes of the garbled circuit. If the zk-proof fails verification, the protocol halts immediately, eliminating the need for any additional challenge-response rounds. This design significantly simplifies the on-chain logic and reduces the operational overhead associated with dispute resolution.

The Glock25 variant [Eag25] replaces BitVM3's heavy zk-proof requirement with a cut-and-choose protocol, trading computational cost for interactivity. Instead of generating expensive zk-proofs, participants engage in multiple randomized checks that collectively guarantee circuit correctness with high probability. Glock attains this efficiency through advanced cryptographic primitives such as verifiable secret sharing (VSS), adaptor signatures, and oblivious transfer. These tools maintain soundness while avoiding costly proof generation, though they introduce greater coordination complexity and off-chain communication overhead.

A contrasting, non-optimistic approach is proposed by ColliderVM [KLN25], where zk-proof verification occurs entirely on-chain and is distributed across multiple transactions. To ensure that all parts of the Verifier operate on the same proof instance, each transaction includes a consistency check relying on the collision resistance of the underlying hash function. The system parameters define a constant security gap that quantifies the minimal security degradation between verification stages. While conceptually elegant, ColliderVM remains far from practical deployment due to its substantial on-chain footprint, the large amount of data required from the Verifier, and the heavy computational burden on the Prover to maintain consistency across hash-linked verification steps.

Witness Encryption (WE). WE, introduced by Garg, Gentry, Sahai and Waters [GGSW13], is a class of encryption schemes that use an arbitrary NP statement x as a public key to encrypt a message m. For a true statement x, any party that has the corresponding witness w for x can decrypt to obtain m, whereas for a false statement x, the message m stays computationally hidden. WE is known to be implied by indistinguishability obfuscation (iO) [GGSW13, GGH+13], however, WE appears to be significantly weaker primitive than iO, given the black-box separations between these primitives [GMM17]. Despite WE being seemingly weaker than iO, the only known way to construct WE for NP under standard assumptions is by relying on the recent iO constructions that are based on well-studied hardness assumptions [JLS21, JLS22, RVV24], which in turn are quite inefficient. On the other hand, if we look at direct constructions of WE for NP, then we need to rely on a non-standard and non-falsifiable lattice-based assumption (referred to as evasive LWE) [VWW22, Tsa22]. Similarly, we know that multi-input ABE

(for an arbitrary *polynomial* number of inputs) implies WE for NP [BJK+18]. Though, as of yet, we solely have multi-input ABE constructions that can support only *constant* number of inputs (which is insufficient to obtain WE for NP), and that also under the aforementioned (non-falsifiable) evasive LWE assumption. We remark that the evasive LWE assumption has seen several recent counterexamples [BÜW24, AMYY25, HJL25, DJM+25]. Lastly, prior work [BIJ+20] showed how to construct witness encryption for NP using *affine determinant programs*, which are closely related to AADPs. However, when applied to SNARK-verifiable predicates, the resulting ciphertexts become prohibitively large and infeasible to store.

On the other hand, several theoretical works studied the relationship of WE for some language L with other cryptographic primitives. For example, we know that every language L that has a *smooth projective hash function* [ABP15] (i.e., hash proof systems [CS02]) unconditionally has a WE (but the converse is not known). However, all such languages are in the complexity class SZK [GGSW13], and hence, this approach is unlikely to extend to all of NP (unless the polynomial hierarchy collapses [AH91]). Recently, this was extended to show an equivalence between witness encryption for some language L and existence of *laconic* (i.e., the prover's communication is logarithmic) special-honest verifier zero-knowledge (SHVZK) argument for L [LMP25]. This implies that constructing laconic SHVZK for NP is as hard as constructing WE for NP (and other way around).

Given this state of affairs, recent works focused on actual constructions of WE for *specific* languages as opposed to for all NP. For example, some recent works [BC16, BL20, CFK24, FHAS24, GKPW24] observed that when the relation can be checked using pairing product equations with a *linear* verifier, then one can build a WE scheme for the same relation. We note that these approaches are limited to linear verifiers, and for example, cannot be used to construct WE for Groth16 SNARK [Gro16], which has a *quadratic* verifier. Nevertheless, a recent work [GHK+25] put the aforementioned works into a framework and constructed gadgets, which are succinct linearly-verifiable arguments.

1.4 Workflow of PIPE

To build intuition for the upcoming formalism, we begin by describing at a high level how PIPE can be deployed and executed in practice (See Figure 1). From a developer's point of view, a PIPE output behaves like a programmable Bitcoin address: coins sent to it can only be spent when a specific cryptographic condition is satisfied. This lets developers enforce complex rules directly through cryptography. In more detail:

1. **Step 1: Create PIPE output (Developer).** The developer defines the spending rule that should unlock the funds. This rule is expressed as an NP statement x. Then, the PIPE's setup process generates a Bitcoin public key pk and a pipe state $\mathsf{pipe} = (\mathsf{ct}, \pi)$ which includes an encrytpted key and correctness proof. Internally, the setup process encrypts a private signing key sk using a WE scheme tied to x, meaning sk can only be recovered when the rule described by x is proven true. Additionally, it generates a proof that sk is guaranteed to be recovered if the rule defined by x is satisfied. The developer then publishes pipe and pk which is the corresponding public key of sk. Here pk is essentially, a Bitcoin address with cryptographic logic baked in.

2. **Step 2: Lock Funds (Depositor).** A depositor sends Bitcoin to the PIPE address derived from pk. To Bitcoin, this looks like a normal P2PKH or P2TR transaction. It does not need to know anything about the PIPE condition. Before locking, the depositor verifies that the PIPE's setup process was correctly executed using the correctness proof π so that it is guaranteed that the locked amount can be withdrawn if the conditions are satisfied.

3. **Step 3: Prepare Witness (Recipient).** When the condition in x is met, the recipient generates a *witness* w such that $(x, w) \in R$. In practice, this could be a zkVM execution trace, or any zero-knowledge proof attesting that the rule is satisfied.

4. **Step 4: Decrypt and Sign (Recipient).** The recipient runs PIPE's signing algorithm. Internally, the PIPE's signing algorithm decrypts sk using the valid witness w and produces a standard Schnorr signature σ.

5. **Step 5: Submit Transaction (Recipient → Miner).** The recipient broadcasts (m, σ) to the Bitcoin network. Miners and nodes simply verify σ as a standard Schnorr signature under pk. Thus, the transaction confirms only if the condition in x was actually satisfied.

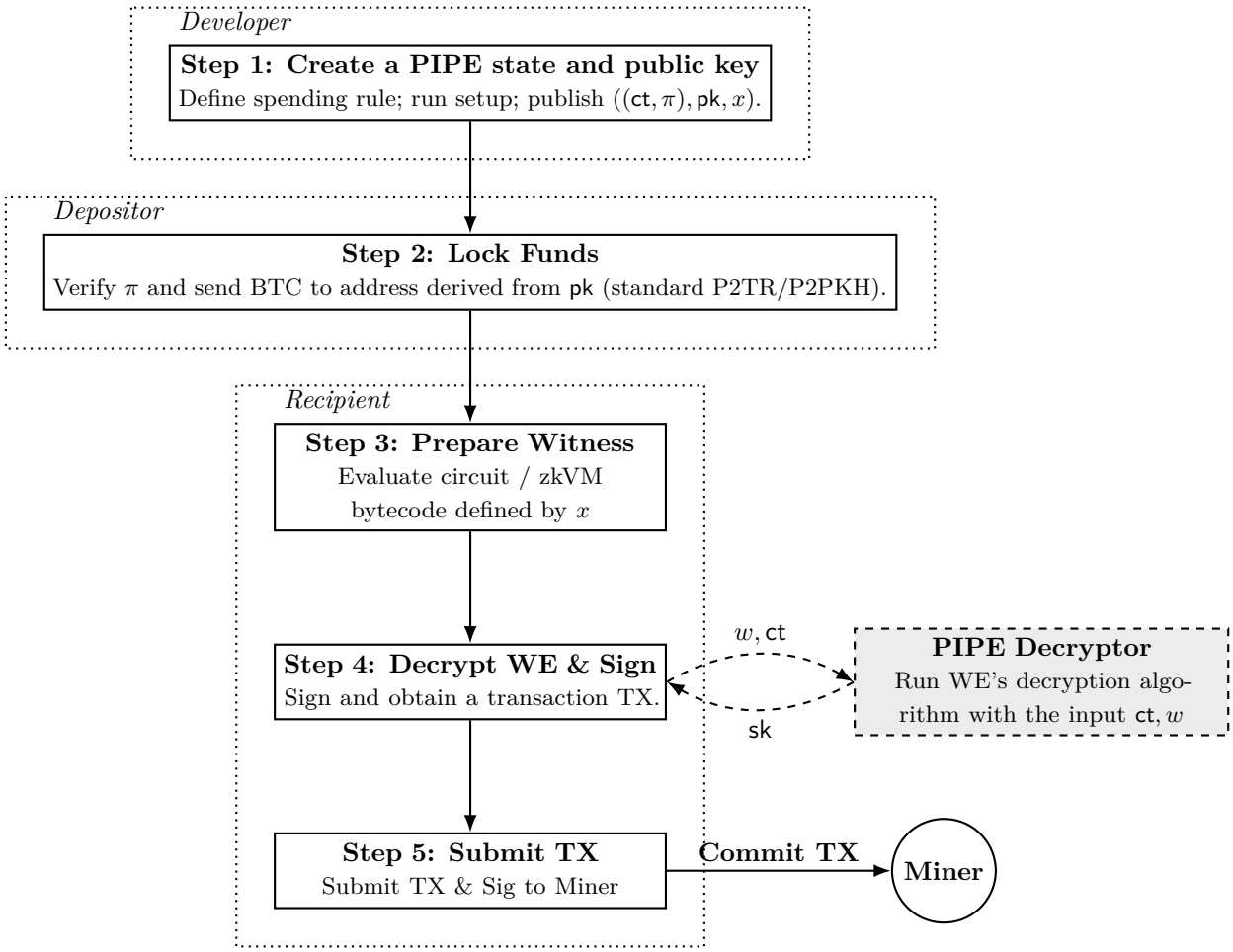

Figure 1: ***PIPE Workflow.*** *Developers create a PIPE by encrypting the signing key under a condition x and publishing* $(\mathsf{pipe}, \mathsf{pk}, x)$*. Depositors lock BTC to* pk *using a normal transaction. When a recipient can produce a witness w such that $(x, w) \in \mathcal{R}$, they use PIPE's signing algorithm to generate a Schnorr signature σ. Miners see only a standard transaction and verify σ as usual.*

From a builder's perspective, a PIPE acts as a *programmable covenant layer* on top of Bitcoin: it lets developers define arbitrary off-chain logic that is enforced cryptographically on-chain. This opens the door to metaprotocols [WLT+24], where Bitcoin L1 acts as a settlement layer for higher-level applications such as trustless bridges, programmable vaults, and zkRollups—all built using standard transactions that Bitcoin already supports.

1.5 BitVM functionality using PIPE

BitVM3/Glock-style protocols implement optimistic verification by having an operator commit to a claimed computation result and allowing challengers to expose inconsistencies. In existing designs, this commitment is implemented using garbled circuits: the operator publishes garbling tables and commitments to input and output labels. Disprove condition is equivalent to a knowledge of the false output label. Although most of this data resides off-chain, the resulting scripts still require multiple conditional hash checks and nontrivial control flow, increasing engineering complexity and on-chain footprint.

PIPE offers a more compact and consensus-friendly way to express the disprove condition. The operator publishes a PIPE instance that can be opened only when the inconsistent case $f(w) = 0$ holds. The Signers Committee verifies the instance using the built-in PIPE.OutVerify algorithm. The optimistic path uses a smaller `AssertTx`: instead of checking per-bit label commitments, it verifies only a single hash of the operator's witness. Disprove transactions become structurally simpler as well: a challenger provides a PIPE opening and the transaction verifies a standard signature. It's comparable in size to the GC-based output-label check. One might worry that a challenger could fabricate an arbitrary $w' : f(w') = 0$. To

prevent it PIPE condition binds the operator to a specific hashed witness, so a valid disproof requires opening that exact commitment, preventing trivial or artificial false-witness attacks. See § 6 for details.

Although PIPE-BitVM's on-chain footprint is smaller and simpler, its off-chain setup and disprove costs depend entirely on the underlying PIPE primitives. In particular, the efficiency of the Witness Encryption scheme and the structure of the underlying NIZK determine the computational cost of producing and opening a PIPE instance. The PIPE-based construction introduces no additional asymptotic overhead beyond these primitives; it simply inherits their performance profile. As a result, future improvements in PIPE's underlying components translate directly into faster and more efficient instantiations of the BitVM-style protocol.

2 Preliminaries

2.1 Notational Conventions

We denote by $\mathbb{Z}$ and $\mathbb{N}$ the sets of integers and natural numbers (positive integers). We denote the security parameter by $\lambda \in \mathbb{N}$, by which each scheme and adversary is parameterized. For $n \in \mathbb{N}$, set $[n] = \{1, 2, \ldots, n\}$. We denote by $x \leftarrow X$ the uniform sampling of the variable x from the set X. We write $x \leftarrow \mathsf{A}(y)$ to denote that a probabilistic polynomial time (PPT) algorithm A on input y, outputs x. When we want to make the randomness r used by A explicit, we write it as $x \leftarrow \mathsf{A}(y; r)$. If A has oracle access to a procedure O, we represent this by superscript A^{O}. If A is a deterministic polynomial time (DPT) algorithm, we use the notation $x \coloneqq \mathsf{A}(y)$. We define *polynomial* functions as $\mathsf{poly}(\lambda) = \bigcup_{d \in \mathbb{N}} O(\lambda^d)$ and *negligible* functions as $\mathsf{negl}(\lambda) = \bigcap_{d \in \mathbb{N}} o(\lambda^{-d})$.

2.2 Hard Relation

We recall the definition of hard relation from [GSST24]. It intuitively says that no efficient adversary can find a valid witness w for a random statement x drawn from the relation R, except with negligible probability.

Definition 2.1 (Hard Relation [GSST24])**.** *Let $R = (R_\lambda)_{\lambda \in \mathbb{N}} \subseteq \mathcal{D}_x \times \mathcal{D}_w = (\mathcal{D}_{x,\lambda} \times \mathcal{D}_{w,\lambda})$ be a (poly-time decidable) binary-relation family over statement and witness pairs $(x, w) \in \mathcal{D}_{x,\lambda} \times \mathcal{D}_{w,\lambda}$. We say that R is a hard-relation family if it satisfies the following properties:*

- ***Efficient Sampling.*** *There exists a PPT sampling algorithm $\mathsf{RGen}(1^\lambda)$ that on input a (unary encoded) security parameter λ outputs a pair (x, w) such that $(x, w) \in R_\lambda$.*
- ***Hardness.*** *We say a relation is hard if for every PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$\Pr\left[(x, w^*) \in R_\lambda \mid (x, w) \leftarrow \mathsf{RGen}(1^\lambda), w^* \leftarrow \mathcal{A}(x)\right] \leq \mathsf{negl}(\lambda),$$

  where the probability is taken over the random coins of RGen and $\mathcal{A}$.

Whenever the security parameter is clear from the context, we abuse the notation for readability and write R instead of R_λ.

2.3 Non-interactive Zero-knowledge Argument

Let R be a binary relation as defined above, and L the language consisting of statements in R. A non-interactive zero-knowledge (NIZK) argument system [BFM88] for language L allows to prove in non-interactive manner that some statements are in L without leaking information about the corresponding witness. We formally define it as follows.

Definition 2.2 (Non-interactive Zero-knowledge Argument System)**.** *A non-interactive zero-knowledge (NIZK) argument system NIZK for a language $L \in NP$ (with witness relation R) is a tuple of PPT algorithms defined as follows*

$\mathsf{PGen}(1^\lambda) \to \mathsf{crs}$**:** *On input a (unary encoded) security parameter λ, it outputs a common reference string* crs.

$\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$**:** *On input a common reference string* crs*, a statement* x *and a witness* w*, it outputs a proof* π*.*

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$**:** *On input a common reference string* crs*, a statement* x *and a proof* π*, it outputs a bit* $b \in \{0,1\}$*.*

We require a NIZK argument system to satisfy the following properties:

- ***Completeness.*** *For every* $(x,w) \in R$*, we have that*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{PGen}(1^\lambda), \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\right] = 1.$$

- ***Computational soundness.*** *For every* $x \notin L$ *and all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{PGen}(1^\lambda), \pi \leftarrow \mathcal{A}(\mathsf{crs}, x)\right] \leq \mathsf{negl}(\lambda).$$

- ***Computational zero-knowledge.*** *There exists a PPT algorithm* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for every PPT adversary* $\mathcal{A}$*,*

$$\left| \Pr\left[\mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1 \mid \mathsf{crs} \leftarrow \mathsf{PGen}(1^\lambda)\right] \right.$$
$$\left. - \Pr\left[\mathcal{A}^{\mathcal{O}(\mathsf{crs},\tau,\cdot,\cdot)}(\mathsf{crs}) = 1 \mid (\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)\right] \right| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{O}(\mathsf{crs}, \tau, \cdot, \cdot)$ *is an oracle that outputs* $\perp$ *on input* (x,w) *such that* $(x,w) \notin R$ *and outputs* $\pi \leftarrow \mathcal{S}_2(\mathsf{crs}, \tau, x)$ *when* $(x,w) \in R$*.*

2.4 Symmetric Encryption

We recall the definition of symmetric encryption scheme.

Definition 2.3 (Symmetric Encryption)**.** *A symmetric encryption scheme* SKE *with key space* $\mathcal{K}$ *and message space* $\mathcal{M}$ *is a tuple of PPT algorithms:*

$\mathsf{KGen}(1^\lambda) \to k$**:** *On input a (unary encoded) security parameter* 1^λ*, it outputs a key* $k \in \mathcal{K}$*.*

$\mathsf{Enc}(k, m) \to \mathsf{ct}$**:** *On input a key* $k \in \mathcal{K}$ *and message* $m \in M$*, it outputs a ciphertext* ct*.*

$\mathsf{Dec}(k, \mathsf{ct}) \to m$**:** *On input a key* $k \in \mathcal{K}$ *and a ciphtext* ct*, it outputs a message* $m \in \mathcal{M}$*.*

Correctness. A symmetric encryption scheme SKE is correct, if for all $\lambda \in NN$, all $k \leftarrow \mathsf{KGen}(1^\lambda)$, and all $m \in \mathcal{M}$, we have that

$$\Pr\left[\mathsf{Dec}(k, \mathsf{ct}) = m \mid \mathsf{ct} \leftarrow \mathsf{Enc}(k, m)\right] = 1.$$

Security. We only require a very weak notion of security called indistinguishability against eavesdroppers (IND-EAV security).

Definition 2.4 (EAV-Security)**.** *A symmetric encryption scheme* SKE *is indistinguishable against eavesdroppers (IND-EAV) secure, if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{IND\text{-}EAV}_{\mathcal{A},\mathsf{SKE}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{IND\text{-}EAV}_{\mathcal{A},\mathsf{SKE}}$ *is defined as follows:*

$\mathsf{IND\text{-}EAV}_{\mathcal{A},\mathsf{SKE}}(\lambda)$

1 : $k \leftarrow\!\!\$\ \mathcal{K}$
2 : $(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)$
3 : $b \leftarrow\!\!\$\ \{0,1\}$
4 : $ct \leftarrow \mathsf{Enc}(k, m_b)$
5 : $b' \leftarrow \mathcal{A}(\mathsf{ct})$
6 : **return** $b = b'$

2.5 Digital Signature

We recall the definition of a digital signature scheme.

Definition 2.5 (Digital Signature)**.** *A digital signature scheme* DS *is a tuple of PPT algorithms defined as follows*

$\mathsf{KGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$**:** *On input a (unary encoded) security parameter* λ*, it outputs a secret and public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$**:** *On input a secret key* sk *and a message* $m \in \{0,1\}^*$*, it outputs a signature* σ*.*

$\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to b$**:** *On input a public key* pk*, a message* $m \in \{0,1\}^*$ *and a signature* σ*, it outputs a bit* $b \in \{0,1\}$*.*

Correctness. A digital signature scheme DS is correct, if for all $\lambda \in \mathbb{N}$ and every message $m \in \{0,1\}^*$, we have that

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, m, \sigma) \mid (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda), \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\right] = 1.$$

Security. The most common security requirement of a signature scheme is existential unforgeability under chosen message attack (EUF-CMA). At a high level, it guarantees that an adversary that does not know the secret key, cannot produce a valid signature on a message m even if the adversary learns polynomially many valid signatures on messages of their choice (but different from m). We formalize it as follows.

Definition 2.6. *A digital signature scheme* DS *is EUF-CMA secure if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{EUF\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{EUF\text{-}CMA}_{\mathcal{A},\mathsf{DS}}$ *is defined as follows:*

$\mathsf{EUF\text{-}CMA}_{\mathcal{A},\mathsf{DS}}(\lambda)$	$\mathcal{O}_{\mathsf{Sign}}(m)$
$1: \mathcal{Q} \coloneqq \emptyset$	$1: \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$
$2: (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$	$2: \mathcal{Q} \coloneqq \mathcal{Q} \cup \{m\}$
$3: (m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(\cdot)}(\mathsf{pk})$	$3:$ **return** σ
$4:$ **return** $(m \notin \mathcal{Q} \wedge \mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1)$	

2.6 Witness Encryption

We recall the definition of witness encryption [GGSW13].

Definition 2.7 (Witness Encryption [GGSW13])**.** *A witness encryption (WE) scheme* WE *for a language* L *(with corresponding witness relation* R *and message space* $\mathcal{M} \subseteq \{0,1\}^*$*) consists of a pair of PPT algorithms:*

$\mathsf{Enc}(1^\lambda, x, m) \to \mathsf{ct}$**:** *On input a (unary encoded) security parameter* λ*, a string* $x \in \{0,1\}^*$ *and a message* $m \in \mathcal{M}$*, it outputs a ciphertext* ct*.*

$\mathsf{Dec}(\mathsf{ct}, w) \to m \vee \perp$**:** *On input a ciphertext* ct *and a string* $w \in \{0,1\}^*$*, it outputs a message* $m \in \mathcal{M}$ *or (a designated error symbol)* $\perp$*.*

Correctness. A witness encryption scheme WE for a language L with witness relation R is correct, if for all $\lambda \in \mathbb{N}$, for any $m \in \mathcal{M}$, and for any $x \in L$ such that $(x, w) \in R$ holds, we have that

$$\Pr\left[\mathsf{Dec}(\mathsf{ct}, w) = m \mid \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, x, m)\right] = 1.$$

Security. In this paper we consider the extractable security as given in [GG17]. Intuitively, such a witness encryption scheme is said to be secure if an adversary can learn some non-trivial information about the encrypted message only if it knows a witness for the instance used during encryption. We define this formally as follows.

Definition 2.8 (Extractable Security [GG17])**.** *A witness encryption scheme* WE *for a language* L *with witness relation* R *is extractable secure if for every* $\lambda \in \mathbb{N}$*, every PPT adversary* $\mathcal{A}$ *with polynomial* $q(\cdot)$*, there exists a PPT extractor* $\mathcal{E}$ *and a polynomial* $p(\cdot)$*, such that for every pair of messages* $m_0, m_1 \in \mathcal{M}$*, every string* $x \in \{0,1\}^*$*, the following holds*

$$\Pr\left[b = b' \mid b \leftarrow\!\$\ \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, x, m_b), b' \leftarrow \mathcal{A}(1^\lambda, x, \mathsf{ct})\right] \geq \frac{1}{2} + \frac{1}{q(\lambda)}$$

$$\implies \Pr\left[(x,w) \in R \mid w \leftarrow \mathcal{E}(1^\lambda, x, m_0, m_1)\right] \geq \frac{1}{p(\lambda)}.$$

2.7 Witness Key Encapsulation Mechanism

We also recall the definition of witness key encapsulation mechanism (WKEM) as defined in [CV21].

Definition 2.9 (Witness Key Encapsulation Mechanism (WKEM) [CV21])**.** *A* witness key encapsulation mechanism *(WKEM) scheme for a language* L *(with corresponding relation* R *and key space* $\mathcal{K}$*) is a pair of PPT algorithms:*

$\mathsf{Encap}(1^\lambda, x) \to (\mathsf{ct}, k)$**:** *On input a (unary encoded) security parameter* λ *and a string* $x \in \{0,1\}^*$*, it outputs a ciphertext* ct *and a key* $k \in \mathcal{K}$*.*

$\mathsf{Decap}(\mathsf{ct}, w) \to k \vee \perp$**:** *On input a ciphertext* ct *and a string* $w \in \{0,1\}^*$*, it outputs a key* $k \in \mathcal{K}$ *or (a designated error symbol)* $\perp$*.*

Correctness. A WKEM scheme WKEM for a language L with witness relation R is correct, if for all $\lambda \in \mathbb{N}$, and for any $x \in L$ such that $(x,w) \in R$ holds, we have that

$$\Pr\left[\mathsf{Decap}(\mathsf{ct}, w) = k \mid (\mathsf{ct}, k) \leftarrow \mathsf{Encap}(1^\lambda, x)\right] = 1.$$

Security. We recall the extractability definition from [CV21]. Intuitively, it ensures that an efficient adversary cannot decapsulate a key without *knowledge* of a valid witness w to the statement x.

Definition 2.10 (Extractability [CV21])**.** *A WKEM scheme* WKEM *for a language* L *with witness relation* R *is* extractable*, if for every* $\lambda \in \mathbb{N}$*, there exists a PPT extractor* $\mathcal{E}$ *such that for any stateful PPT adversary* $\mathcal{A}$*, such that*

$$\Pr\left[\mathsf{KEM\text{-}CPA}_{\mathcal{A},\mathsf{WKEM}}(\lambda) = 1\right] \geq \frac{1}{2} + \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$*, it holds that*

$$\Pr\left[(x,w) \in R \;\middle|\; \begin{array}{l} x \leftarrow \mathcal{A}(1^\lambda) \\ w \leftarrow \mathcal{E}^{\mathcal{A}}(1^\lambda, x) \end{array}\right] \geq \delta(\lambda),$$

for some non-negligible function $\delta(\lambda)$*. The experiment* $\mathsf{KEM\text{-}CPA}_{\mathcal{A},\mathsf{WKEM}}$ *is defined as follows*

$\mathsf{KEM\text{-}CPA}_{\mathcal{A},\mathsf{WKEM}}(\lambda)$

1 : $x \leftarrow \mathcal{A}(1^\lambda)$
2 : $b \leftarrow\!\$\ \{0,1\}$
3 : $(\mathsf{ct}, k_0) \leftarrow \mathsf{Encap}(1^\lambda, x)$
4 : $k_1 \leftarrow\!\$\ \mathcal{K}$
5 : $b' \leftarrow \mathcal{A}(\mathsf{ct}, k_b)$
6 : **return** $b = b'$

From WKEM to WE. Following the result of Fleischhacker et al. [FHAS24], any *extractable* WKEM can be directly transformed into an *extractable* WE scheme by combining it with an EAV-secure symmetric encryption scheme (Definition 2.4). The construction is depicted in Figure 2.

$\mathsf{WE.Enc}(1^\lambda, x, m)$	$\mathsf{WE.Dec}(\mathsf{ct}, w)$
$1: (\mathsf{ct}_1, k) \leftarrow \mathsf{WKEM.Encap}(1^\lambda, x)$	$1:$ **parse** $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2)$
$2: \mathsf{ct}_2 \leftarrow \mathsf{SKE.Enc}(k, m)$	$2: k := \mathsf{WKEM.Decap}(\mathsf{ct}_1, w)$
$3:$ **return** $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2)$	$3: m := \mathsf{SKE.Dec}(k, \mathsf{ct}_2)$
	$4:$ **return** m

Figure 2: *Construction of an extractable WE from an extractable WKEM [FHAS24].*

The security of the construction is established with the following theorem from [FHAS24].

Theorem 1 ([FHAS24])**.** *Let* WKEM *be an extractable WKEM for a language L (with witness relation R and key space $\mathcal{K}$) and* SKE *be an IND-EAV secure symmetric encryption scheme (with key space $\mathcal{K}$ and message space $\mathcal{M}$). Then,* WE *construction from Figure 2 is an extractable secure WE scheme for the language L and message space $\mathcal{M}$.*

3 Witness Signature Scheme

In this section, we introduce a new cryptographic primitive, the ***Witness Signature (WS)*** scheme, which formalizes the security foundations of our approach and captures the properties required to achieve our goals. In a nutshell, WS is a cryptographic scheme that generates a signature if certain conditions are satisfied.

Definition 3.1 (Witness Signature)**.** *Let $R \subseteq \mathcal{D}_x \times \mathcal{D}_w$ be a hard relation (Definition 2.1), and let $L_R = \{x \mid \exists w\colon (x, w) \in R\}$ denote the corresponding language of valid statements. Let $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme. A WS scheme with respect to a hard relation R and a signature scheme* DS *is a conditional signature scheme consisting of algorithms $\mathsf{WS}_{R,\mathsf{DS}} = (\mathsf{Setup}, \mathsf{OutVerify}, \mathsf{WSign})$ defined as follows:*

$\mathsf{Setup}(1^\lambda, x) \to (\mathsf{wout}, \mathsf{pk})$**:** *On input a (unary encoded) security parameter λ and a statement $x \in \{0,1\}^*$, it outputs a state* wout *and a public key* pk*.*

$\mathsf{OutVerify}(\mathsf{wout}, \mathsf{pk}, x) \to b$**:** *On input a state* wout*, a statement $x \in L_R$ and a public key* pk*, it outputs a bit $b \in \{0,1\}$ (i.e., 1 if* wout, pk *is valid, and 0 otherwise).*

$\mathsf{WSign}(\mathsf{wout}, w, m) \to \sigma \vee \perp$**:** *On input a state* wout*, a witness w, and a message m, it outputs a signature σ if $(x, w) \in R$ and $\perp$ otherwise.*

We now introduce the security properties of a WS scheme.

We require a WS scheme to satisfy correctness, unforgeability and verifiability as we define below. Roughly speaking, correctness requires that whenever a valid witness is used to sign, the resulting signature always verifies.

Definition 3.2 (Correctness)**.** *A WS scheme $\mathsf{WS}_{R,\mathsf{DS}}$ with respect to a hard relation R and a signature scheme* DS *satisfies correctness, if for all $\lambda \in \mathbb{N}$, all $(x, w) \leftarrow \mathsf{RGen}(1^\lambda)$, and all messages $m \in \{0,1\}^*$,*

$$\Pr\left[\mathsf{DS.Verify}(\mathsf{pk}, \sigma, m) = 1 \;\middle|\; \begin{array}{r}(\mathsf{wout}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda, x) \\ \sigma \leftarrow \mathsf{WSign}(\mathsf{wout}, w, m)\end{array}\right] = 1.$$

Next, we define unforgeability, which ensures that producing a valid signature is no easier than finding a corresponding witness for the underlying statement.

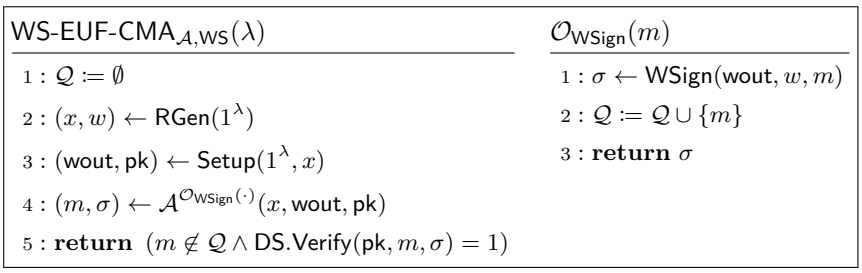

Figure 3: *The WS-EUF-CMA experiment*

Definition 3.3 (Unforgeability)**.** *A WS scheme* $\mathsf{WS}_{R,\mathsf{DS}}$ *with respect to a hard relation* R *and a signature scheme* DS *is existentially unforgeable under adaptive chosen-message attacks (WS-EUF-CMA) if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{WS\text{-}EUF\text{-}CMA}_{\mathcal{A},\mathsf{WS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{WS\text{-}EUF\text{-}CMA}_{\mathcal{A},\mathsf{WS}}$ *is defined as in Figure 3, and the probability is taken over the randomness of all probabilistic algorithms and of* $\mathcal{A}$*.*

Finally, we introduce the *verifiability* property of WS, which ensures that the output produced during the setup phase can be publicly validated as a well-formed WS instance for honestly generated statements. This property allows third parties to confirm that the WS state was correctly generated without participating in the signing process.

Definition 3.4 (Verifiability)**.** *A WS scheme satisfies verifiability if for all* $\lambda \in \mathbb{N}$*, all statement/witness pairs* $(x, w) \leftarrow \mathsf{RGen}(1^\lambda)$*, all public keys* pk*, all messages* $m \in \{0,1\}^*$*, all states* $\mathsf{wout} \in \{0,1\}^*$*, we have*

$$\mathsf{OutVerify}(\mathsf{wout}, \mathsf{pk}, x) = 1 \implies \mathsf{DS.Verify}(\mathsf{pk}, \mathsf{WSign}(\mathsf{wout}, w, m), m) = 1,$$

with all but negligible probability.

In § 4, we introduce our WS construction based on witness encryption (WE) and call it PIPE.

4 PIPE

In the first version of this work [Kom24], we explored constructing witness signature using more expressive cryptographic primitives such as *functional encryption* (FE) [BSW11, O'N10] and *indistinguishability obfuscation* (iO) [BGI+01, GGH+13]. Both of these paradigms provide a natural conceptual foundation for realizing conditional signature functionality: in principle, they allow a signing key to be cryptographically bound to a statement x, and to become usable only when the corresponding witness w is available. Although both FE- and iO-based instantiations capture the abstract notion of conditional signing, our current analysis shows that they are not optimal in terms of *practicality* and *implementability*.

For these reasons, in the present version of the paper, we focus on the *WE-based instantiation of WS* that we call *PIPE*, which provides the most balanced combination of generality, conceptual clarity, and practical feasibility. WE, at least for simple relations, can be constructed using well-studied cryptographic assumptions and allows for a simpler security analysis.

Our construction PIPE. Given that our aim is to deploy our construction on top of Bitcoin, we describe a WS scheme where the underlying digital signature scheme is Schnorr signature. We remark though that our construction works for any unforgeable signature scheme. PIPE is a $\mathsf{WS}_{R,\mathsf{Schnorr}}$ scheme (as per Definition 2.7), constructed using a WE scheme for a hard relation R, the Schnorr signature scheme $\mathsf{Schnorr}$, and a NIZK argument system $\mathsf{NIZK}_{R'}$ (Definition 2.2) for a relation R' defined below.

$\mathsf{Setup}(1^\lambda, x)$: Sample $\mathsf{crs} \leftarrow \mathsf{NIZK}_{R'}.\mathsf{PGen}(1^\lambda)$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Schnorr.KGen}(1^\lambda; r_{\mathsf{sig}})$, where r_{sig} is the randomness of the algorithm. Compute

$$\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, x, \mathsf{sk}; r_{\mathsf{enc}}),$$

i.e., encrypt sk under the statement x using the witness encryption scheme.

Next, compute $\pi \leftarrow \mathsf{NIZK}_{R'}.\mathsf{Prove}(\mathsf{crs}, (x, \mathsf{pk}, \mathsf{ct}), (\mathsf{sk}, r_{\mathsf{sig}}, r_{\mathsf{enc}}))$, where the relation R' is defined as follow

$$R' = \left\{((x, \mathsf{pk}, \mathsf{ct}), (\mathsf{sk}, r_{\mathsf{sig}}, r_{\mathsf{enc}})) \mid (\mathsf{sk}, \mathsf{pk}) = \mathsf{Schnorr.KGen}(1^\lambda; r_{\mathsf{sig}}) \wedge \mathsf{ct} = \mathsf{WE.Enc}(1^\lambda, x, \mathsf{sk}; r_{\mathsf{enc}})\right\}.$$

The setup algorithm outputs the state $\mathsf{wout} \coloneqq (x, \mathsf{ct}, \pi)$ and the public key pk.

$\mathsf{OutVerify}(\mathsf{wout}, \mathsf{pk}, x)$**:** It parses $\mathsf{wout} = (x, \mathsf{ct}, \pi)$ and outputs $\mathsf{NIZK}_{R'}.\mathsf{Verify}(\mathsf{crs}, (x, \mathsf{pk}, \mathsf{ct}), \pi)$.

$\mathsf{WSign}(\mathsf{wout}, w, m)$**:** It parses $\mathsf{wout} = (x, \mathsf{ct}, \pi)$, and checks whether $(x, w) \in R$. If $(x, w) \notin R$, then it returns $\perp$. Otherwise, computes

$$\mathsf{sk} \leftarrow \mathsf{WE.Dec}(\mathsf{ct}, w).$$

Then, it computes a Schnorr signature

$$\sigma \leftarrow \mathsf{Schnorr.Sign}(\mathsf{sk}, m)$$

and outputs σ.

To verify a signature, the verifier runs the Schnorr signature verification algorithm:

$$b \leftarrow \mathsf{Schnorr.Verify}(\mathsf{pk}, m, \sigma).$$

Intuitively, the PIPE scheme "locks" a signing key inside a *witness encryption* tied to a public statement x. The setup commits to a valid Schnorr public key and produces a ciphertext that hides the corresponding secret key, which can only be recovered by someone who knows a valid witness w for x. A zero-knowledge proof ensures that the ciphertext was correctly generated and indeed contains the secret key matching the public key. Thus, the signing capability remains unusable until the external condition $(x, w) \in R$ is satisfied at which point the key can be recovered and a valid signature produced. This enables signatures (or transactions) that become valid only when a given statement is true, a property well suited for on-chain applications such as Bitcoin covenants.

Remark. We note that the setup phase of PIPE should ideally be executed as a multi-party computation (MPC) protocol. In this setting, the secret signing key sk is generated via a distributed key-generation (DKG) procedure, and the MPC protocol must ensure that all values used to construct the PIPE state are computed jointly without revealing any private intermediate information to any participant.

Theorem 2 (Correctness of WS based on PIPE)**.** *If the Schnorr signature scheme* $\mathsf{Schnorr}$ *and witness encryption scheme* WE *are correct, then our construction satisfies correctness (Definition 3.2).*

It is easy to verify the theorem statement, and hence, we omit its proof here.

Theorem 3 (Verifiability of WS based on PIPE)**.** *If the Schnorr signature scheme* $\mathsf{Schnorr}$ *and the WE scheme* WE *(over the hard relation R) are correct, and the NIZK argument system* NIZK *is computationally sound, then our construction satisfies verifiability (Definition 3.4).*

Proof. Let us fix some arbitrary message $m \in \{0,1\}^*$, $\lambda \in \mathbb{N}$, statement/witness from the hard relation $(x, w) \leftarrow \mathsf{RGen}(1^\lambda)$, public key pk, and a state $\mathsf{wout} \in \{0,1\}^*$ such that

$$\mathsf{OutVerify}(\mathsf{wout}, \mathsf{pk}, x) = 1.$$

This implies that for a state $\mathsf{wout} \coloneqq (x, \mathsf{ct}, \pi)$, it holds that $\mathsf{NIZK}_{R'}.\mathsf{Verify}(\mathsf{crs}, (x, \mathsf{pk}, \mathsf{ct}), \pi) = 1$. By the soundness of $\mathsf{NIZK}_{R'}$ we have that there exists (with all but negligible probability) a tuple $(\mathsf{sk}, r_{\mathsf{sig}}, r_{\mathsf{enc}})$ such that $(\mathsf{sk}, \mathsf{pk}) = \mathsf{Schnorr.KGen}(1^\lambda; r_{\mathsf{sig}}) \wedge \mathsf{ct} = \mathsf{WE.Enc}(1^\lambda, x, \mathsf{sk}; r_{\mathsf{enc}})$. Because $(x, w) \leftarrow \mathsf{RGen}(1^\lambda)$, by the correctness of WE we have that $\mathsf{WE.Dec}(\mathsf{ct}, w) = \mathsf{sk}$, and the correctness of Schnorr scheme ensures that $\mathsf{Schnorr.Verify}(\mathsf{pk}, \mathsf{Schnorr.Sign}(\mathsf{sk}, m), m) = 1$. Hence, our construction satisfies verifiability with all but negligible probability. □

Theorem 4 (WS-EUF-CMA Security of WS based on PIPE)**.** *If the Schnorr signature scheme* $\mathsf{Schnorr}$ *is EUF-CMA secure (Definition 2.6), R is a hard relation, NIZK argument system* $\mathsf{NIZK}_{R'}$ *(for a relation R') is computational zero-knowledge (Definition 2.2), and the WE scheme is* WE *extractable secure (Definition 2.8), then our construction is WS-EUF-CMA secure (Definition 3.3).*

Proof. Let $\mathcal{A}$ be a PPT adversary that wins the WS-EUF-CMA game with probability ϵ. The following sequence of games shows that ϵ is bounded by the probability of the adversary breaking the properties defined in the theorem.

Game G_0: This is the real WS-EUF-CMA experiment (Figure 3). The challenger samples $(x, w) \leftarrow \mathsf{RGen}(1^\lambda)$, runs $(\mathsf{wout}, \mathsf{pk}) \leftarrow \mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Setup}(1^\lambda, x)$ and gives $(x, \mathsf{wout}, \mathsf{pk})$ to $\mathcal{A}$. The challenger simulates the signing oracle by returning $\mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Sign}(\mathsf{wout}, w, \cdot)$. Finally, the adversary outputs (m^*, σ^*) and wins if $\mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Verify}(\mathsf{pk}, \sigma^*, m^*) = 1$ and m^* was not queried before to the WSign oracle.

Game G_1: Replace the honest Setup and Prove algorithms of the underlying NIZK argument system NIZK with simulated ones inside the $\mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Setup}$ algorithm. Concretely, $\mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Setup}$ samples $(\mathsf{crs}, \tau) \leftarrow \mathsf{NIZK}_{R'}.\mathcal{S}_1(1^\lambda)$, and computes the proof as $\pi \leftarrow \mathsf{NIZK}_{R'}.\mathcal{S}_2(\mathsf{crs}, \tau, (x, \mathsf{pk}, ct))$. By the zero-knowledge property of NIZK, the adversary's view changes only negligibly, i.e., $\left|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]\right| \leq \mathsf{negl}(\lambda)$.

Game G_2: In this game, the simulation of the WSign oracle is modified, which on input m returns $\mathsf{Schnorr}.\mathsf{Sign}(\mathsf{sk}, m)$ (using the secret key sk sampled inside the $\mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Setup}$ algorithm). Since x is assumed to be in L_R, the output of the WSign oracle is distributed identically as in the previous game due to the completeness of the PIPE scheme. Therefore, we have $\Pr[\mathsf{G}_1 = 1] = \Pr[\mathsf{G}_2 = 1]$.

Game G_3: We modify again the $\mathsf{PIPE}_{R,\mathsf{DS}}.\mathsf{Setup}$ algorithm, where instead of computing $\mathsf{ct} \leftarrow \mathsf{WE}.\mathsf{Enc}(x, \mathsf{sk})$ we compute $\mathsf{ct}' \leftarrow \mathsf{WE}.\mathsf{Enc}(x, 0^{|\mathsf{sk}|})$ for a fixed string $0^{|\mathsf{sk}|}$ (as before, we use the NIZK simulator to produce the proof π). Assume towards contradiction that $\mathcal{A}$ can distinguish these hybrids with non-negligible probability. Then, by the extractable security of WE, there exists a PPT extractor $\mathcal{E}$ that on input $(1^\lambda, x, \mathsf{sk}, 0^{|\mathsf{sk}|})$ outputs a witness w such that $(x, w) \in R$. However, such an extractor Ext can be used to construct an adversary against the hardness of R, in turn contradicting the hardness of R. Hence, the two hybrids are computationally indistinguishable, i.e., $\left|\Pr[\mathsf{G}_2 = 1] - \Pr[\mathsf{G}_3 = 1]\right| \leq \mathsf{negl}(\lambda)$.

We remark that from now on, sk is neither used by NIZK nor by WE.

Game G_4: In this final game, the success probability of the adversary $\mathcal{A}$ is exactly that of the breaking the EUF-CMA security of the $\mathsf{Schnorr}$ signature scheme. More concretely, we can construct a reduction $\mathcal{B}$ against the EUF-CMA security of the $\mathsf{Schnorr}$ signature scheme. $\mathcal{B}$ reduction receives the public key pk from the EUF-CMA challenger of the $\mathsf{Schnorr}$ signature scheme, and answers all WSign queries by forwarding them to the Sign oracle of the EUF-CMA challenger of the $\mathsf{Schnorr}$ signature scheme. Upon $\mathcal{A}$ outputting a valid fresh forgery (m^*, σ^*), $\mathcal{B}$ outputs the same pair to the EUF-CMA challenger of the $\mathsf{Schnorr}$ signature scheme. Hence, we have that $\left|\Pr[\mathsf{G}_3 = 1] - \Pr[\mathsf{G}_4 = 1]\right| \leq \mathsf{negl}(\lambda)$.

Putting everything together we obtain

$$\epsilon \leq \left|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_4 = 1]\right| \leq \mathsf{negl}(\lambda).$$

The advantages on the right-hand side are negligible by assumption. Hence, it follows that our WE-based PIPE construction is WS-EUF-CMA secure. □

5 Witness Encryption from Arithmetic Affine Determinant Programs

A central remaining challenge for deploying PIPEs is the instantiation of a WE scheme suitable for enforcing NP predicates. In this section, we describe our candidate WE construction based on *Arithmetic Affine Determinant Programs* (AADPs) [aad26]. Our approach follows the general affine determinant program (ADP) framework [BIJ+20] for witness encryption. While the construction does not admit a reduction to standard falsifiable assumptions, it offers a structured and explicit algebraic design that aligns naturally with the PIPE framework.

Both ADP and AADP-based witness encryption constructions are heuristic and do not currently admit security proofs under standard falsifiable assumptions. Instead, their security is supported by algebraic cryptanalysis. We designed the AADP variant primarily for efficiency reasons.

The ADP-based WE [BIJ+20] targets boolean NP languages and requires verification logic to be expressed as boolean circuits. As a result, arithmetic-heavy procedures such as SNARK verification must first be booleanized which leads to significant overhead. Moreover, ADPs do not natively enforce that inputs are boolean: each input bit must be explicitly constrained to lie in $\{0, 1\}$, typically by adding separate bit-check constraints through the all-accept ADP. These additional constraints further increase

the size of the resulting determinant program significantly i.e., leading to an infeasible to store ciphertext size of roughly 10^{27} field elements for a SNARK verification.

In contrast, our new design AADPs operate directly over arithmetic constraint systems defined on field elements. This allows verification logic to be expressed natively without explicit bit checking mechanism, substantially reducing the number of constraints. As a consequence, AADPs yield smaller determinant programs and more compact WE ciphertexts, making them a better fit for the PIPE framework. Our preliminary estimates indicate that, for a witness encryption scheme that verifies a SNARK proof, the resulting ciphertext size is on the order of **338 TB**, leaving substantial room for further optimization.

While we do not claim that this approach represents a final or optimized instantiation of PIPEs, it shows how existing cryptographic tools can be adapted to support powerful, SNARK-verifiable conditions without requiring any changes to Bitcoin consensus rules.

5.1 AADP-Based Witness Encryption

We now describe the witness encryption scheme based on AADPs at a level sufficient to understand the algebraic operations performed during encryption and decryption, while abstracting away from low-level matrix constructions. See our paper [aad26] for the details of the construction.

At a high level, the scheme represents NP verification as a system of arithmetic constraints, compiles these constraints into a single affine matrix whose rank drops exactly on satisfying assignments, and allows decryption using the singularity of the matrix.

Projectively safe constraint systems. An AADP is generated from an arithmetic constraint system over a large field $\mathbb{F}$. The constraint system consists of polynomial equations in variables $(x_0, x_1, \ldots, x_n) \in \mathbb{F}^{n+1}$ where x_0 is a homogenizing variable and $(x_1, \ldots, x_n)$ encode the witness.

Each constraint enforces an arithmetic relation of the form

$$a(x_0, \ldots, x_n) \cdot b(x_0, \ldots, x_n) = c(x_0, \ldots, x_n) \cdot d(x_0, \ldots, x_n),$$

where each a, b, c, d is a linear form. Systems of this form are expressive enough to encode general NP verification procedures, including SNARK verification circuits.

The constraint system is *projectively safe* (Definition 3 in [aad26]) if it admits no nontrivial solutions in which the homogenizing variable x_0 is zero. Equivalently, every assignment that satisfies all constraints must have $x_0 \neq 0$, and hence can be rescaled so that $x_0 = 1$. Under this normalization, each satisfying projective assignment corresponds to a unique affine solution. This property ensures that the constraint system has no spurious satisfying assignments that arise solely from projective rescaling rather than from a valid witness.

5.2 AADP generation

The algorithm AADP.Gen takes a projectively safe constraint system $\mathcal{C}$ and produces an arithmetic affine determinant program.

Rather than checking each constraint individually, AADP.Gen embeds *all constraints simultaneously* into a single affine matrix expression. Each constraint contributes a constant-sized matrix block whose rank depends on whether the constraint is satisfied. These blocks are combined so that:

- if all constraints are satisfied, the resulting matrix has rank strictly less than full;
- if any constraint is violated, the matrix becomes full rank except with negligible probability.

The output of AADP.Gen is a matrix-valued function

$$M_{\mathcal{C}} : \mathbb{F}^{n+1} \to \mathbb{F}^{2m+1 \times 2m+1},$$

defined as a sum of matrix-valued components

$$M_{\mathcal{C}}(x) := \sum_{i=0}^{m-1} M_i(x),$$

where each $M_i(x)$ is a $2m+1 \times 2m+1$ matrix whose entries are affine functions of the input variables $(x_0, \ldots, x_n)$.

Each component $M_i(x)$ corresponds to a single constraint in the system $\mathcal{C}$ and is constructed so that its rank depends on whether that constraint is satisfied by the assignment x. The full matrix $M_{\mathcal{C}}(x)$ aggregates all constraints simultaneously by summing their corresponding matrices.

Let $\mathcal{C}$ be a projectively safe arithmetic constraint system and let $M_{\mathcal{C}} \leftarrow \mathsf{AADP.Gen}(\mathcal{C})$. For any fixed input x, over the randomness of the AADP generation algorithm, the following properties hold (See Theorem 1 in [aad26]):

- If x satisfies all constraints in $\mathcal{C}$, then $\det(M(x)) = 0$ with overwhelming probability.
- If x violates at least one constraint in $\mathcal{C}$, then $M_{\mathcal{C}}(x)$ is statistically indistinguishable from a random full-rank matrix and, consequently, $\det(M(x)) \neq 0$ with overwhelming probability.

Now we are ready to introduce AADP-based WE scheme:

Definition 5.1 (AADP-based WE)**.** *Let L be an NP language represented by a projectively safe constraint system $\mathcal{C}$.*

$\mathsf{Enc}(1^\lambda, C, msg) \to \mathsf{ct}$**:** *To encrypt a message m:*

1. *run* $\mathsf{AADP.Gen}$ *on the constraint system $\mathcal{C}$ to obtain an AADP M,*
2. *embed the message msg into a designated affine component of the program, producing a ciphertext* ct *represented as an AADP i.e.,*

$$\widehat{M} = M + msg \cdot x_0 \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]^T \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]$$

We note that here, x_0 denotes a formal variable in M.

$\mathsf{Dec}(\mathsf{ct}, w) \to msg$**:** *Given a ciphertext* ct *and a witness w, the decryptor:*

1. *if w is not correct, outputs $\perp$. Otherwise,*
2. *forms the affine input $x = (x_1, \ldots, x_n)$ corresponding to w.*
3. *Solve in t :* $\det(E - t \cdot 1 \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]^T \cdot [\underbrace{0, \ldots, 0}_{2m \text{ times}}, 1]) = 0$ *which corresponds to msg*
4. *outputs msg.*

Correctness follows because for any valid witness w, the determinant vanishes, leaving a linear equation in t whose unique solution is the encrypted message. See Lemma 4 in [aad26] for the correctness proof.

Security: We emphasize that neither the original witness encryption scheme based on ADPs [BIJ+20] nor the our AADP-based construction [aad26] is supported by a reduction to standard cryptographic assumptions. Establishing a formal security proof for ADP or AADP-based witness encryption remains an important open problem. Nevertheless, existing cryptanalytic efforts have not identified any concrete attack that violates soundness in the absence of a valid witness.

Why projective safety is necessary. In our cryptanalysis, we realized that we need projective safety because AADP evaluation cannot prevent adversarial evaluation on projective inputs where $x_0 = 0$. If the underlying constraint system admits nontrivial solutions with $x_0 = 0$, then such projective assignments may cause unintended determinant cancellations. These cancellations can introduce accepting inputs that do not correspond to any valid witness for the intended NP statement. Projective safety rules out this class of attacks by guaranteeing that the only solutions to the constraint system with $x_0 = 0$ are trivial and excluded.

5.3 Instantiating PIPE with AADP-based WE

Within the PIPE framework, the AADP-based witness encryption scheme is used to encrypt a Bitcoin signing key under an NP statement. A party that possesses a valid witness can decrypt the ciphertext to recover the signing key and authorize a spend. For the purposes of PIPE constructions, we are particularly interested in NP languages whose verification can be expressed via succinct proofs.

Let L be an NP language with associated witness relation $R_L(x, w)$. That is,

$$x \in L \quad \Longleftrightarrow \quad \exists w \text{ s.t. } R_L(x, w) = 1,$$

where verification of R_L is polynomial-time. While this verification may be efficient, it need not be succinct, nor easily embeddable into a projectively safe arithmetic constraint system. Since the ciphertext size and the cost of encryption and decryption in AADP-based WE grow with the size of the encoded instance, it is important to work with succinct witnesses for efficiency.

To address this, we consider languages whose verification admits a SNARK for all NP. Let $\mathsf{SNARK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a SNARK system for R_L. Given public parameters pp, a witness w for an instance x can be compiled into a succinct proof

$$\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w),$$

such that $\mathsf{Verify}(\mathsf{pp}, x, \pi) = 1$ if and only if $(x, w) \in R_L$, except with negligible probability.

We define a derived NP language L_{snark} as follows:

$$L_{\mathsf{snark}} := \{(\mathsf{pp}, x) | \exists \pi \text{ such that } \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1\}.$$

Observe that $L_{\mathsf{snark}} \in \text{NP}$, since verification of a SNARK proof is polynomial-time, and that membership in L implies membership in L_{snark} under the completeness of the proof system.

The AADP-friendly proving system. After defining the SNARK-verifiable language L_{snark}, we briefly discuss the proving system that we suggested in the AADP framework [aad26] and explain why it is well suited for integration with AADP-based witness encryption.

The proving system proposed in the AADP paper is a pairing-based SNARK designed with *arithmetization compatibility* as a primary goal. Rather than minimizing prover time or proof size alone, the system is structured so that its verification procedure admits a clean and efficient translation into arithmetic constraints. In particular, the verifier consists of algebraic checks over field elements and group operations that can be expressed using low-degree polynomial relations.

This property is crucial for AADP-based witness encryption. Since the WE construction ultimately relies on compiling verification logic into a projectively safe arithmetic constraint system, any proving system used to define L_{snark} must admit a faithful arithmetization without introducing spurious projective solutions. The proving system in the AADP paper is explicitly designed to meet this requirement: its verification equations are naturally expressed as arithmetic constraints and can be homogenized while preserving projective safety.

Beyond its amenability to arithmetization, a second important motivation for this proving system is *constraint efficiency*. In AADP-based witness encryption, the size of the resulting determinant program and hence the size of the ciphertext grows with the number of arithmetic constraints being enforced. The proving system in [aad26] is designed so that verification can be expressed using a relatively small number of structured constraints, avoiding the large constraint blowup that can arise from generic circuit-based SNARK verifiers.

5.3.1 Cost Estimates

Following the cost model in [aad26], the arithmetized verifier requires approximately $v \approx 14083$ variables and $g \approx 14371$ multiplication constraints ("gates"). Using the AADP ciphertext size formula $\text{CTSize} \approx v(2g+1)^2|\mathbb{F}|$, this yields an estimated ciphertext size on the order of ≈ 338 TB for the concrete instantiation considered in [aad26]. We stress that these are based on a modeling script rather than a complete end-to-end implementation. Despite the substantial concrete costs, the AADP-based construction represents, to the best of our knowledge, the most efficient and concretely implementable candidate witness encryption scheme capable of verifying general SNARK proofs but its security is heuristic.

5.4 Status and open problems

AADP-based witness encryption provides a conceptually clean and expressive mechanism for realizing PIPEs, but it remains an active and unexplored research direction. In particular, the following open problems are central to advancing the practical and theoretical understanding of the approach:

- identifying and formalizing minimal algebraic conditions sufficient to guarantee soundness of the AADP-based WE scheme,
- designing proving systems and verifier arithmetizations that better align with the AADPs with the goal of reducing constraint complexity and ciphertext size,
- investigating whether batch variants of AADP-based WE are possible, allowing multiple statements or messages to be handled more efficiently since we can save on the size.

6 PIPEs BitVM enhancements

In this section we show how the PIPE primitive may be used to enhance the optimistic-verification flow of the BitVM protocol. We show that the cryptographic and on-chain components of classic BitVM such as garbled circuits (GC), label commitments, and zk-proofs of consistency map naturally to a single PIPE instance.

A BitVM computation is defined by a Boolean circuit for a predicate

$$f : \{0,1\}^n \to \{0,1\}.$$

Both BitVM and BitVM-PIPE involve the same set of actors, and the emulation works because their responsibilities align structurally.

- **Operator.** The operator claims knowledge of a witness w such that $f(w) = 1$. The operator provides all necessary information to make it possible to generate a compact fraud proof in case if $f(w) = 0$.
- **Signers Committee (SC).** Verifies the operator's setup, publishes the information provided by the operator, and signs the on-chain transactions AssertTx and DisproveTx.
  - The AssertTx enforces the operator to start the protocol's dispute period by publishing some additional information necessary for fraud proof construction.
  - The DisproveTx allows anyone to publish a fraud proof in case if the operator cheats and $f(w) = 0$.
- **Challengers.** Anyone able to detect operator's misbehavior during the dispute window and publish the DisproveTx transaction containing a valid fraud proof.

In the BitVM case a fraud proof is equivalent to knowledge of the garbled circuit false-output label. To emulate this using PIPE, we reinterpret BitVM's disprove condition as the PIPE underlying hard relation:

$$\mathcal{R} = \big(f(w) = 0\big) \wedge \big(h = \mathsf{Hash}(w)\big).$$

Below we present the step-by-step correspondence between the GC-based BitVM flow and its PIPE emulation.

BitVM using GC	**BitVM using PIPE**
Operator constructs a garbled circuit for f and sends to the SC: • garbled tables, • commitments to input labels and the false-output label, • a zk-proof of their consistency.	Operator knows a witness w with $f(w) = 1$. He computes $h = \mathsf{Hash}(w)$ and runs $\mathsf{PIPE.Setup}(1^\lambda, h) = ((h, ct, \pi), pk)$ for a hard relation $\mathcal{R}$. Operator sends the setup output to the SC.
SC verifies the zk-proof ensuring that the garbling table and label commitments are consistent. SC then publishes the garbling table and the transaction templates AssertTx and DisproveTx off-chain, and signs both transactions.	SC checks whether $\mathsf{PIPE.OutVerify}((h, ct, \pi), pk, h) = 1.$ If so, SC publishes AssertTx, DisproveTx, and $((h, ct, \pi), pk)$ off-chain.
Operator publishes AssertTx including openings of the committed input and false-output labels.	Operator publishes AssertTx including a witness w consistent with h.
If $f(w) = 0$, a challenger can compute the false-output label from the public GC data and publish DisproveTx.	If $f(w) = 0$, a challenger can compute a valid PIPE signature $\sigma = \mathsf{PIPE.WSign}((h, ct, \pi), w, m)$. The signing algorithm succeeds because $(h, w) \in \mathcal{R}$. The challenger publishes DisproveTx containing σ, and the script verifies the Schnorr signature under pk.
If no valid DisproveTx is posted during the dispute window, the optimistic path finalizes and the computation result is accepted.	If no valid DisproveTx is posted during the dispute window, the optimistic path finalizes and the computation result is accepted.

The emulation preserves the trust structure of GC-based BitVM protocols. We model the Signers Committee (SC) as a single logical party composed of n signers, and a permissionless set of challengers. Informally, the SC controls which spending conditions are put on-chain, while challengers enforce these conditions against a potentially dishonest operator.

The SC is implemented as an n-of-n multisig on Bitcoin: all n signers must cooperate to authorize AssertTx and DisproveTx. This has two consequences:

- *Liveness* depends on all n signers being online and willing to sign; a single unresponsive or malicious signer can halt progress.
- *Soundness* depends on the SC as a whole behaving honestly when the setup is produced. If at least one signer is honest and follows the protocol, an invalid setup (e.g., wrong GC, wrong PIPE instance, or wrong script template) will not be signed and therefore will never appear on-chain.

Correctness with respect to the claimed computation holds as long as at least one challenger is honest. Challengers are permissionless and untrusted; their only role is to detect operator dishonesty during the dispute window. In BitVM, this means reconstructing the false-output label from the public GC; in BitVM-PIPE, it means opening the PIPE instance when $f(w) = 0$. If the operator cheats and the on-chain script is correct, a single honest challenger suffices to trigger the DisproveTx path.

Liveness in the optimistic case requires that all n SC members remain online and sign the required transactions; no liveness assumption is placed on challengers. Finality is reached once AssertTx confirms and no valid dispute is posted before the timeout.

Under these assumptions, BitVM-PIPE inherits the same safety–liveness profile as GC-based BitVM: the SC is a fully trusted n-of-n committee that fixes the on-chain spending conditions, while a single honest challenger suffices to enforce these conditions against a dishonest operator.

One structural difference between GC-based BitVM and BitVM-PIPE concerns the timing at which the operator's witness is fixed. In BitVM, the garbling, label-commitments, and the zk-consistency proof can be generated without knowing the actual input w; the setup phase is witness-agnostic. In BitVM-PIPE, by contrast, the PIPE instance includes the value $h = \mathsf{Hash}(w)$, so the operator must know w at setup time. This distinction does not affect the optimistic-verification or dispute logic.

Beyond the structural correspondence, the on-chain footprint of BitVM-PIPE is smaller than that of GC-based BitVM.

The opening transaction AssertTx becomes smaller. In GC-based BitVM, the script must verify per-bit commitments to the input labels, which induces two hash computations for every bit of the operator's input w together with the associated control opcodes. In BitVM-PIPE, the assertion transaction checks only the single hash value $h = \mathsf{Hash}(w)$. If w is an object that fits into a single Bitcoin stack item (for example, a Groth16 proof), then the AssertTx script becomes a very small constant-size program.

The dispute transaction DisproveTx is also compact: it consists of a single signature check.

Finally, the cost of both the setup phase and its verification depends only on the underlying PIPE primitive, and therefore solely on the chosen WE construction. As a result, the entire BitVM-PIPE construction inherits its asymptotic and concrete efficiency directly from the underlying PIPE instance.

7 Conclusion

In this work, we introduced PIPEs v2, a cryptographic scheme for enforcing arbitrary spending conditions on Bitcoin without changing its consensus rules and without additional trust mechanisms. By releasing a signing key only when a witness to an NP statement is provided, PIPEs ensure that a valid Schnorr signature can be produced only if the prescribed condition holds. We captured this idea formally through a new cryptographic primitive called witness signatures, and proved that PIPEs built from witness encryption and standard signatures realize this abstraction securely.

A central question for the feasibility of PIPEs is how to instantiate a practical and secure WE scheme suitable for use on Bitcoin. In PIPEs v2, we focused on witness encryption based on AADPs as a concrete and arithmetic-native construction. AADPs enable NP predicates particularly SNARK-verifiable statements to be expressed as explicit algebraic objects, allowing spending conditions to be enforced through determinant evaluation without any on-chain changes.

While AADP-based witness encryption is not presented as a final or optimized instantiation, it provides a fully specified and analyzable framework that makes the trade-offs of cryptographically enforced spending conditions explicit. By grounding PIPEs in a single, arithmetic-native witness encryption approach, PIPEs v2 clarifies how expressive off-chain logic can be enforced in practice using existing cryptographic tools.

Overall, PIPE v2 demonstrates that expressive, verifiable spending conditions can be achieved on Bitcoin purely through cryptography. Advancing this direction, by developing WE schemes with stronger extractability guarantees, exploring alternative proof systems, and applying PIPEs to vaults, bridges, and rollup architectures, offers a promising path toward richer functionality on Bitcoin.

References

[aad26] Implementable witness encryption from arithmetic affine determinant programs. *Cryptology ePrint Archive*, 2026. (Cited on pages 2, 15, 16, 17, and 18.)

[AAL+24] Lukas Aumayr, Zeta Avarikioti, Robin Linus, Matteo Maffei, Andrea Pelosi, Christos Stefo, and Alexei Zamyatin. BitVM: Quasi-turing complete computation on Bitcoin. Cryptology ePrint Archive, Report 2024/1995, 2024. (Cited on page 5.)

[ABP15] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Disjunctions for hash proof systems: New constructions and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 69–100. Springer, Berlin, Heidelberg, April 2015. (Cited on page 6.)

[AH91] William Aiello and Johan Hastad. Statistical zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42(3):327–345, 1991. (Cited on page 6.)

[AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada. Evasive LWE: Attacks, variants & obfustopia. Cryptology ePrint Archive, Report 2025/375, 2025. (Cited on page 6.)

[BC16] Olivier Blazy and Céline Chevalier. Structure-preserving smooth projective hashing. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 339–369. Springer, Berlin, Heidelberg, December 2016. (Cited on page 6.)

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. Cryptology ePrint Archive, Report 2014/349, 2014. (Cited on page 4.)

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. (Cited on page 8.)

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001. (Cited on page 13.)

[BIJ+20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 82:1–82:39. LIPIcs, January 2020. (Cited on pages 2, 6, 15, and 17.)

[BJK+18] Zvika Brakerski, Aayush Jain, Ilan Komargodski, Alain Passelègue, and Daniel Wichs. Non-trivial witness encryption and null-iO from standard assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 425–441. Springer, Cham, September 2018. (Cited on page 6.)

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Cham, November 2020. (Cited on page 6.)

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Berlin, Heidelberg, March 2011. (Cited on page 13.)

[BÜW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 418–449. Springer, Singapore, December 2024. (Cited on page 6.)

[CFK24] Matteo Campanelli, Dario Fiore, and Hamidreza Khoshakhlagh. Witness encryption for succinct functional commitments and applications. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 132–167. Springer, Cham, April 2024. (Cited on page 6.)

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Berlin, Heidelberg, April / May 2002. (Cited on page 6.)

[CV21] Gwangbae Choi and Serge Vaudenay. Towards witness encryption without multilinear maps - extractable witness encryption for multi-subset sum instances with no small solution to the homogeneous problem. In Jong Hwan Park and Seung-Hyun Seo, editors, *ICISC 21*, volume 13218 of *LNCS*, pages 28–47. Springer, Cham, December 2021. (Cited on page 11.)

[DJM+25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 73–92. Springer, Cham, August 2025. (Cited on page 6.)

[Eag25] Liam Eagen. Glock: Garbled locks for Bitcoin. Cryptology ePrint Archive, Report 2025/1485, 2025. (Cited on page 5.)

[FHAS24] Nils Fleischhacker, Mathias Hall-Andersen, and Mark Simkin. Extractable witness encryption for KZG commitments and efficient laconic OT. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 423–453. Springer, Singapore, December 2024. (Cited on pages 6 and 12.)

[FLB25] Ariel Futoransky, Gabriel Larotonda, and Fadi Barbara. A note on the security of the BitVM3 garbling scheme. Cryptology ePrint Archive, Report 2025/1291, 2025. (Cited on page 5.)

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Cham, November 2017. (Cited on page 11.)

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. (Cited on pages 5 and 13.)

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. (Cited on pages 2, 5, 6, and 10.)

[GHK+25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 504–539. Springer, Cham, August 2025. (Cited on page 6.)

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 352–386. Springer, Cham, August 2024. (Cited on page 6.)

[GMM17] Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. Lower bounds on obfuscation from all-or-nothing encryption primitives. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 661–695. Springer, Cham, August 2017. (Cited on page 5.)

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. (Cited on page 6.)

[GSST24] Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Foundations of adaptor signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 161–189. Springer, Cham, May 2024. (Cited on page 8.)

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum iO from circular security with random opening assumption. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 3–38. Springer, Cham, August 2025. (Cited on page 6.)

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021. (Cited on page 5.)

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. (Cited on page 5.)

[JP16] Thaddeus Dryja Joseph Poon. The Bitcoin Lightning Network: Scalable off-chain instant payments. Website Link, 2016. (Cited on page 5.)

[KLN25] Victor I. Kolobov, Avihu M. Levy, and Moni Naor. ColliderVM: Stateful computation on Bitcoin without fraud proofs. Cryptology ePrint Archive, Report 2025/591, 2025. (Cited on page 5.)

[Kom24] Mikhail Komarov. Bitcoin PIPEs — Covenants and ZKPs on Bitcoin Without Soft Fork. https://www.allocinit.xyz/uploads/placeholder-bitcoin.pdf, 2024. (Cited on pages 1, 2, 3, and 13.)

[LAA+25] Robin Linus, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Andrea Pelosi, Orfeas Thyfronitis Litos, Christos Stefo, David Tse, and Alexei Zamyatin. Bridging Bitcoin to second layers via BitVM2. Cryptology ePrint Archive, Report 2025/1158, 2025. (Cited on page 5.)

[LMP25] Yanyi Liu, Noam Mazor, and Rafael Pass. On witness encryption and laconic zero-knowledge arguments. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 429–461. Springer, Cham, August 2025. (Cited on page 6.)

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013. (Cited on page 4.)

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. (Cited on page 13.)

[Poe21] Andrew Poelstra. Cat and schnorr tricks i. https://blog.blockstream.com/cat-and-schnorr-tricks-i/, January 2021. Originally published on Medium at https://medium.com/blockstream/cat-and-schnorr-tricks-i-faf1b59bd298. (Cited on page 4.)

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 3–36. Springer, Cham, December 2024. (Cited on page 5.)

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 535–559. Springer, Cham, August 2022. (Cited on page 5.)

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Cham, December 2022. (Cited on page 5.)

[WLT+24] Hongbo Wen, Hanzhi Liu, Shuyang Tang, Tianyue Li, Shuhan Cao, Domo, Yanju Chen, and Yu Feng. Stateless and verifiable execution layer for meta-protocols on Bitcoin. Cryptology ePrint Archive, Report 2024/408, 2024. (Cited on page 7.)