Bitcoin PIPES v2

Michel Abdalla, Brent Carmer, Muhammed El Gebali, **Handan Kilinc Alper**, Mikhail Komarov,

Yaroslav Rebenko, Lev Soukhanov, Erkan Tairi, Elena Tatuzova, Patrick Towa

[[alloc]init]

Bitcoin vs Programmability

Bitcoin today:

- Limited scripting language
- No native covenants (a predicate $\mathcal{P}$ over transactions)
- No direct ZK/SNARK verification
- Ongoing protocol ossification $\rightarrow$ no soft forks

What we want:

- Enforce **arbitrary spending conditions**
- Support **zkRollups, bridges, vaults, metaprotocols**

What is PIPE?

Core idea:

A signature exists $\Leftrightarrow$ an NP statement is true

On-chain view:

- Just a normal signature

Witness Signature

Witness Signature (WS)

- Let $R \subseteq \mathcal{X} \times \mathcal{W}$ be a **hard NP relation**
- Language: $L_R = \{x \mid \exists w : (x, w) \in R\}$
- Let $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$

Witness Signature (WS)

- Let $R \subseteq \mathcal{X} \times \mathcal{W}$ be a **hard NP relation**
- Language: $L_R = \{x \mid \exists w : (x, w) \in R\}$
- Let $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$

Algorithms:

- $(wout, pk) \leftarrow \mathsf{Setup}(1^\lambda, x)$
- $\sigma \leftarrow \mathsf{WSign}(wout, w, m)$
- $b \leftarrow \mathsf{OutVerify}(wout, pk, x)$

Witness Signature (WS)

- Let $R \subseteq \mathcal{X} \times \mathcal{W}$ be a **hard NP relation**
- Language: $L_R = \{x \mid \exists w : (x, w) \in R\}$
- Let $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$

Algorithms:

- $(wout, pk) \leftarrow \mathsf{Setup}(1^\lambda, x)$
- $\sigma \leftarrow \mathsf{WSign}(wout, w, m)$
- $b \leftarrow \mathsf{OutVerify}(wout, pk, x)$

Correctness:

$$(x, w) \in R \implies \mathsf{Verify}(pk, m, \sigma) = 1$$

Signing capability is equivalent to knowing a witness for x

WS Unforgeability (WS-EUF-CMA)

Experiment $\text{WS-EUF-CMA}_{\mathcal{A}}(1^\lambda)$**:**

1. $(x, w) \leftarrow \mathit{RGen}(1^\lambda)$
2. $(\mathit{wout}, \mathit{pk}) \leftarrow \text{Setup}(1^\lambda, x)$
3. $\mathcal{A}^{\mathcal{O}_{\text{WSign}}(\cdot)}(x, \mathit{wout}, \mathit{pk})$ outputs (m^*, σ^*)

Oracle:

$$\mathcal{O}_{\text{WSign}}(m): \quad \sigma \leftarrow \text{WSign}(\mathit{wout}, w, m)$$

Winning condition:

$$m^* \notin Q \ \wedge \ \text{Verify}(\mathit{pk}, m^*, \sigma^*) = 1$$

Security:

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

PIPE v2

Witness Encryption (WE)

- Relation $R \subseteq \mathcal{X} \times \mathcal{W}$
- Language: $L = \{x \mid \exists w : (x, w) \in R\}$
- Message space: $\mathcal{M} \subseteq \{0,1\}^*$

Algorithms:

- $ct \leftarrow \mathsf{Enc}(1^\lambda, x, m)$
- $m' \leftarrow \mathsf{Dec}(ct, w) \in \mathcal{M} \cup \{\perp\}$

Correctness:

$$(x, w) \in R \implies \mathsf{Dec}(ct, w) = m$$

PIPE v2

PIPE v2 is a WS instantiated using WE + Schnorr + Hard relation $R(x, w)$

PIPE v2

PIPE v2 is a WS instantiated using WE + Schnorr + Hard relation $R(x, w)$

Setup$(1^\lambda, x)$**:**

- $(sk, pk) \leftarrow \mathsf{KGen}$
- $ct \leftarrow \mathsf{WE.Enc}(1^\lambda, x, sk)$
- $wout := (x, ct, pk)$

PIPE v2

PIPE v2 is a WS instantiated using WE + Schnorr + Hard relation $R(x, w)$

Setup$(1^\lambda, x)$**:**

- $(sk, pk) \leftarrow \mathsf{KGen}$
- $ct \leftarrow \mathsf{WE.Enc}(1^\lambda, x, sk)$
- $wout := (x, ct, pk)$

WSign$(wout, w, m)$**:**

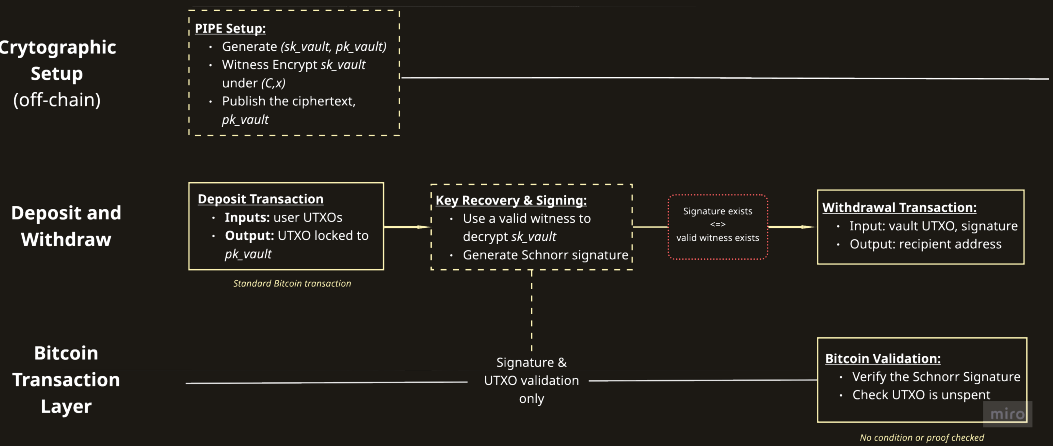
If $(x, w) \notin R$ return $\perp$
$sk \leftarrow \mathsf{WE.Dec}(ct, w)$
$\sigma \leftarrow \mathsf{Sign}(sk, m)$
return σ

Security of PIPE v2

Security relies on:

- WE extractability
- Hard relation R
- Schnorr EUF-CMA

PIPE Workflow Diagram



PIPE vs BitVM

	BitVM	PIPE
Interaction	Interactive	**Non-interactive**
Security	Challenge-based	**Witness Signature**
Verification	On-chain	**Off-chain**
Trust	1-out-of-N + liveness	**1-out-of-N only for Setup (no liveness)**

Our Witness Encryption Scheme

WE Literature

State of WE:

- From iO [JLS21,JLS22,RVV24](very inefficient)
- From evasive LWE [VWW22,Tsa22]
- GapMDP [BIOW20]
- ADP [BIJ+20]
- Limited constructions for special languages

Takeaway: General WE is **impractical**

SNARK Motivation

Goal:

- Enforce arbitrary NP statements

Approach:

- Use SNARK proof π
- Verify inside WE relation

Reduce general NP to succinct verification

WE Comparison

Setting:

- Binary constraints: $\approx 10^9$ (for VWW22 and BIJ+20)
- $\lambda = 100$

	VWW22 (Evasive LWE)	**BIOW20 (GapMDP)**	**BIJ+20 (ADP)**
Ciphertext Size	10^{77} TB	Unknown	10^{14} TB

WE Comparison

Setting:

- Binary constraints: $\approx 10^9$ (for VWW22 and BIJ+20)
- Arithmetic constraints: ≈ 15000 (Ours)
- $\lambda = 100$

	VWW22 (Evasive LWE)	**BIOW20 (GapMDP)**	**BIJ+20 (ADP)**	**Ours (AADP)**
Ciphertext Size	10^{77} TB	Unknown	10^{14} TB	$10^{2.5}$ $\approx$338 TB

(Arithmetic) Affine Determinant Programs: Structure

An (arithmetic) affine determinant program ((A)ADP) is defined by matrices (gates) affine in x

$$M_1(x), M_2(x), \ldots, M_m(x) \in \mathbb{F}^{k \times k}.$$

For an input $x = (x_0, \ldots, x_n)$, define

$$M(x) = \sum_{i=1}^{m} M_i(x).$$

(Arithmetic) Affine Determinant Programs: Inputs and Acceptance

Input types:

- Affine inputs: $x_0 = 1$
- Projective inputs at infinity: $x_0 = 0$, $x \neq 0$

Acceptance condition:

$$\det M(x) = 0$$

Note: In our construction, we assume $|\mathbb{F}| > 2^{\lambda}$.

Our Arithmetic Constraint System

Input: Projectively safe constraint system $(A, B, C, D) \in \mathbb{F}^{m \times (n+1)}$

Our Arithmetic Constraint System

Input: Projectively safe constraint system $(A, B, C, D) \in \mathbb{F}^{m \times (n+1)}$

with each constraint $i \in [m]$

$$\underbrace{\left(\sum_{k=0}^{n} A_i^k x_k\right)}_{a_i(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} B_i^k x_k\right)}_{b_i(x)} = \underbrace{\left(\sum_{k=0}^{n} C_i^k x_k\right)}_{c_i(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} D_i^k x_k\right)}_{d_i(x)}$$

Our Arithmetic Constraint System

Input: Projectively safe constraint system $(A, B, C, D) \in \mathbb{F}^{m \times (n+1)}$

with each constraint $i \in [m]$

$$\underbrace{\left(\sum_{k=0}^{n} A_i^k x_k\right)}_{a_i(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} B_i^k x_k\right)}_{b_i(x)} = \underbrace{\left(\sum_{k=0}^{n} C_i^k x_k\right)}_{c_i(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} D_i^k x_k\right)}_{d_i(x)}$$

Equivalently,

$$\boxed{a_i(x) b_i(x) = c_i(x) d_i(x)}$$

Our Arithmetic Constraint System

Input: Projectively safe constraint system $(A, B, C, D) \in \mathbb{F}^{m \times (n+1)}$

with each constraint $i \in [m]$

$$\underbrace{\left(\sum_{k=0}^{n} A_i^k x_k\right)}_{a_i(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} B_i^k x_k\right)}_{b_i(x)} = \underbrace{\left(\sum_{k=0}^{n} C_i^k x_k\right)}_{c_i(x)} \cdot \underbrace{\left(\sum_{k=0}^{n} D_i^k x_k\right)}_{d_i(x)}$$

Equivalently,

$$\boxed{a_i(x) b_i(x) = c_i(x) d_i(x)}$$

Valid Solution

An affine input $(x_0, x_1, \ldots, x_n) \in \bar{\mathbb{F}}^{n+1}$ with $x_0 = 1$ such that all constraints are satisfied.

Constraint System $\rightarrow$ AADP

For each constraint i such that $a_i(x)b_i(x) = c_i(x)d_i(x)$

- Set

$$U_i(x) = \begin{pmatrix} a_i(x) & c_i(x) & -\xi_i(x) & 0 \\ d_i(x) & b_i(x) & 0 & \xi_i(x) \\ 0 & 0 & b_i(x) & c_i(x) \\ 0 & 0 & d_i(x) & a_i(x) \end{pmatrix}$$ such that $\xi_i(x)$ is a random linear form.

- Randomly embed: $\boxed{M_i(x) = L_i U_i(x) R_i}$ where $L_i \leftarrow \mathbb{F}^{(2m+1)\times 4}$, $R_i \leftarrow \mathbb{F}^{4\times(2m+1)}$.

$$M(x) = \sum_{i=1}^{m} M_i(x)$$

AADP Rank Property

Key Lemma:
For any input x:

$$\text{rank}(U_i(x)) = \begin{cases} 2 & \text{if constraint } i \text{ holds} \\ 4 & \text{otherwise} \end{cases}$$

Therefore:

$$\text{rank}(M(x)) = \begin{cases} 2m & \text{if } x \text{ satisfies all constraints} \\ 2m+1 & \text{otherwise (full rank)} \end{cases}$$

$$\text{Constraint satisfaction} \implies \text{rank deficiency} \implies \det(M(x)) = 0$$

Witness Encryption from AADP: Encryption

Step 1: Build AADP

$$M(x) \leftarrow \mathsf{AADP.Gen}(A, B, C, D)$$

Step 2: Encode message
Modify one entry of M_0:

$$M_c(x) = M(x) + msg \cdot x_0 \cdot e^T e$$

where

$$e = (0, \ldots, 0, 1)$$

Witness Encryption from AADP: Decryption

Input: $M_c(x)$, witness $s = (1, s_1, \ldots, s_n)$

Evaluate:

$$E := M_c(s)$$

Recover *msg*:

$$\det(E - t \cdot ee^T) = 0$$

Output: the unique solution t

Valid witness $\Rightarrow$ rank drop $\Rightarrow$ recover *msg*

AADP-Based WE Security

Security is heuristic
(no reduction to standard assumptions)

- Detailed analysis of potential attack avenues
- **We invite cryptanalysis:** We provide concrete challenges with rewards for breaking our AADP-based WE scheme.

allocinit.xyz/posts/challenges

Result and Takeaways

What we achieve:

- With PIPE v2, we design a scheme where we can enforce arbitrary spending conditions without changing Bitcoin
- We construct the first plausibly implementable WE for general NP (heuristic)

We are currently working on further optimizations and modifications, with ideas related to structured matrices.

Papers:

- **PIPE v2:**
  https://eprint.iacr.org/2026/186
- **AADP-based WE:**
  https://eprint.iacr.org/2026/175

allocinit.xyz/posts/challenges