Placeholder Verificaion on Bitcoin via FH-MIPE Covenants

Mikhail Komarov
nemo@allocin.it
`[[alloc] init]`

September 7, 2024

Abstract

Zero-Knowledge Proofs verification on the Bitcoin has long been considered unfeasible due to the limitations of the existing Bitcoin Script language. Specifically, the absence of covenants, such as CAT, CTV, $CSFS$ (and a small upper limit on the script size), has prevented the implementation of Merkle tree paths verification required for FRI/LPC-alike commitment schemes commitments verification along with arithmetization definitions computations requiring OP_MUL to be enabled. Despite various proposals (e.g. BIP-420) to re-enable or introduce new covenant opcodes, these changes have not been adopted (yet?), leaving ZKP verification on Bitcoin an unresolved challenge.

This paper proposes an approach to (i) define application-specific covenants on Bitcoin by leveraging FH-MIPE and (ii) enable a hash-based commitment scheme (LPC) proof system (Placeholder[1]) proofs verification on Bitcoin via (i) emulating abscent covenants (e.g. CAT) through the use of Function Hiding Multi-Input Predicate Encryption (FH-MIPE) and (ii) by introducing FH-MIPE predicate-defined Placeholder verification covenant (effectively introducing the Placeholder verification covenant opcode with this). The method proposed involves generating unique keys and signatures that are conditionally valid based on the satisfaction of Placeholder proof conditions. This approach not only overcomes the current limitations of Bitcoin Script but also opens up new possibilities for implementing new kinds of applications on Bitcoin (via application-specific FH-MIPE covenants) alongside with true Bitcoin zkRollups.

1 Introduction

Zero-Knowledge Proofs verification on the Bitcoin has long been considered unfeasible due to the limitations of the existing Bitcoin Script language. Specifically, the absence of covenants, such as CAT, CTV, $CSFS$ (and a small upper limit on the script size), has prevented the implementation of Merkle tree paths verification required for FRI/LPC-alike commitment schemes commitments verification along with arithmetization definitions computations requiring OP_MUL to be enabled. Despite various proposals (e.g. BIP-420) to re-enable or introduce new covenant opcodes, these changes have not been adopted (yet?), leaving ZKP verification on Bitcoin an unresolved challenge.

The obvious solution to this problem is to upgrade Bitcoin's protocol, introduce (or re-introduce) absent opcodes, which leads to the necessity to achieve social consensus, which is a quite complicated process. This means that the next solution in line is to emulate covenants necessary for particular application.

This paper proposes an approach to (i) define application-specific covenants on Bitcoin by leveraging FH-MIPE predicates and (ii) enable a hash-based commitment scheme (LPC) proof system (Placeholder[1]) proofs verification on Bitcoin via (i) emulating abscent covenants (e.g. CAT) through the use of Function Hiding Multi-Input Predicate Encryption (FH-MIPE) and (ii) by introducing FH-MIPE predicate-defined Placeholder verification covenant (effectively introducing the Placeholder verification covenant opcode with this). The method proposed involves generating unique keys and signatures that are conditionally valid based on the satisfaction of Placeholder proof conditions. This approach not only overcomes the current limitations of Bitcoin Script but also opens up new possibilities for implementing new kinds of applications on Bitcoin (via application-specific FH-MIPE covenants) alongside with true Bitcoin zkRollups.

2 Preliminaries

2.1 Covenants

Covenants are restrictions on Bitcoin transactions. They define rules about where and how Bitcoin can be spent, adding a layer of programmability to Bitcoin transactions. Covenants are not currently part of Bitcoin's native functionality and require the community to agree on and implement specific upgrades. Several notable attempts were made recently to introduce controversial covenants opcodes (without any luck):

1. **CheckTemplateVerify (CTV)**: *CTV* is a covenant which only allows the exact next transaction to be executed. It allows a user to commit to a specific transaction by ensuring that only the hash of that transaction matches a predefined value. It was proposed back in 2020 and got assigned BIP-119.

2. **Concatenation (CAT)**: *CAT* operation is crucial for more advanced covenants. It concatenates two data items on the stack, enabling more complex scripts and conditions. For instance, *CAT* can be used to implement covenants that check multiple conditions on the Bitcoin being spent, allowing for a broader range of transaction types and restrictions.

   CAT's significance lies in its ability to support complex Bitcoin Script operations that go beyond simple locking and unlocking scripts. This makes it possible to create more sophisticated covenants that can enforce a wide variety of spending conditions.

   In particular *CAT* can help construct Merkle trees for verifying commitments of hash-based commitment scheme-based proof systems proofs within Bitcoin script natively.

   There were a couple of BIPS introduced to reflect *CAT*: BIP-347 and a more recent (and a popular one) - BIP-420.

2.1.1 Covenants: *CAT*

The absence of concatenation covenant (expressed in an `OP_CAT` opcode) in Bitcoin has made certain operations cumbersome or impossible. `OP_CAT` is essential for efficient Merkle tree operations for verifying commitments. Without `OP_CAT`, simulating its functionality requires cumbersome workarounds that are often impractical. The re-enabling of `OP_CAT` would simplify these operations and make Bitcoin scripts more powerful and flexible.

The re-enabling of `OP_CAT` has been positively received by the Bitcoin community, with several proposals and discussions taking place to bring it back. The draft BIP-420 (`https://github.com/bip420/bip420`) for `OP_CAT` has undergone several iterations, and its implementation in Bitcoin Core is actively being discussed. Re-enabling `OP_CAT` would require a soft fork, which, if successful, would mark a significant enhancement in Bitcoin's scripting capabilities.

The progress towards re-enabling `OP_CAT` is promising, with discussions and reviews happening in the Bitcoin Core PR Review Club and other forums. The potential activation of `OP_CAT` would enable more advanced scripts and applications on Bitcoin, paving the way for Turing-complete applications and improved functionality.

2.1.2 Bitcoin-friendly Proof Systems

To verify zero-knowledge proofs on Bitcoin, the proof system must be efficient and fit within Bitcoin's constraints. A Bitcoin-friendly proof system should minimize the weight units used in the script, stay within stack limits, and utilize existing opcodes like hash functions to reduce computational costs. Hash-based commitment scheme-enabled proof systems, such as Placeholder, are more likely to be compatible with Bitcoin due to their reliance on hash functions and a more ofter usage of smaller prime fields.

By leveraging recursive verification and optimizing for Bitcoin's limitations, it is possible to create proof systems that are both efficient and practical for on-chain verification. The combination of `OP_CAT` and efficient proof systems can enable powerful and flexible covenants, enhancing Bitcoin's programmability and privacy features.

Unfortunately, `OP_CAT` being enabled will take time. This means to unlock Bitcoin native SNARK verification, we need to emulate missing covenants and use a hash-based commitment scheme-enabled proof system to enable the verificaiton without the upgrade.

2.2 Placeholder Proof System

Placeholder [1] is a zero-knowledge succinct non-interactive argument of knowledge based on $\mathcal{P}$lon$\mathcal{K}$-style arithmetization. Placeholder's commitment scheme and types of arithmetization, are replaceable and configurable. Low-level Placeholder circuits can adapt to selected parameters, such as table size, date degree, and lookup options. These properties enable the flexible configuration of Placeholder with trade-offs between circuit parameters, trust assumptions, and efficiency of proof generation. Due to this flexibility, Placeholder can accommodate particular cases, consistently achieving efficient results.

zkSNARK is a type of zero-knowledge proof system that allows one to prove the authenticity of a statement to a verifier without revealing any additional information beyond the statement's validity. The "succinct" and "non-interactive" aspects of zk-SNARKs refer to the fact that the proof is short and does not require any interaction between the prover and verifier beyond the initial setup.

Conceptually, general SNARK construction contains three steps:

1. Translate the problem into a set of polynomials.
2. Commit the polynomials.
3. Prove some relations on the committed polynomials.

Placeholder follows this general SNARK construction and contains two main modules:

1. Arithmetization: Defines the arithmetic representation of the proving statement. Placeholder uses $\mathcal{P}$lon$\mathcal{K}$-based representation with custom gates. The idea was introduced in the TurboPLONK paper [2] and modified later in other proof systems like Halo2 [3] and Kimchi.
2. Commitment Scheme: Placeholder uses the List Polynomial Commitment scheme ([4], [1]) for polynomials obtained from the arithmetization procedure.

Because of the use of List Polynomial Commitment (LPC), Placeholder is positioned as a perfect proof system to be verified on Bitcoin.

But the problem is that even this method doesn't guarantee the so-called "pessimistic" verification on Bitcoin because of commitments-only check being possible. To verify the circuit part, it is required to introduce `OP_MUL` and a larger acceptable script size which requires Bitcoin protocol upgrade. This means a different method should be introduced and it is required to emulate all the following at once:

1. Missing covenants (e.g. CAT)
2. Missing opcodes (e.g. `OP_MUL`)
3. Larger script size

2.3 Functional Encryption

Functional Encryption (FE) is a technique that allows computation over encrypted data to yield decrypted results. It supports restricted secret keys that enable a key holder to learn a specific function of the encrypted data, without learning anything else about the data. For example, given an encrypted program, the secret key may enable the key holder to learn the output of the program on a specific input without learning anything else about the program.

The concept of Functional Encryption was formally studied by Boneh, Sahai, and Waters in [5], who provided precise definitions and discussed its security challenges. The security of FE is non-trivial to define; a natural game-based definition is inadequate for some functionalities, leading to a simulation-based definition, which, while provably secure in the random oracle model, cannot be satisfied in the standard model.

In a functional encryption system, a decryption key allows a user to learn a function of the encrypted data. Briefly, in an FE system for functionality $F(\cdot,\cdot)$ (modeled as a Turing Machine), an authority holding a master secret key can generate a key sk_k that enables the computation of the function $F(k,\cdot)$ on encrypted data. More precisely, using sk_k, the decryptor can compute $F(k,x)$ from an encryption of x. The security of the system guarantees that one cannot learn anything more about x.

An FE scheme for a functionality F is a tuple of four polynomial-time algorithms: *setup*, *keygen*, *enc*, and *dec*, satisfying the following correctness condition for all k in the key space K and x in the plaintext space X:

1. $(pp, mk) \leftarrow setup(1^\lambda)$ (generate a public and master secret key pair)
2. $sk \leftarrow keygen(mk, k)$ (generate secret key for k)
3. $c \leftarrow enc(pp, x)$ (encrypt message x)
4. $y \leftarrow dec(sk, c)$ (use sk to compute $F(k,x)$ from c)

The output y should equal $F(k,x)$ with probability 1.

Standard public-key encryption is a simple example of functional encryption where $K = \{1, \epsilon\}$ and $F(k,x) = x$ if $k = 1$ and $F(k,x) = \text{len}(x)$ if $k = \epsilon$.

2.4 Function-Hiding Inner Product Encryption

Besides many different kinds of FE schemes out there, as reported in [6], in many real scenarios it is important to consider also the privacy of the computed function. The motivation behind this is the fact that a typical workflow of a Bitcoin transaction involves techniques around manipulating pre-signed transactions (or an encrypted signing key) which, if being revealed not in the right moment, would break the whole protocol. If the FE scheme in use does not guarantee any hiding of the function (which is the case for many existing FE schemes), then a hypothetical key sk_f might reveal the predicate functionality contents f, which is undesirable when f itself contains sensitive information (aka a pre-signed transaction or a private key). This has motivated the study of function privacy in FE, see for instance [7, 8, 9].

An IPE scheme is called function-hiding if the keys and ciphertexts reveal no additional information about the related vectors beyond their inner product. The fully function-hiding IPE achieves the most robust IND-based notion of both data and function privacy in the private-key setting in the standard model. The model of full function privacy is sketched hereas described in [9, 10]. Adversaries are allowed to interact with two left-or-right oracles $KeyGen_b(mk,\cdot,\cdot)$ and $Enc_b(mk,\cdot,\cdot)$ for a randomly chosen $b \in \{0,1\}$, where $KeyGen_b$ takes two functions f_0 and f_1 as input and it returns a functional decryption key $sk_{f_b} = KeyGen(mk, f_b)$. The algorithm Enc_b takes two messages x_0 and x_1 as input and it outputs a ciphertext $c_{x_b} = Enc(mk, x_b)$. Adversaries can adaptively interact with oracles for any polynomial (a priori unbounded) number of queries. To exclude inherently inevitable attacks, there is a condition for adversarial queries that all pairs (x_0, x_1) and (f_0, f_1) must satisfy $f_0(x_0) = f_1(x_1)$.

Only two approaches have been proposed for (fully) function-private IPE schemes in the private-key setting. One is to employ the Brakerski-Segev general transformation from (non-function-private) FE schemes for general circuits [8]. The transformation itself is efficient since it simply combines symmetric key encryption with FE in a natural manner. Anyway, this approach requires computationally intensive cryptography tools, such as IND obfuscation, to realise non-function-private FE for general circuits, meaning it may be relatively inefficient overall. The other approach may be more practical. It directly constructs IPE schemes by using the dual-pairing vector spaces (DPVS) introduced by Okamoto and Takashima [11, 12].

In the last few years, there has been a flurry of works on the construction of function-hiding IPE, starting with the work of Bishop, Jain, and Kowalczyk [6]. They propose a function-hiding IPE scheme under the SXDH assumption, which satisfies an adaptive IND-based security definition. But the security model has one limitation: all ciphertext queries x_0, x_1 and all secret key queries y_0, y_1 are restrained by $\langle x_0, y_0\rangle = \langle x_0, y_1\rangle = \langle x_1, y_0\rangle = \langle x_1, y_1\rangle$. In [9], Datta et al. develop a full function-hiding IPE scheme built in the setting of asymmetric bilinear pairing groups of prime order. The security of the scheme is based on the well-studied SXDH assumption where the restriction on adversaries' queries is only $\langle x_0, y_0\rangle = \langle x_1, y_1\rangle$. Here, secret keys and ciphertexts of n-dimensional vectors consist of $4n+8$ group elements. Tomida et al., in [13], construct a more efficient function-hiding IPE scheme than that of [9] under the XDLIN assumption, where secret keys and ciphertexts consist of $2n+5$ group elements. Kim et al., in [14], put forth a fully-secure function-hiding IPE scheme with smaller parameter sizes and run-time complexity than those in [6, 9]. The scheme is proved SIM-based secure in the generic model of bilinear maps. In [15], Zhao et al. present the first SIM-based secure secret-key IPE scheme under the SXDH assumption in the standard model. The authors claim that the scheme can tolerate an unbounded number of ciphertext queries and adaptive key queries. Zhao et al. in [16] propose a new version of the scheme, which is an improvement in terms of computational and storage complexity. In a very recent work [17], Liu et al. present a more efficient and flexible private-key IPE scheme with SIM-based security. To ensure correctness, the scheme requires that the computation of inner products is within a polynomial range, where the discrete logarithm of $g^{\langle x,y\rangle}$ can be found in polynomial time. In [17], the authors compare their proposed IPE scheme with those in [9, 13, 15, 16]. The performance of this scheme appears superior in both storage complexity and computation complexity. Moreover, secret keys and ciphertexts are shorter.

Although most aforementioned IPE schemes are efficient and based on standard assumptions, they all have one inconvenient property: they are bounded. The maximum length of vectors has to be fixed at the beginning, and afterward, one cannot handle vectors whose lengths exceed it. This could be inconvenient when it is hard to predict which data will be encrypted in the setup phase. One may think to solve the problem by setting the maximum length to a large value. However, the size of parameters expands at least linearly with the fixed maximum length, and such a solution incurs an unnecessary efficiency loss. In the context of IP-PE and ABE, there exist unbounded schemes (see, for instance, [18, 19, 20, 6]), whose public parameters do not impose a limit on the maximum length of vectors or number of attributes used in the scheme. In [13], Tomida and Takashima construct two concrete unbounded IPE schemes based on the standard SXDH assumption, both secure in the standard model: the first is a private-key IPE with fully function hiding, the second scheme is a public-key IPE with adaptive security. Concurrently and independently, in [21], Dufour-Sans and Pointcheval describe an unbounded IPE system supporting identity access control with succinct keys. Their construction is proved selectively IND-secure in the random oracle model based on the standard DBDH assumption. In [13], it is shown a comparison, in terms of efficiency, among private-key schemes that are fully function hiding [20, 19, 15] and public-key schemes with adaptive security in the standard model [22].

3 Proposal

As the end goal is to use the FH-MIPE covenants to verify Placeholder, let's recall the Placeholder verification procedures and define necessary primitives as FH-MIPE covenants for it:

Algorithm 1 LPC.EvalVerify

Input:
proof $\mathcal{P}$,
evaluation points $\{\xi^{(k)}\}_{k=0}^{l-1}$,
roots of Merkle trees $\{\mathsf{root}_k\}_{k=0}^{K-1}$,
transcript

Output: verification result = true/false

$\{\boldsymbol{z}^{(0)}, \ldots, \boldsymbol{z}^{(l-1)}, \pi\} = \mathtt{parse}(\mathcal{P})$
Interpolate polynomials $U_k(X) = \mathsf{lagrange\_interpolation}(\{\xi_j^{(k)}, \boldsymbol{z}_j^{(k)}\})$ for $0 \leq k < l, 0 \leq j < |\xi^{(k)}|$
Compute $V_k(X) = \prod_{j=0}^{|\xi^{(k)}|-1}(X - \xi_j^{(k)})$
if $\mathsf{FRI.Verify}(\pi, \{\mathsf{root}_k\}_{k=0}^{K}, \{U_k(X)\}_{k=0}^{l-1}, \{V_k(X)\}_{k=0}^{l-1}, \mathsf{transcript}) = \mathsf{false}$ **then return** false
return true

Algorithm 2 FRI.Verify

Input: FRI proof π, Merkle roots $\{\mathsf{T\_root}_k\}_{k=0}^{K-1}$, $\{U_k(X)\}_{k=0}^{l-1}$, $\{V_k(X)\}_{k=0}^{l-1}$, transcript
Output: verification result = true/false

$\{\mathsf{fri\_root}_0, \dots, \mathsf{fri\_root}_{\mathsf{steps_{FRI}}-1}, \pi^{(0)}, \dots \pi^{(r_q-1)}, \mathsf{final\_polynomial}\} = \mathtt{parse}(\pi)$
$\mathcal{D}^{(0)} = \mathcal{D}, \quad \mathcal{D}^{(i+1)} = q_{r_i}\left(\mathcal{D}^{(i)}\right)$, for $i = 0, \dots, \mathsf{steps_{FRI}} - 2$
for all $k = 0, \dots, L-1$ **do**
    transcript.$append(\mathsf{T\_root}_k)$
end for
$\tau :=$ transcript.$challenge(\mathbb{F})$
$t := 0$
for all $i := 0, \dots, \mathsf{steps_{FRI}} - 1$ **do**
    transcript.$append(\mathsf{fri\_root}_i)$
    for all $step := 0, \dots, r_i - 1$ **do**
        $\alpha_t :=$ transcript.$challenge(\mathbb{F})$
        $t := t + 1$
    end for
end for
for all $query = 0, \dots, r_q - 1$ **do**
    $\{\pi^*, \pi_0, \dots, \pi_{\mathsf{steps_{FRI}}-1}\} = \mathtt{parse}(\pi^{(round)})$
    $x^{(0)} =$ transcript.$challenge(\mathcal{D}_0)$
    $x^{(i+1)} = q_{r_i}(x^{(i)}),\ i = 0, \dots, \mathsf{steps_{FRI}} - 1$
    Construct cosets $S^{(i)} = \{s \in \mathcal{D}^{(i)} \mid q_{r_i}(s) = x^{(i+1)}\}$, for $i = 0, \dots, \mathsf{steps_{FRI}} - 1$ ▷ $|S^{(i)}| = m^{r_i}$
    `Initial proof check`
    t := 0;
    for all $k := 0, \dots, K-1$ **do**
        if $\pi^*.\mathsf{auth}_k.\mathsf{root} \neq \mathsf{T\_root}_k$ **then return** $false$
        if $\mathsf{MT.Validate}(\pi^*.\mathsf{auth}_k, \{\pi^*.\mathsf{val}^{(t)}, \dots, \pi^*.\mathsf{val}^{(t+l_k-1)}\}) = false$ **then return** $false$
        $t := t + l_k$
    end for
    Compute values of combined polynomial Q values val from $\pi_k.\mathsf{val}$

$$\mathsf{val} = \left\{ \prod_{k=0}^{l-1} \tau^{l-k-1} \frac{\pi^*.\mathsf{val}_s^{(k)} - U_k(s)}{V_k(s)} \right\}_{s \in S^{(0)}}$$

    `Round proofs check`
    $t := 0, S := S^{(0)}$
    for all $i := 0, \dots, \mathsf{steps_{FRI}} - 1$ **do**
        if $\pi_i.\mathsf{auth}.\mathsf{root} \neq \pi.\mathsf{fri\_root}_i$ **then return** $false$
        if $\mathsf{MT.Validate}(\pi_i.\mathsf{auth}, \mathsf{val}) = false$ **then return** $false$
        for all $step := 0, \dots, r_i - 1$ **do**
            $S_{\mathsf{next}} := \{q(s)\}_{s \in S}$
            $\mathsf{interpolant}_s := \mathsf{lagrange\_interpolation}(\{s_j, \mathsf{val}_{s_j}\}_{q(s_j)=s})(\alpha_t)$ for $s \in S_{\mathsf{next}}$
            $t := t + 1, S := S_{\mathsf{next}}, \mathsf{val} := \{\mathsf{interpolant}_s\}_{s \in S_{\mathsf{next}}}$ ▷ $|S_{\mathsf{next}}| = r_i - step - 1$
        end for
        if $\mathsf{val} \neq \pi_i.\boldsymbol{y}_{(x^{(i+1)})}$ **then return** $false$ ▷ $|\mathsf{val}| = 1$
        $\mathsf{val} := \pi_i.\boldsymbol{y}$
    end for
    if $\mathsf{final\_polynomial}(x^{(\mathsf{steps_{FRI}})}) \neq \mathsf{val}$ **then return** $false$
end for
return true

And, specifically, the complete verification procedure, excluding PermArgumentVerify and LookupArgumentVerify (as those would simply require `OP_MUL` introduction) sub-procedures, looks as follows:

Algorithm 3 Verify

Input: $\pi_{\mathsf{Placeholder}}, \mathsf{preprocessed\_data}, \mathsf{transcript}$

Output: true/false

Parse proof $\pi_{\mathsf{Placeholder}}$ into:

$\pi_{\mathsf{comm}} = \{\ \overline{\mathsf{variable}},\ \overline{\mathsf{V\_polynomials}},\ \overline{\mathsf{lookup^{perm}}},\ \overline{\mathsf{quotient}},\ \overline{\mathsf{fixed}}\}$

$\pi_{\mathtt{eval}}$ is evaluation proofs for

$\mathtt{polynomial\_evaluations} = \{$

$\quad w_i(y), w_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N_{wt}} - 1, \quad s_i(y), s_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N_{pi}} - 1$

$\quad\quad$ for all corresponding $d \in \mathbf{o}$,

$\quad V^\sigma(y), V^\sigma(\zeta \cdot y), a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y), V_L(y), V_L(\zeta \cdot y),$

$\quad \{T_i(y)\}, i = 0, \dots, N_T - 1$

$\quad c_i(y), c_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N_{cn}} - 1, \quad l_i(y), l_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N_{lk}} - 1, \quad q_i(y), q_i(\zeta^d \cdot y), i = 0, \dots, \mathsf{N_{sl}} - 1$

$\quad\quad$ for all corresponding $d \in \mathbf{o}$,

$\quad q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y), L_0(y)\}$

Verify Permutation Argument:

Denote polynomials included in permutation argument as $f_0, \dots, f_{\mathsf{N_{perm}}-1}$

Get values $\{f_i(y)\}, \{S_i^e(y)\}, \{S_i^\sigma(y)\}, V^\sigma(y), V^\sigma(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y)$ from $\pi_{\mathsf{Placeholder}}$

Calculate

$$\begin{aligned} F_0(y), F_1(y), F_2(y) = \mathsf{PermArgumentVerify}(&y, V^\sigma(y), V^\sigma(\zeta \cdot y), \{f_i(y)\}, \\ &\{S_i^e(y)\}, \{S_i^\sigma(y)\}, q^{\mathrm{pad}}(y), q^{\mathrm{last}}(y), L_0(y),\ \mathsf{transcript}) \end{aligned}$$

Verify Lookup Argument:

Denote polynomials included in lookup argument as $a_0, \dots, a_{\mathsf{N_{lk}}-1}$

Get values $\{a_i(y)\}, \{l_i(y)\}, a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y), V_L(y), V_L(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y)$ from $\pi_{\mathsf{Placeholder}}$

Calculate:

$$\begin{aligned} F_3(y), F_4(y), F_5(y), F_6(y), F_7(y) = \mathsf{LookupArgumentVerify}(& \\ &\overline{\mathsf{lookup^{perm}}}, \{a_i(y)\}, \{l_i(y)\}, \\ &a^{\mathrm{perm}}(y), a^{\mathrm{perm}}(\zeta^{-1} \cdot y), l^{\mathrm{perm}}(y), \\ &V_L(y), V_L(\zeta \cdot y), L_0(y), q^{\mathrm{last}}(y), q^{\mathrm{pad}}(y),\ \mathsf{transcript}) \end{aligned}$$

$\mathsf{transcript}.append(\overline{\mathsf{V\_polynomials}})$

Verify Basic Constraints:

For $i = 0, \dots, \mathsf{N_{sl}} - 1$

$$g_i(X) = q_i(X) \cdot (\theta^{k_i - 1 + \nu_i} \mathcal{C}_{i_0}(X) + \dots + \theta^{\nu_i} \mathcal{C}_{i_{k-1}}(X))$$

Calculate a constraints-related numerator of the quotient polynomial $F_8(y) = \sum\limits_{0 \le i < \mathsf{N_{sl}}} (g_i(y))$

Quotient polynomial check:

if $\sum\limits_{i=0}^{8} \alpha_i F_i(y) \neq Z(y)T(y)$ **then return** $false$

Get challenges $\{\alpha_i \in \mathbb{F}\}_{i=0}^8,\ \theta \in \mathbb{F},\ y \in \mathbb{F} \setminus H$ from transcript

$\mathsf{transcript}.append(\overline{\mathsf{quotient}})$

Evaluation proof check

if LPC.EvalVerify($\mathtt{polynomials\_evaluations}, \pi_{comm}, \mathsf{transcript}$) $= false$ **then return** $false$

This means, to achieve direct verification of Placeholder proofs on Bitcoin practically two approaches can be considered:

1. Implement absent trivial covenants (e.g. CAT) within FH-MIPE predicates and to implement LPC/FRI commitments verification using FH-MIPE covenant signing as an equivalent of using `OP_CAT` itself and to eventually introduce `OP_MUL` as an additional opcode.

   This approach would lead to the commitments being verified, but not the gates part (aka the circuit definition). This means the verification would still be some considered some kind of "optimistic" 'cause only commitments will be checked (i.e. $LPCVer$ and $FRIVer$) (no constraint-related checks will be made).

2. Implement the whole Placeholder verification procedure within the FH-MIPE predicate using some FH-MIPE scheme enabling circuits-based predicates.

Considering the complexity of the verifier, both of these verification approaches would require to pick a functional encryption scheme, which would allow predicates to be defined with at least polynomial-size circuits (such schemes can be found among predicate encryption schemes) as $F(k, x)$ would need to define, among other primitives, the ECDSA signing algorithm as the encrypted transaction needs to be signed at the last stage of a predicate.

Some results have been obtained for PE schemes for circuits. In [23], Gorbunov et al. present a leveled PE scheme for all circuits (Boolean predicates of bounded depth), with succinct ciphertexts and secret keys independent of the size of the circuit. The achieved privacy notion is a selective SIM-based variant of attribute-hiding, assuming the hardness of the subexponential LWE problem. Recall that the strong variant notion (full attribute-hiding) is impossible to realize for many messages [24].

Some results have also been obtained for general-purpose FE, in which functions associated with secret keys can be any arbitrary circuits. In [25], Garg et al. give constructions for IND obfuscation (based on multilinear maps), and they use it to construct FE for all polynomial-size circuits. In [26], there are constructions directly based on multilinear maps. Further analysis can be found in [24] where Lin and Tessaro reduce the degree of the required multilinear map to 3.

It is also required to take into consideration that an obfuscated/encrypted signing entity private key is necessary to be baked into the covenant, which means the choice of a particular FE scheme would depend on if it supposes the presence of IO or function-hiding.

3.1 Function-hiding Inner Product Encryption

As mentioned in , it is required to have a transaction signing key encrypted within the covenant. This means the FE scheme should be a function-hiding one. If the FE scheme in use does not guarantee any hiding of the function (which is the case for many existing FE schemes), then the key sk_f might reveal f, which is undesirable when f itself contains sensitive information (Secp256k1 signing key). As discussed in 2.4, the scheme of a choice should be either:

1. Function-hiding one capable of having predicates deifned with circuits.

2. Multi-input one with indistinguishable obfuscator being present, again, being capable of defining predicates with circuits.

3.2 Functiona Encryption Schemes for Circuits

Functional Encryption schemes for circuits have been studied extensively due to their powerful expressive abilities. Such schemes allow computation of arbitrary circuits over encrypted data, enabling a wide range of applications. The main goal in this area is to construct FE schemes that support circuits of arbitrary polynomial size while maintaining security and efficiency.

One of the early works in this area was by Gorbunov, Vaikuntanathan, and Wee in 2015, who proposed a general-purpose FE scheme for circuits based on learning with errors (LWE) [27]. This scheme supports the evaluation of any polynomial-size circuit on encrypted data while ensuring that the only information leaked is the output of the circuit.

In 2016, Garg, Gentry, Halevi, and Zhandry introduced a more efficient construction based on the notion of multi-input functional encryption (MIFE) [28]. Their scheme supports the evaluation of circuits on inputs encrypted under different keys, which broadens the applicability of FE to scenarios involving multiple data sources.

Recently, there has been a significant focus on improving the efficiency of FE schemes for circuits. A notable contribution in this direction was made by Ananth, Brakerski, and Vaikuntanathan in 2017, who proposed an FE scheme for circuits with better security guarantees and reduced ciphertext size [29].

3.3 Functional Encryption Scheme of Choice

To satisfy all the requirements of the concept though, a modification of the scheme described in [30] is proposed to be used. The scheme proposed to be used is an inner-product based, designed to be instantinated under pretty widespread SXDH assumptions, but, unfortunately, it is not generic enough as it doesn't support general-purpose circuits, straightforward introduction of which, as discussed in 2.4, would require to introduce indistinguishable obfuscation in a way similar to Brakerski-Segev transformation, which would make the scheme impractical.

To bypass the necessity for an IO, it is proposed to introduce a DPVS-specific arithmetization, which, because of variable vector length support, is doable. Considering that usually in a DPVS-based functional encryption scheme, computations are performed over dual vector spaces using bilinear maps. The vector spaces represent both the encrypted data and the functions, while the bilinear pairing allows us to compute a result without revealing the inputs or the function. Let's consider an example.

3.3.1 Vector Spaces and Pairing Operation

Define two vector spaces: $\mathbb{V}_1, \mathbb{V}_2 \subseteq \mathbb{F}_q^n$ where $\mathbb{F}_q$ is a finite field and n is the dimension of the vectors.

The bilinear pairing is: $e : \mathbb{V}_1 \times \mathbb{V}_2 \to \mathbb{G}_T$ and satisfies: $e(a \cdot \mathbf{v}_1, \mathbf{v}_2) = e(\mathbf{v}_1, a \cdot \mathbf{v}_2) \quad \forall a \in \mathbb{F}_q$ where $\mathbf{v}_1 \in \mathbb{V}_1$ and $\mathbf{v}_2 \in \mathbb{V}_2$.

3.3.2 Arithmetization of Circuit Gates

Let the circuit consist of addition, multiplication, and comparison gates. For each gate g, define input and function vectors $\mathbf{g}_{\text{input}}$ and $\mathbf{g}_{\text{function}}$.

Addition Gate For $x_3 = x_1 + x_2$, encode:

$$\mathbf{v}_{\text{add}} = [x_1, x_2, -x_3]$$

and the function vector is:

$$\mathbf{f}_{\text{add}} = [1, 1, 1]$$

with the inner product condition:

$$\mathbf{v}_{\text{add}} \cdot \mathbf{f}_{\text{add}} = 0 \quad \text{(verifying the addition)}$$

Multiplication Gate For $x_3 = x_1 \times x_2$, encode:

$$\mathbf{v}_{\text{mult}} = [x_1, x_2, -x_3]$$

and the function vector is:

$$\mathbf{f}_{\text{mult}} = [1, 1, 1]$$

with the inner product condition:

$$\mathbf{v}_{\text{mult}} \cdot \mathbf{f}_{\text{mult}} = 0 \quad \text{(verifying the multiplication)}$$

Comparison Gate For $x_3 = \max(x_1, x_2)$, a conditional comparison function is used. The vector encoding would involve a comparison function, which could be polynomially encoded based on field properties.

3.3.3 Final Result

After performing the operations through bilinear pairings over the vector spaces, the final output is computed as:

$$\text{Output} = e(\mathbf{x}_{\text{result}}, \mathbf{y}_{\text{function}})$$

where $\mathbf{x}_{\text{result}}$ is the final data vector and $\mathbf{y}_{\text{function}}$ is the function vector.

3.4 Functional Encryption Covenants

As an example of a FH-MIPE covenant, for the sake of the Placeholder verification goal, CAT was chosen to be expressed via function-hiding functional encryption. This would require to define, besides ECDSA signing procedure, an implementation of a circuit-expressed `OP_CAT` as a predicate.

3.4.1 Setup

A setup of the overall scheme goes as follows:

Algorithm 4 *CovenantSetup*

Input:
FE/PE Master Keys: $(m, M) \in K = \{0, 1\}_* \cup \{\epsilon\}$,
ECDSA Keys: $(sk, pk) \in \mathcal{F}_p : y^2 = x^3 + 7 \ (mod\ p) \ \forall\ p = 2^{256} - 2^{32} - 977$,
Output: Ciphertext C_f

$m :\ C_p \leftarrow Encrypt(M, p)$
$PrivateKey(sk, \alpha, \beta) = Add(m, Concat(\alpha, \beta))$
$Publickey(pk\alpha, \beta) = TweakAdd(pk, Concat(\alpha, \beta))$
Functionality for $F(p, \alpha, \beta, tx) = ECDSASign(PrivateKey(p,\ \alpha, \beta),\ tx)$
Compute the public key: $u \leftarrow PublicKey(p,\ \alpha, \beta)$
Encrypt the function: $C_f = EncryptFunc(m,\ F)$
Trusted party is supposed to erase m and p and the signing now can be arranged as follows:

$$\sigma \leftarrow C_f(C_p,\ Encrypt(M, tx))$$

From now on, σ can be used as a signature of u over tx. If m and p were erased, C_f can only sign in case CAT is "executed" correctly.

3.5 Placeholder Verification as a Covenant

As the circuit which defines predicate within the chosen FH-MIPE scheme is possible to be arranged to represent a Turing-complete computation, a much more complicated covenant can be constructed. For example, the signing procedure can be prefixed/conditioned with the Placeholder verification procedure. This means that it is required to define the predicate functionality as follows at first:

Algorithm 5 *VerifySignPredicate*

Input:
ECDSA Private Key: sk
Public Input: $input$
Transaction: tx
Placeholder zkProof: $proof$
Output: Signature S

if $\mathsf{Verify}(proof, input) = \mathsf{true}$ **then return** $ECDSASign(sk,\ tx)$
return true

Algorithm 6 *CovenantSetup*

Input:
FE/PE master keys: $(m, M) \in K = \{0,1\}_* \cup \{\epsilon\}$,
ECDSA Keys: $(sk, pk) \in \mathcal{F}_p : y^2 = x^3 + 7\ (mod\ p)\ \forall\ p = 2^{256} - 2^{32} - 977$,
Public Input: *input*
Transaction: *tx*

Output: Functionality $C_f : K \times X \Leftrightarrow \{0,1\}_*$

$m :\ C_p \leftarrow Encrypt(M, sk)$
$PrivateKey(sk, proof, input) = Add(m, \mathsf{Verify}(proof, input))$
$Publickey(pk, proof, input) = TweakAdd(pk, \mathsf{Verify}(proof, input))$
$F(sk, tx, proof) = VerifySignPredicate(sk, input, tx, proof)$
Compute the public key: $u \leftarrow PublicKey(pk,\ proof, input)$
Encrypt the function: $C_f = EncryptFunction(m,\ F)$
Protocol initializer erases m and sk.

$$\sigma \leftarrow C_f(C_p,\ Encrypt(M, tx))$$

From now on, σ can be used as a signature of u over tx. If m and sk were erased, C_f can only sign in case the covenant functionality F is "executed" correctly.

3.5.1 Usage

Architecturally speaking, such a FH-MIPE covenant is a binary which contains an ECDSA signing key hidden within the predicate, which if the covenant is not satisfied, would simply not sign the withdrawal transaction.

This means sending to a FH-MIPE-restricted account would look like this:

1. Retrieve an encrypted opcode binary with covenant e and private key sk.
2. Select the covenant instance parameters i if needed.
3. Tweak sk by e_i.
4. Send funds to e_i.

Accordingly, redeeming from the account would look as:

1. Prepare the state transition with a specific tx and collect all necessary verification data.
2. Pass the data and tx to the encrypted opcode binary.
3. Submit a transaction with a signature generated by a covenant binary.

4 Conclusion

This paper proves that a function-hiding multi-input functional encryption is enough to introduce covenants for Bitcoin without any necessity for a protocol upgrade and express complex application-specific via those. As an example, to verify a hash-based commitment scheme-based proof system (Placeholder) proof, conditionally signing Bitcoin transaction with ECDSA signature.

Having covenants implemented this via FH-MIPE means:

1. Computation integrity verification happening outside of Bitcoin Script, while still being checked by it.
2. Fixed witness size 64 bytes signature, script size 32 bytes private key.
3. Composability of a signature with other opcodes in a script with other conditions.

4. Non-interactive protocol definition means no infrastructure is necessary to maintain liveness, which improves security assumptions to native Bitcoin L1-level ones.

5. Setup procedure, due to the nature of currently existing SA-FE schemes, supposes the initial covenant deployer to behave honest. This can be improved with traditional MPC-based trusted setup, so only one participant out of N generating master key would be required to be honest.

6. Trivial covenants remain bytewise and verification wise more efficient.

In the same time, having Placeholder verification implemented as an FH-MIPE covenant brings to the table:

1. Full proof verification, which means not only commitments are being verified (like it is with an `OP_CAT` case), but the verification also includes a circuit (gates) part into it which means security assumptions improve to the strongest ones (no challenger oracle is required to be present within the verification protocol).

2. Single-round ZKP verification on Bitcoin without any trust assumptions other than a trusted setup assumption.

References

1. Cherniaeva A., Shirobokov I., Komarov M. Placeholder. https://cms.nil.foundation/uploads/placeholder_reasearch_2023_c5cd673c14.pdf. 2024.

2. Gabizon A., Williamson Z. J. Proposal: The Turbo-PLONK program syntax for specifying SNARK programs. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf.

3. PLONKish Arithmetization - The halo2 book. https://zcash.github.io/halo2/concepts/arithmetization.html.

4. Kattis A., Panarin K., Vlasov A. RedShift: Transparent SNARKs from List Polynomial Commitment IOPs. Cryptology ePrint Archive, Report 2019/1400. 2019. https://ia.cr/2019/1400.

5. Boneh D., Sahai A., Waters B. Functional Encryption: Definitions and Challenges. Cryptology ePrint Archive, Paper 2010/543. 2010. https://eprint.iacr.org/2010/543. URL: https://eprint.iacr.org/2010/543.

6. Bishop A., Jain A., Kowalczyk L. Function-hiding inner product encryption // International Conference on the Theory and Application of Cryptology and Information Security / Springer. 2015. P. 470–491.

7. Boneh D., Raghunathan A., Segev G. Function-private identity-based encryption: Hiding the function in functional encryption // Annual Cryptology Conference / Springer. 2013. P. 461–478.

8. Brakerski Z., Segev G. Function-private functional encryption in the private-key setting // Journal of Cryptology. 2018. Vol. 31, no. 1. P. 202–225.

9. Datta P., Dutta R., Mukhopadhyay S. Functional encryption for inner product with full function privacy // Public-Key Cryptography–PKC 2016 / Springer. 2016. P. 164–195.

10. Datta P., Dutta R., Mukhopadhyay S. Strongly full-hiding inner product encryption // Theoretical Computer Science. 2017. Vol. 667. P. 16–50.

11. Okamoto T., Takashima K. Homomorphic encryption and signatures from vector decomposition // International Conference on Pairing-Based Cryptography / Springer. 2008. P. 57–74.

12. Okamoto T., Takashima K. Hierarchical predicate encryption for inner-products // International Conference on the Theory and Application of Cryptology and Information Security / Springer. 2009. P. 214–231.

13. Tomida K., Takashima K. Function-hiding inner product encryption with flexible public parameters // International Conference on the Theory and Application of Cryptology and Information Security / Springer. 2017. P. 312–341.

14. Kim S., Kim J., Seo J.-H. A new approach to practical function-private inner product encryption // Theoretical Computer Science. 2019. Vol. 783. P. 22–40.

15. Simulation-based security of function-hiding inner product encryption / Q. Zhao, Q. Zeng, X. Liu et al. // Science China Information Sciences. 2018. Vol. 61, no. 4. P. 1–3.

16. Zhao Q., Zeng Q., Liu X. Improved construction for inner product functional encryption // Security and Communication Networks. 2018. Vol. 2018.

17. Liu J. et al. A more efficient and flexible private-key IPE scheme with SIM-based security // International Conference on Security and Cryptography for Networks / Springer. 2020. P. 544–562.

18. Decentralizing inner-product functional encryption / M. Abdalla, D. Catalano, D. Fiore et al. // Annual International Conference on the Theory and Applications of Cryptographic Techniques / Springer. 2019. P. 128–157.

19. Chen K., Wee H., Wichs D. Efficient and strongly secure functional encryption: Point and inner product functionalities // Journal of Cryptology. 2018. Vol. 31, no. 1. P. 208–250.

20. Simple functional encryption schemes for inner products / M. Abdalla, F. Bourse, A. Caro et al. // International Conference on the Theory and Application of Cryptology and Information Security / Springer. 2019. P. 30–60.

21. Dufour-Sans Q., Pointcheval D. Unbounded inner-product functional encryption with succinct keys and its application to attribute-based encryption // International Conference on Security and Cryptography for Networks / Springer. 2019. P. 308–329.

22. Abdalla M., Catalano D., Fiore D. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings // Annual International Conference on the Theory and Applications of Cryptographic Techniques / Springer. 2015. P. 597–627.

23. Gorbunov S., Vaikuntanathan V., Wee H. Leveled fully homomorphic signatures from standard lattices // Cryptology ePrint Archive. 2013. https://eprint.iacr.org/2013/142.

24. Sahai A., Seyalioglu H. Worry-Free Encryption: Functional Encryption with Public Keys // Cryptology ePrint Archive. 2010. https://eprint.iacr.org/2010/229.

25. Candidate indistinguishability obfuscation and functional encryption for all circuits / S. Garg, C. Gentry, S. Halevi et al. // 54th Annual Symposium on Foundations of Computer Science (FOCS). 2013. P. 40–49.

26. Lin H., Tessaro S. Indistinguishability Obfuscation from Trilinear Maps and Block-Wise Local PRGs // Advances in Cryptology – CRYPTO 2017. Vol. 10401 of *Lecture Notes in Computer Science*. 2017. P. 530–569.

27. Gorbunov S., Vaikuntanathan V., Wee H. Predicate Encryption for Circuits from LWE // Annual Cryptology Conference / Springer. 2015. P. 503–523.

28. Functional Encryption Without Obfuscation / S. Garg, C. Gentry, S. Halevi et al. // Theory of Cryptography Conference / Springer. 2016. P. 480–511.

29. Ananth P., Brakerski Z., Vaikuntanathan V. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps // Annual International Conference on the Theory and Applications of Cryptographic Techniques / Springer. 2017. P. 152–181.

30. Behera B. C., Ramanna S. C. Multi-Input Functional Encryption for Unbounded Inner Products. Cryptology ePrint Archive, Paper 2024/919. 2024. URL: https://eprint.iacr.org/2024/919.

31. Gabizon A., Williamson Z. J., Ciobotaru O. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953. 2019. https://ia.cr/2019/953.

32. Fast Reed-Solomon interactive oracle proofs of proximity / E. Ben-Sasson, I. Bentov, Y. Horesh et al. // 45th international colloquium on automata, languages, and programming (icalp 2018) / Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

33. Gabizon A., Williamson Z. J. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315. 2020. https://ia.cr/2020/315.

34. Lookup argument - The halo2 book. https://zcash.github.io/halo2/design/proving-system/lookup.html.

35. Chiesa A., Ojha D., Spooner N. Fractal: Post-Quantum and Transparent Recursive Proofs from Holography. Cryptology ePrint Archive, Report 2019/1076. 2019. https://ia.cr/2019/1076.

36. Function-hiding inner product encryption is practical / S. Kim, K. Lewi, A. Mandal et al. // International Conference on Security and Cryptography for Networks / Springer. 2018. P. 544–562.

37. Shor P. W. Algorithms for quantum computation: Discrete logarithms and factoring // Proceedings of the 35th Annual Symposium on Foundations of Computer Science. 1994. P. 124–134.

38. Debnath S. R., Patranabis D. MQ-based functional encryption for inner products // International Conference on the Theory and Application of Cryptology and Information Security / Springer. 2020. P. 171–202.

39. Cocks C. An identity-based encryption scheme based on quadratic residues // Proceedings of the 8th IMA International Conference on Cryptography and Coding. 2001. P. 360–363.

40. Datta P., Okamoto T., Tomida J. Full-Hiding (Unbounded) Multi-Input Inner Product Functional Encryption from the k-Linear Assumption. 2018.

41. Agrawal S., Goyal R., Tomida J. Multi-Input Quadratic Functional Encryption: Stronger Security, Broader Functionality // Theory of Cryptography / Ed. by E. Kiltz, V. Vaikuntanathan. Cham: Springer Nature Switzerland, 2022. P. 711–740.

42. Datta P., Okamoto T., Takashima K. Efficient Attribute-Based Signatures for Unbounded Arithmetic Branching Programs. Cryptology ePrint Archive, Paper 2019/363. 2019. URL: https://eprint.iacr.org/2019/363.

43. Tomida J., Abe M., Okamoto T. Efficient Functional Encryption for Inner-Product Values with Full-Hiding Security // Information Security / Ed. by M. Bishop, A. C. A. Nascimento. Cham: Springer International Publishing, 2016. P. 408–425.