Glock: Garbled Locks for Bitcoin

Liam Eagen[1]

[1]Alpen Labs

August 15, 2025

Abstract

Bitcoin [Nak09] is a decentralized, permissionless network for digital payments. Bitcoin also supports a limited set of smart contracts, which restrict how bitcoin can be spent, through bitcoin script. In order to support more expressive scripting functionality, Robin Linus introduced the BitVM family of protocols [Lin23a, LAZ+24]. These implement a weaker form of "optimistic" smart contracts, and for the first time allowed bitcoin to verify arbitrary computation. BitVM allows a challenger to publish a "fraud proof" that the computation was carried out *incorrectly* which can be verified on chain, even when the entire computation cannot. Jermey Rubin introduced [1] an alternative optimistic smart contract protocol called Delbrag. This protocol uses Garbled Circuits (GC) to replace the BitVM fraud proof with by simply *revealing a secert.* He also introduced the Grug technique for malicious security.

We introduce a new formalization of GC based optimistic techniques called Garbled Locks or Glocks. Much like Delbrag, we use the GC to leak a secret and produce a signature as a fraud proof. We further propose the first concretely practical construction that does not require Grug. Like BitVM2 and Delbrag, Glock25 reduces verification of arbitrary bounded computation to verification of a SNARK. In Glock25, we use a designated verifier version of a modified of the SNARK Pari [DMS24] with smaller proof size. We make Glock25 maliciously secure using a combination of Cut-and-Choose, Verifiable Secret Sharing (VSS), and Adaptor Signatures. These techniques reduce the communication, computational, and on-chain complexity of the protocol compared to other approaches to construct a Glock, e.g. based on Groth16.

1 Introduction

Bitcoin is a peer-to-peer decentralized payment systems. Anyone can connect to the bitcoin network and receive and send payments without a trusted intermediary. Bitcoin uses a novel mechanism called Proof of Work (PoW) to establish consensus over a sequence of "blocks" of transactions. This sequence is called a blockchain, and since the

[1]Our work was largely carried out independently of and prior to the publication of Delbrag

creation of bitcoin many other blockchain protocols have been created. These introduce features such as privacy, as in Monero and Zcash, as well as extensible smart contracts, most famously in Ethereum. Bitcoin is not capable of supporting these features directly without modifications to the protocol, which are rare and contentious. The absence of these features, in particular related to privacy and scalability, severely limits users' ability to use bitcoin.

Privacy The lack of privacy means users identities and on chain activities are linkable, exposing users to risk of theft and seizure. This also makes the chain unattractive for commerce, as businesses unavoidably leak information about their customers and suppliers. The lack of privacy coupled with a global ledger also breaks the fungibility of bitcoin, i.e. coins can be discriminated based on their transaction history, which makes bitcoin more centralized and permissioned.

Scalability The lack of scalability means that bitcoin cannot be used by everyone in the world in a permissionless way. Instead users are forced to rely on centralized third parties to interact with the system. These third parties are thus empowered to place additional restrictions on what users can do with their bitcoin by withholding services. Where other blockchains use extensible smart contracts to implement these features in a trust-minimized way, bitcoin cannot.

Network Utilization The effects of enabling scalability technologies on bitcoin like rollups will have complex effects on the dynamics of the network. For example, rollups will likely increase demand for blockspace and thereby increase the prevailing fee rate and miner revenue. Since Glocks consist of verifying a small number of signatures on chain, this increase will not be accompanied by increased computational burden on miners. This will increase the security budget and may help the long term security of the network and miner decentralization. However, the effects could be harmful if they introduce MEV(il) [Cor24] and more research is needed to understand them in full.

1.1 Rollups

There have long been efforts in the bitcoin community to address these problems of privacy, scalability, and extensibility. Proposed techniques include sidechains [BCD$^+$14], the Lightning Network [JP16], Ark [ark23], Spiderchains [Bot], state channels [Hor18], and others with complicated tradeoffs.

In parallel, largely in Ethereum, others [Whi18, arb, opt, bas, zks, sta] have pursued an alternative scaling technology called a rollup [But18]. A rollup, like a sidechain is a parallel "layer 2" [2] blockchain, with its own separate transactions and execution semantics which are verified on L1 via a smart contract. Unlike a sidechain, all the rollup transactions are placed onto the "layer 1" to ensure they are publicly *available*. Thus, a rollup inherits the consensus mechanism of the layer 1 blockchain. While there

[2]The precise criteria for being a layer 2 are contentious

has been some work investigating rollups on bitcoin [Lig22], they have been relatively less explored than other scaling solutions.

1.1.1 SNARKs and zkRollups

Succinct Non-interactive ARguments of Knowledge (SNARKs) are a cryptographic tool for verifying computation. They allow a Prover to construct a succinct proof that they carried out some computation correctly. This proof can be verified by anyone, without communicating with the prover, much more efficiently than carrying out the computation [3]. In the last decade, SNARKs have gone from a mostly theoretical cryptographic construction to a practical, widely deployed technology.

SNARKs can be used to construct rollups in a very natural way, usually called a zkRollup or Validity Rollup. We simply construct a succinct proof that all the rollup transactions are valid, and condition the release of funds from the rollup on a valid SNARK. This allows the L1 to verify the execution semantics of the rollup by instead verifying a much simpler SNARK. It also allows reducing the amount of data a rollup needs to publish, for example by aggregating multiple signatures into a single proof. Unfortunately, bitcoin cannot verify SNARKs, so it is not possible to build zkRollups on bitcoin.

1.1.2 Optimistic Rollups

An alternative construction for rollups is an “Optimistic Rollup.” Rather than proactively proving a rollup is valid, optimistic rollups only produce a fraud proof when a rollup enters an *invalid* state. This easier much easier than constructing a SNARK, since to prove a rollup state is invalid, we only need to identify a single fault. One mechanism by which this occurs is called bisection [KGC$^+$18], as used in Arbitrum [arb] and Optimism [opt]. However, optimistic protocols in general require another party, beyond the SNARK Prover, to construct a fraud proof.

1.2 BitVM and BitVM2

When we say “verify SNARKs” we mean verify them as part of consensus. That is, all the bitcoin nodes verify the SNARK and reject blocks that contain invalid SNARKs. It is natural to ask if we might be able to relax this notion, while still retaining enough of the useful properties to construct an interesting rollup. That is, we would like to conditionally release funds from a rollup only if a Prover can prove that the rollup execution semantics allow for it.

Inspired by optimistic protocols, Linus introduced the revolutionary BitVM family of protocols. The reason why bitcoin cannot directly verify SNARKs is because there are severe limitations on the size of bitcoin scripts. If it were possible to construct larger bitcoin scripts or to share state between bitcoin scripts, we could verify SNARKs.

[3]Technically, only “work saving” [Tha22] SNARKs have this property

BitVM, and later BitVM2, combine the ideas of optimistic rollups with a novel way to connect the state of bitcoin scripts.

The "Prover" of a BitVM computation signs the intermediate computation states and puts these signatures on bitcoin. By signing sufficiently fine-grained state transitions, we can split the computation into pieces small enough to verify on bitcoin. If the Prover carries out the computation incorrectly, arriving at the wrong final state, then at least one intermediate step must have failed. We can then put the failing state transition on chain and show the Prover has faulted.

There are at least two variants of BitVM commonly called BitVM (one) and BitVM2. In BitVM, two predesignated parties engage in a bisection game to identify where in the computation the Prover faulted. This variant requires multiple rounds of interaction, and the Verifier must be fixed in advance. There are several instantiations of this protocol including BitVMX [LAM+24] and BitSNARK [FYG24]. In BitVM2, the Prover signs all the intermediate states, and anyone can provide a fraud proof. This means that BitVM2 is only suitable for small programs, in particular for SNARK verification.

Due to the idiosyncracies of bitcoin, there are a number of practical issues that arise when we attempt to instantiate BitVM-style protocols. We must use a signature scheme that is verifiable in bitcoin script, typically either Lamport [Lam79] or Winternitz [DSS05] signatures. We also need a way to encode state machines into multiple bitcoin transactions, which is not possible in bitcoin consensus due to the lack of covenants.

To emulate a covenant, we use a trusted committee to pre-sign a graph of bitcoin transactions that encode the BitVM game. This requires estimating many parameters of the system in advance, including the fee rate and time to finalize a transaction [4]. Alternatively, we can use ephemeral anchors. It is also possible to emulate covenants using very simple trusted hardware [Dom].

1.3 Glock

We introduce a new primitive for optimistic verification of computation on bitcoin called a Garbled Lock or Glock for short. The key insight, which was independently discovered by Jeremy Rubin and described in Delbrag [Rub25], is that we can replace the fraud proof mechanism of BitVM with a cryptographic object called a Garbled Circuit (GC).

In a GC protocol, there are two parties: a Garbler $\mathcal{G}$and an Evaluator $\mathcal{E}$. The protocol proceeds in three rounds: garble, authenticate, and evaluate and is carried out entirely *off-chain*.

1. $\mathcal{G}$ will choose secret input labels ℓ and output labels o and "garble" the circuit

$$[\mathcal{C}]_o^\ell = \mathsf{Garble}(\ell, o; \mathcal{C})$$

   $\mathcal{G}$ and $\mathcal{E}$ may additionally carry out a protocol to ensure that $[\mathcal{C}]_o^\ell$ is correctly constructed to ensure *malicious security*.

[4]Bitcoin lacks absolute finality, so this will always be probabilistic

2. $\mathcal{G}$ will choose some input $w \in \mathbb{F}_2$ and authenticate it by revealing input labels

$$[w]_\ell = \left(\ell_i^{(w_i)}\right)_{i\in[n]}$$

3. $\mathcal{E}$ will use "evaluate" the GC using the input labels to compute

$$[\mathcal{C}(w)]_o = \mathsf{Eval}([\mathcal{C}]_o^\ell, [w]_\ell)$$

If the protocol is secure, then $\mathcal{E}$ should learn $[m]_o$ if and only if $\mathcal{G}$ authenticated some w under ℓ such that $\mathcal{C}(w) = m$. So, if $\mathcal{C}(w) = 0$ constitutes a fault then $[0]_o$ constitutes a fraud proof. Given an authentication mechanism that is compatible with GC and verifiable on chain, we can use this fraud proof to slash on-chain.

In an auspicious harmony, the mechanism GCs use to authenticate inputs and outputs composes perfectly with the mechanism that BitVM already uses to efficiently authenticate on bitcoin: Lamport or more generally "projective" signatures. We say a signature is projective if a signature over a message can be "projected" into signatures over pieces, e.g. bits, of the message, and if given signatures over pieces of the message we can construct a signature over the whole message

$$\mathsf{Sign}(\mathsf{sk}, w \in \mathbb{F}_2^n) = \left(\mathsf{Sign}(\mathsf{sk}_i, w_i \in \mathbb{F}_2)\right)_{i\in[n]}.$$

Since a signature can be decomposed into signatures over bits of the message, we can enumerate all the possible signatures for each bit and we can treat these signatures as the input and output labels for a GC.

$$\ell_i^{(b)} = \mathsf{Sign}(\mathsf{sk}_i, b) \qquad o_i^{(b)} = \mathsf{Sign}(\mathsf{sk}'_i, b)$$

Thus, by signing a message w with the secret key sk we obtain the input labels for the GC since

$$[w]_\ell = \left(\ell_i^{(w_i)}\right)_{i\in[n]} = \left(\mathsf{Sign}(\mathsf{sk}_i, w_i)\right)_{i\in[n]} = \mathsf{Sign}(\mathsf{sk}, w).$$

Crucially, if we have a projective signature scheme that is compatible with bitcoin, we can force $\mathcal{G}$ to reveal the authenticated input on chain and can allow $\mathcal{E}$ to slash using the authenticated output. The simplest such scheme is a Lamport signature, where the secret key is simply a pair of random secrets. The public key is the hash of the secrets, and the signature is simply one of the secrets. We introduce an alternative scheme combining adaptor signature [GSST24] bit commitments [Lin23b] with verifiable secret sharing [CGMA85]. This scheme naturally generalizes to larger alphabets at no additional on-chain cost, yielding significantly smaller on-chain signatures.

Glock$(\mathcal{C}, \mathsf{Sign})$
If $\mathcal{G}$ authenticates $w \in \mathbb{F}_2^n$ such that $\mathcal{C}(w) = 1$, release funds after time t

1. $\mathcal{G}$ setup
   (a) Selects signature keys $(\mathsf{sk}_i, \mathsf{pk})_{i\in[n]}$ and $(\mathsf{sk}', \mathsf{pk}')$
   (b) Constructs labels $\ell_i^{(b)} = \mathsf{Sign}(\mathsf{sk}_i, b)$ and $o^{(b)} = \mathsf{Sign}(\mathrm{sec}'_i, b)$
   (c) Constructs $[\mathcal{C}]_o^\ell$
   (d) Sends $[\mathcal{C}]_o^\ell$ to the $\mathcal{E}$
2. $\mathcal{G}$ and $\mathcal{E}$ verify correctness of garbling table $[\mathcal{C}]_o^\ell$
3. Presign transactions $\mathsf{tx}_0, \mathsf{tx}_1, \mathsf{tx}_2$
   (a) tx_0 requires signature w under pk thereby revealing $\ell_i^{(w_i)}$
   (b) tx_1 requires signing the bit 0 under pk' by revealing $o^{(0)}$
   (c) tx_2 releases funds and requires waiting time t and tx_1 be unpublished
4. $\mathcal{G}$ publishes transaction tx_0 with signature $\sigma = [w]_\ell$ for message w
5. $\mathcal{E}$ evaluates $[\mathcal{C}]_o^\ell$ using $\left(\ell_i^{(b)}\right)_{i\in[n]}$ and learns $o^{(\mathcal{C}(w))}$
   (a) If $\mathcal{C}(w) = 0$, $\mathcal{E}$ publishes tx_1 with $o^{(0)}$
   (b) Otherwise, after time t, $\mathcal{G}$ publishes tx_2 and claims funds

This protocol by itself is not very useful and leaves several things underspecified. For example, the details of presigning are intentionally omitted, since it can work very differently in different contexts. Also, the details of projective signature verification are omitted as they can vary. The mechanism by which tx_1 can block the publication of tx_2 is also unspecified, although in practice we likely want to use a *connector output* [con23].

Glocks can be "permissioned" where the Evaluator is fixed in advance, or "permissionless" where the Evaluator is not. Glock25 is a permissioned Glock. The on-chain parts of the protocol make no claims about the GC scheme used, only that it accept input via wires over a small set. For example, the protocol naturally generalizes to wires over two, four, or even eight bit values. We are also free to choose a SNARK with a verification circuit $\mathcal{C}$ that efficiently arithmetizes into whichever garbling scheme we choose. Making these choices wisely, between circuit, garbling scheme, and signature scheme make the difference between (im)practicality.

1.4 Glock25

We propose an instantiation of a permissioned, maliciously secure Glock that is concretely practical. To do this, we depart from common SNARKs and instead use a novel Designated Verifier (DV) SNARK with much simpler verifier. The Garbler is the Prover for this SNARK and the Evaluator is the Verifier. This requires the Evaluator provide their secret verification key as an input to the SNARK verification circuit. We keep this private from the Prover/Garbler using a verifiable Oblivious Transfer (OT) protocol.

The DV SNARK replaces expensive pairing verification operations with simpler scalar multiplication operations and can be instantiated over any, not necessarily pairing-friendly, elliptic curve. We propose using a binary elliptic curve for compatibility with binary circuit based garbled circuits. Using such a curve, we reduce the AND gate complexity of the verifier by several orders of magnitude compared to a similarly sized prime field.

We also propose a novel method of achieving malicious security using Cut-and-Choose (CaC) with adaptor signatures that reduces the on-chain cost compared to using Lamport signatures. This technique is likely of independent interest for BitVM based protocols as well as it allows encoding data using a larger "base" without incurring on-chain costs. Apart from schemes that use Grug for malicious security, which require complex fraud proofs on chain, Glock25 is the only known, concretely practical Glock construction.

1.4.1 Designated Verifier SNARKs

Recall that we would like to use a Glock to condition funds on the validity of a SNARK. This means, we need to garble a SNARK verification circuit. To reduce on-chain costs as much as possible, we would like to use the smallest SNARK possible. While there are many known small SNARKs, including Groth16 [Gro16], Polymath [Lip24], and Pari [DMS24], they all require pairing-friendly elliptic curves and have relatively expensive verification circuits. In particular, the verifier must evaluate several elliptic curve pairings to verify a SNARK.

Verifying pairings requires working a large extension of a large prime field. This prime field is not a free parameter of the curve, so we can't use techniques for efficient field arithmetic used by Curve25519 [Ber06] or Secp256k1 [seca]. If we use a GC scheme for binary circuits, this field arithmetic must be encoded as a binary circuit. Making the scheme maliciously secure incurs another multiplicative blowup. Overall, this makes the cost of the garbled circuits for pairing based SNARK verifiers unacceptably large especially for our applications.

While it is possible that further research will discover efficient GC schemes for pairing based SNARK verifiers, we pursue a different approach based on *designated verifier* SNARKs. Following the technique of DV KZG [Orr24], we can replace pairings with fixed elements of the SRS by DV scalar multiplications. In particular, we propose a novel SNARK that is a minor modification of Pari. Our SNARK has a smaller proof size, saving a field element, and slightly simpler verifier and may be of independent interest.

Since we no longer need pairings, we can instantiate over any elliptic curve. For compatibility with binary circuit based garbling schemes, we choose to use a binary elliptic curve. These curves have subexponential time attacks [Sem15], which makes them unpopular. However, since they yield such a large efficiency benefit and since these attacks are slower than exponential time attacks for our security parameter, we believe it is acceptable.

In order to replace the pairings with scalar multiplications, we need to use another technique: Oblivious Transfer (OT) [Rab81]. This technique is very classical in garbled circuits for MPC. The Evaluator, who is the Verifier of the SNARK, needs to learn input labels corresponding to their secret verification key They also need to keep the key secret from the Garbler, as it would allow them to forge proofs. OT allows the Garbler to obliviously send these labels to the evaluator, without learning which labels it sent and without sending any other labels.

1.4.2 Malicious Security

This scheme assumes that the Garbler has honestly produced the GC $G[\mathcal{C}]$. A malicious Garbler could construct an invalid garbling, which would cause the evaluator to fail to derive o_0 even if the input is not satisfying. This would render the scheme insecure, so we need a mechanism to make Glocks maliciously secure.

There are many approaches known in the literature to achieve this, many of which can be applied in a blackbox manner. The simplest conceptually is to use another zkSNARK to prove that the Garbler constructed the table correctly. This has the advantage of being very communication efficient and efficient for the Evaluator. Unfortunately, it places a high computational burden on the Garbler, which for many schemes may be prohibitively high. While we believe the SNARK of correct grabling approach could be practical for DV Pari binary circuit, use appropriate SNARK friendly primitives where possible, we do not use it to instantiate Glock25.

Another approach is known as Cut-and-Choose (CaC). In this approach, the Garbler produces μ garbling tables with different input and output labels, and $\mathcal{E}$ chooses a subset to check during setup. Then, during evaluation, the rest of the tables are opened. If any invalid tables are discovered during setup, the protocol is aborted, and if any valid tables are discovered during evaluation $\mathcal{E}$ succeeds. By correctly choosing parameters, the probability of $\mathcal{E}$ not aborting during setup and failing during evaluation is exponentially small in the communication complexity.

Straightforward CaC poses a problem for Glock efficiency. It would require the Evaluator to be able to spend tx_1 with any subset of $1 + \mu/2$ outputs labels, and would require $\mathcal{G}$ put $\mu/2$ labels on chain in tx_0. The technique of [LP12] to secret share the labels provides a partial solution. Using a verifiable secret sharing scheme, this allows us to avoid output label dependence on μ, but still leaves the issue of the inputs. We propose a novel solution using adaptor signatures and a verifiable secret sharing scheme to put only a single signature on-chain and to reduce the size of the signature compared to the Lamport version.

Other Security Considerations There are several additional parts of the protocol that need to be maliciously secure. $\mathcal{E}$ must prove that the setup for their SNARK was performed correctly. There exist known techniques in the literature for this [BGG17] in the context of other SNARKs which can be readily adapted here. The Evaluator must also prove that they used OT to choose the correct labels for their verification key. We propose an efficient technique for this, although it is also possible to use a zero knowlege proof generically to achieve the same. There is also the possibility that either $\mathcal{E}$ or $\mathcal{G}$ fails to correctly carry out the setup procedure by failing to send messages. This is outside the scope of the protocol, and we will assume that there is some mechanism to attribute liveness failures during setup.

1.5 Concurrent Work

There has been significant, recent interest in GC based schemes. Rubin in Delbrag [Rub25] independently discovered and first published the technique of using GC for verifying off-chain computation on bitcoin. To achieve practical malicious security, he later proposed Grug introduces an optimistic, on-chain method whereby $\mathcal{E}$ can provide a fraud proof for an incorrectly constructed GC. This fraud proof is a bitcoin script, as in BitVM, but only needs to show that a single gate is malformed.

Linus published a GC based schemes called BitVM3 (RSA) [Lin25]. In the first, he proposed a novel RSA-based, reusable garbling scheme. This scheme would have allowed reusing a single GC across multiple inputs, saving on communication and verification costs when instantiated multiple times. Unfortunately this scheme seems to be broken [Eag25, FLB25]. Later, he published a new scheme called BitVM3s which uses a construction similar to Grug.

Chen [Che25] has suggested using alternative cryptographic primitives such as ABE [SW05] for the same style of construction. This would also allow reusable garbling, but unfortunately existing ABE schemes are not practical for SNARK verification circuits. Bitlayer [Bit25] has suggested using alternative garbling schemes which do not grow with the size of the circuit, but these schemes take much longer to garble and evaluate. There has also been significant progress on the application of GC to bridging, in particular "Ekrem's Trick" to use a single GC for all deposits [Tea25]. The specifics of bridge design are outside the scope of this paper, and we defer them to future work.

2 Preliminaries

Glocks touch on a number of different primitive technologies, including SNARKs, Garbled Circuits, bitcoin, etc. As a result, we require a fairly wide breadth of preliminary information to describe the protocol completely. We will not attempt to completely cover all the preliminaries here and will not formalize many properties of our scheme. Instead we hope to communicate just enough detail to understand about the primitives, to understand the scheme, and to be informally convinced of its correctness.

2.1 Notation

To refer to a finite fields of size p^k and characteristic p, we write $\mathbb{F}_{p^k}$. We denote an elliptic curve by E and the canonical prime r order subgroup of E by $\mathbb{G}$. We use additive notation for elliptic curves, with points written in capital letters $P, Q, ... \in \mathbb{G}$ and scalars in lower case letter $a, b, ... \in \mathbb{F}_r$. We reserve Greek letters $\alpha, \tau, ...$ for special values like random oracle challenges and the Verifier key. We write elements of a SNARK prover and verifier keys as $[F(X, Y, X)]$ which are polynomials in the SRS randomness.

To refer to a MAC of a vector of bits $w \in \mathbb{F}_2^n$ under key $\ell \in \mathbb{F}_p^{2\times n}$ we write $[w]_\ell$. To refer to a GC of a binary circuit $\mathcal{C} \in \mathbb{F}_2^n \to \mathbb{F}_2^m$ under input key ℓ and output key $o \in \mathbb{F}^{2\times m}$ we write $[\mathcal{C}]_o^\ell$.

We refer to the parties involved in the protocol such as the Garbler $\mathcal{G}$, Evaluator $\mathcal{E}$, Prover $\mathcal{P}$, and Verifier $\mathcal{V}$ using captial letters. In Glocks, we have that $\mathcal{G} = \mathcal{P}$ and $\mathcal{E} = \mathcal{V}$.

We write relations $\mathcal{R}$ for a circuit $\mathcal{C}$ using standard notation for public x and private w inputs $\mathcal{R} = \{(x; w) : \mathcal{C}(x, w) = 1\}$. When we define a SNARK, we do so with respect to such a relation.

2.2 Bitcoin

For simplicity and clarity, we consider a simpler abstraction over bitcoin. A bitcoin blockchain C of length L is a sequence of transactions tx. Each transaction includes a list of input "coins," output "coins," and input "witness" for each coin. Each coin has an associated variable denomination amount and a locking script, which accepts a witness

$$\mathsf{C} = \big(\mathsf{tx}_i\big)_{i\in[L]} \qquad \mathsf{tx} = \Big(\big(\mathsf{c}_i; \mathsf{w}_i\big)_{i\in[n]}, \big(\mathsf{c}'_j\big)_{j\in[m]}\Big) \qquad \mathsf{c} = \big(\mathsf{v} \in \mathbb{Z}, \mathsf{s}\big).$$

Bitcoin Transaction Validity
A transaction tx is valid with respect to a chain C if

1. Every input coin of tx is the output coin of some earlier $\mathsf{tx}_j \in \mathsf{C}$
$$\forall i \in [n] : \exists j < i : \mathsf{tx}_j \in \mathsf{C} \wedge \exists k : \mathsf{tx}.\mathsf{c}_i = \mathsf{tx}_j.\mathsf{c}_k.$$
2. The sum of input amounts equals the sum of output amounts
$$\sum_{i\in[n]} \mathsf{tx}.\mathsf{c}_i = \sum_{j\in[m]} \mathsf{tx}.\mathsf{c}'_j.$$
3. Every input coin's script accepts the corresponding witness
$$\forall i \in [n] : \mathsf{tx}.\mathsf{c}_i.\mathsf{s}(\mathsf{tx}.\mathsf{w}_i) = \top.$$

The first point means that the sequence of transaction is a topological ordering of a natural Directed Acyclic Graph of transactions.

Bitcoin script is a stack-based programming language based on Forth which is extremely limited. There is a maximum script size, a maximum stack size, and a very limited set of opcodes. It supports two "types" of values which we call "small script" and "big script." Small script values are 32 bit integers, and scripts can add, subtract, compare, but notably not multiply small script values. Big script values are byte strings of up to 500 bytes, and scripts can hash these strings, treat them as Schnorr signatures and public keys and check that the signature is valid over the spending transaction, but notably cannot "decompose" them into small script elements. Bitcoin script also supports conditional logic and stack manipulation opcodes. So, for example we could construct a script that verifies the first element of the witness is the preimage of a hash.

2.2.1 Covenants

Bitcoin script can only impose conditions on how the current coin can be spent. It cannot constrain where this coin is sent to, i.e. the script is invariant with respect to the spending transaction. This means we cannot build state machines out of bitcoin transactions directly. Bitcoin script also cannot access data outside the witness of the spending transaction.

Instead, we use presigned transactions [SHMB20] to impose these "covenants" on bitcoin transactions. This requires trusting set of signers not to presign any other transactions, as this would break the covenant. In some cases, this is compatible with the incentives of the protocol, but in other cases it requires an additional security assumption. There are a number of covenant proposals for bitcoin, including `OP_CTV` [RO20], `OP_TXHASH` [RB23], and `OP_CAT` [HS23]. Some of these would allow eliminating presigning entirely or even directly verifying SNARKs on bitcoin.

2.3 Signatures

A cryptographic signature scheme is a two-party protocol between a signer and a verifier. The signer has a public, private key pair $(\mathsf{sk}, \mathsf{pk})$. They should be able to produce signatures over messages $m \in \mathcal{M}$ using sk and anyone should be able to verify these signatures using pk. We informally say a signature scheme is secure if one cannot produce signatures without knowledge of the secret key. This should be true even if the adversary can choose the message. We say a signature scheme is a "one-time" signature scheme if signing multiple messages breaks the security of the scheme.

Signature with security parameter λ and message space $\mathcal{M}$

1. $\mathsf{Setup}(1^\lambda) \to S$ accepts the security parameter and returns public parameters s.

2. $\mathsf{KeyGen}(\mathsf{s}) \to (\mathsf{sk}, \mathsf{pk})$ accepts the public parameters and returns a key pair

3. $\mathsf{Sign}(\mathsf{s}, \mathsf{sk}, m \in \mathcal{M}) \to \sigma$ takes the secret key and message and returns a signature

4. $\mathsf{Verify}(\mathsf{s}, \mathsf{pk}, m, \sigma) \to \{0, 1\}$ accepts the public key and satisfies

$$\mathsf{Verify}(\mathsf{s}, \mathsf{pk}, m, \sigma) = 1 \implies (\mathsf{sk}, \mathsf{pk}) = \mathsf{KeyGen}(1^\lambda) \wedge \sigma = \mathsf{Sign}(\mathsf{s}, \mathsf{sk}, m)).$$

2.3.1 Projective Signatures

Suppose we can decompose the message space $\mathcal{M} = \mathcal{A}^n$ over an alphabet $\mathcal{A}$. A signature scheme is projective if we can also decompose the public key and signature in a way that respects message decomposition. That is for sub-keys and signatures $(\mathsf{sk}_i, \mathsf{pk}_i, \sigma_i)_{i \in [n]}$ we have

$$\mathsf{Sign}(\mathsf{sk}, m) = \big(\mathsf{Sign}(\mathsf{sk}_i, m_i)\big)_{i \in [n]} \qquad \mathsf{Verify}(\mathsf{pk}, m, \sigma) = \bigwedge_{i \in [n]} \mathsf{Verify}(\mathsf{pk}_i, m_i, \sigma_i).$$

If the Sign algorithm is deterministic, then projective signatures are necessarily one-time. We can enumerate all possible signatures $\sigma_i^{(a)} = \mathsf{Sign}(\mathsf{sk}_i, a)$, and given multiple signatures on different messages we can combine different sub signatures to forge. However, one-time projective signatures admit a very simple verification algorithm and are compatible with GCs. To verify, we can enumerate all possible sub-signatures and then test each component of a signature σ is equal to a valid sub-signature.

2.3.2 Lamport Signatures

The prototypical example of such a scheme is the Lamport signature [Lam79]. Secret keys are random pairs of bit strings, public keys are hashes of these strings, and signatures are one of the two secrets. When we instantiate using a collision resistant hash function that is supported by bitcoin script, these can be verified on bitcoin.

Lamport Signatures for a single bit and hash function $h \in \mathbb{F}_2^\lambda \to \mathbb{F}^{2\lambda}$

1. $\mathsf{KeyGen} \to \left(\mathsf{sk}_b \leftarrow \mathbb{F}_2^\lambda, \mathsf{pk}_b = h(\mathsf{sk}_b)\right)_{b=0,1}$
2. $\mathsf{Sign}(\mathsf{s}, \mathsf{sk}, b) \to \mathsf{sk}_b$
3. $\mathsf{Verify}(\mathsf{s}, \mathsf{pk}, b, \sigma) \to \mathsf{pk}_b \overset{?}{=} h(\sigma)$

Lamport signatures naturally generalize to larger message spaces $\mathcal{A}$ by replacing all the bits with elements of $\mathcal{A}$. However, this increases the complexity of the verification algorithm proportionally to the size of $\mathcal{A}$.

2.3.3 Schnorr Signatures

A Schnorr signature [Sch89] is a discrete log based signature that is natively verifiable by bitcoin. We describe the algorithms for Schnorr signature verification for completeness here.

Schnorr Signatures for a hash function $h \in \mathbb{F}_2^* \to \mathbb{F}^{2\lambda}$

1. $\mathsf{Setup} \to G \in \mathbb{G}$ choose a generator of a curve with order $r \sim 2\lambda$
2. $\mathsf{KeyGen} \to (\mathsf{sk} \leftarrow \mathbb{F}_r, \mathsf{pk} = \mathsf{sk}G)$
3. $\mathsf{Sign}(\mathsf{s}, \mathsf{sk}, m) \to (s, R = uG)$ where
   (a) $u \leftarrow F_r$
   (b) $s = u + h(\mathsf{pk}, R, m)\mathsf{sk}$
4. $\mathsf{Verify}(\mathsf{s}, \mathsf{pk}, b, \sigma) \to sG \overset{?}{=} R + h(\mathsf{pk}, R, m)\mathsf{pk}$

2.3.4 Adaptor Signatures

An adaptor signature [GSST24] is a modification to the Schnorr signature protocol to leak a discrete logarithm from a valid signature over a message m. We fix a nonce R and a group element $T = tG$ and have the signer shares the invalid "signature" type object

$$\sigma' = (s' = u + h(\mathsf{pk}, R + T, m)\mathsf{sk}, R).$$

The verifier can check that this signature object is "valid" by checking that

$$s'G = R + h(\mathsf{pk}, R + T, m)\mathsf{pk}.$$

Now, upon learning the valid signature $\sigma = (s, R+T)$ for m with nonce $R+T$ we can compute

$$(s - s')G = \big(R + T + h(\mathsf{pk}, R+T, m)\mathsf{pk}\big) - \big(R + h(\mathsf{pk}, R+T, m)\mathsf{pk}\big) = T.$$

Thus, by the injectivity of scalar multiplication, we have that $s - s' = t$. In order to force the signer to use the same nonce in both σ' and σ, we need to use a Schnorr multi-signature. We defer description of the full protocol [GSST24].

2.4 Verifiable Secret Sharing

A Secret Sharing (SS) protocol allows a Dealer to split a secret s into multiple shares s_i such that different subsets of these shares can recover s. The most famous example of such a protocol is Shamir's secret sharing protocol, which is parameterized over two integers $1 \leq k \leq n$. Given any k of the shares, one can reconstruct the secret s.

Shamir Secret Sharing for k out of n over a field $\mathbb{F}$

1. $\mathsf{Deal}(s) \to (s_i = f(i))_{i\in[n]}$ where
   (a) $f_i \leftarrow \mathbb{F}$ for $i \in [k-1]$
   (b) Let $f(X) = s + \sum_{i\in[k-1]} f_i X^i$.
2. $\mathsf{Recover}(S, (s_i)_{i\in S}) \to f(0)$ where
   (a) Interpolate $f(i) = s_i$ for $i \in S$.

Correctness follows from the fact that the interpolated polynomial agrees with the secret polynomial $f(X)$ on at least k points. The interpolated polynomial must therefore be exactly equal to the secret polynomial, so $f(0) = s$.

A Verifiable SS (VSS) additionally allows anyone to check that the secret shares were correctly constructed. To enable this, we assume assume that $\mathbb{F}$ is the scalar field of an elliptic curve and modify Deal to also output commitments to the coefficients of $f(X)$. Since these commitments are linear, anyone can verify given a share s_i that it was constructed from a particular committed $f(X)$ by evaluating the polynomial "in the exponent."

Verifiable Secret Sharing for k out of n over a field $\mathbb{F}$

1. $\mathsf{Deal}(s) \to (s_i = f(i), F_0 = sG, F_i = f_iG)_{i\in[n]}$ where
   (a) $f_i \leftarrow \mathbb{F}$ for $i \in [k-1]$
   (b) Let $f(X) = s + \sum_{i\in[k-1]} f_iX^i$.
2. $\mathsf{Recover}(S, (s_i)_{i\in S}) \to f(0)$ where
   (a) Interpolate $f(i) = s_i$ for $i \in S$.
3. $\mathsf{Verify}(i, s_i) \to s_iG \overset{?}{=} \sum_{j=0}^{k-1} i^jF_j$

We will make extensive use of VSS to efficiently make Glock25 maliciously secure. We also define the points $P_i = \sum_{j=0}^{k-1} i^jF_j = s_iG$ for $i \in [n]$. We can likely modify Glock25 to use a different VSS scheme very easily.

2.5 SNARKs

A SNARK or Succinct Non-interactive ARgument of Knowledge is a two-party protocol between a randomized Prover $\mathcal{P}$ and Verifier $\mathcal{V}$. A proof is defined with respect to a relation $\mathcal{R}$ over public inputs x and a witness w. In order to be a SNARK, the proof π must asymptotically smaller than $|w|$, and $\mathcal{P}$ should be able to construct π without interacting with $\mathcal{V}$. Using the Fiat-Shamir [FS87] transformation, public coin, interactive protocols can be made non-interactive by replacing the public coin randomness with the output of a hash function. [5]

SNARK

1. $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to (\mathsf{pk}, \mathsf{vk})$ outputs prover and verifier keys
2. $\mathsf{Prove}(\mathsf{pk}, x, w) \to \pi$ if $(x, w) \in \mathcal{R}$
3. $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to \{0, 1\}$ outputs 1 if $\exists w : \pi = \mathsf{Prove}(\mathsf{pk}, x, w)$

A SNARK requires "knowledge soundness" which informally states that if the Prover outputs an accepting proof π then they must know an accepting witness for the relation. SNARKs can also be zero knowledge, in which case the given π one cannot learn any information about w except that $(x, w) \in \mathcal{R}$. If the vk contains secret information, i.e. $\mathcal{P}$ could use vk to forge proofs, then we call the SNARK *Designated Verifier* (DV).

[5] This can be a subtle process, although the details of this are outside the scope of this paper.

2.5.1 Pairing Based SNARKs to DV SNARKs

Many SNARKs, including the smallest SNARKs, are constructed from elliptic curves with bilinear pairings. Pairings accept two curve points, typically in different subgroups of the r torsion, and are linear in both

$$e(A+B,C+D) = e(A+B,C)e(A+B,D) = e(A,C+D)e(B,C+D).$$

These pairings allow the verifier to multiply committed values in the exponent, but are very expensive cryptographic operations. In some proofs, like Groth16 [Gro16], both the first and second argument to the pairing are variable. In others, link Plonk [GWC19] and generally in KZG [KZG10] based proofs, the second argument is a constant in the proof. For these kinds of proofs, we can adapt the technique of DV-KZG [Orr24] to make the SNARK DV and replace pairings with scalar multiplications. We can replace the second argument points with their discrete logs and add these discrete logs to vk. These discrete logs are known during Setup.

We apply this transformation to Pari and use the result, with minor modifications in Glock25. This new SNARK as an added benefit has an exceptionally small proof size of $2\mathbb{G}+\mathbb{F}$. There are even smaller DV SNARKs [ADI25], but they work much differently and may have more complex verification circuits.

2.6 Garbled Circuits

A Garbled Circuits protocol is a two-party protocol conducted between a Garbler $\mathcal{G}$ and an Evaluator $\mathcal{E}$. It allows the $\mathcal{E}$ to evaluate a circuit $\mathcal{C}$ over an authenticated input from $\mathcal{G}$ and produce an authenticated output. This can optionally hide the structure of the circuit, although this is not necessary for Glock25. It may also be the case that $\mathcal{E}$ fails if $\mathcal{G}$ is malicious, which we can make negligible via a malicious security protocol. If $\mathcal{E}$ succeeds, then their derived output behaves like a proof that they evaluated $\mathcal{C}$ on an input from $\mathcal{G}$.

Garbled Circuit

1. $\mathsf{Setup}(1^\lambda) \to \mathsf{s}$ produces public parameters for the scheme
2. $\mathsf{Garble}(\ell, o; \mathcal{C}) \to \mathsf{GC}$ garbles the circuit under input and output MAC keys ℓ and o
3. $\mathsf{Auth}(k, w) \to \mathsf{MAC}$ returns the MAC of the value w under the key k
4. $\mathsf{Eval}(\mathsf{GC}, \mathsf{MAC})$ returns the MAC of $\mathcal{C}(w)$

2.6.1 Free XOR and Privacy Free Garbling

In standard Yao-style GC, the input MACs are very simple: the garbling of a bit is simply a Lamport signature of the bit. That is, one of two random secret values. This is used to MAC every wire in the circuit. The Free XOR [KS08] style optimizations use a different MAC. Each wire in the GC has a MAC key of the form $\Delta, K_i \in \{0,1\}^\lambda$ where Δ is the same for every wire. To MAC a bit we now use

$$[b]_{\Delta,K_i} = b\Delta \oplus K_i.$$

Now, we can XOR MACs to compute a MAC of the XORs of the bits

$$[a]_{\Delta,K} \oplus [b]_{\Delta,L} = [a \oplus b]_{\Delta,K\oplus L}.$$

As a result, the GC can garble XOR gates for free in the sense that they don't involve any cryptographic operations and don't increase the size of the GC.

2.6.2 Information Theoretic MACs

Free XOR admits a natural generalization to other groups via Information Theoretic (IT) MACs. To authenticate a value $x \in G$ in some group, we fix a MAC key (ϕ, k) where $\phi \in \mathrm{Endo}(G)$ is an endomorphism of G and $k \in G$ is a random group element. Writing the group in additive notation, the MAC is then constructed as

$$[x]_{\phi,k} = \phi(x) + k.$$

For Free XOR, the group is the additive group of a binary extension field, and the message is an element of $\mathbb{F}_2$. We will use IT MACs in the scalar field of an elliptic curve to simplify verifiable oblivious transfer.

2.6.3 Cut-and-Choose

A malicious Garbler could send an invalid garbling table to the Evaluator, which would cause the Evaluator to fail. There are two ways in which we could prevent this, either by having the Garbler construct a zkSNARK of validity for GC construction, or using cut-and-choose. The former technique is not feasible due to the size of the garbling table, so we use the latter.

In CaC, the Garbler sends multiple tables to the Evaluator during setup Then, the evaluator chooses half of these tables to open. $\mathcal{G}$ sends the input labels and from these $\mathcal{E}$ can check the tables are correctly constructed. If any tables are invalid, the Evaluator aborts the protocol.

During evaluation, the Garbler will authenticate the same input for all the remaining tables. If any of these tables are valid, we would like $\mathcal{E}$ to derive a secret. In fact, this is not strictly necessary for our application since we could check in bitcoin script that $\mathcal{E}$ simply knows one of the remaining output secrets. However, this would bloat on chain costs. Instead we use the standard technique of [LP12] to secret share the output labels.

This way, using the labels from the tables opened during setup, the Evaluator can derive a final secret so long as a single table is valid during evaluation.

Cut-and-Choose with μ tables

1. **Garble**$(\mathcal{C}, \mu) \to (\ell', \mathsf{G}')$
   (a) $\mathcal{G}$ produces μ garbling tables $\mathsf{GC}_i = [\mathcal{C}]_{o_i}^{\ell_i}$
   (b) $\mathcal{G}$ sends commitments $\mathsf{C}_i = \mathsf{cm}(\mathsf{GC}_i)$ to $\mathcal{E}$
   (c) $\mathcal{E}$ chooses $\mu/2$ tables at random $S \subset [\mu]$
   (d) $\mathcal{G}$ sends ℓ_i for $i \in S$ and GC_i for $i \notin S$
   (e) $\mathcal{E}$ constructs all GC_i for $i \in S$ and then verifies commitments
   (f) If any commitments are invalid, abort
   (g) Otherwise, output $\ell' = \big(\ell_i\big)_{i \in S}$ and $\mathsf{GC}' = \big(\mathsf{GC}_i\big)_{i \in [\mu]/S}$.

2. **Eval**$(\ell', \mathsf{GC}') \to (j, [\mathcal{C}]_{o_j}^{\ell_j})$
   (a) $\mathcal{G}$ sends $[w]_{o_i}^{\ell_j}$ for all $j \in [\mu]/S$
   (b) $\mathcal{E}$ attempts to evaluate all remaining tables
   (c) With probability $\binom{\mu}{\mu/2}^{-1}$, at least one evaluation j succeeds
   (d) Output $(j, [\mathcal{C}]_{o_j}^{\ell_j})$

To understand the security of the scheme, we need to compute the probability that the Evaluator does not abort during setup and fails during evaluation. Using μ tables and a secret sharing scheme for $\mu/2+1$ out of μ, this occurs when the Garbler constructs $\mu/2$ invalid tables and the Evaluator chooses exactly the other $\mu/2$ valid tables to open during setup. The probability of this occurring is exponentially small in μ

$$\binom{\mu}{\mu/2}^{-1} \sim \frac{\sqrt{\pi\mu}}{2^{\mu}}.$$

2.6.4 Oblivious Transfer

Oblivious Transfer (OT) allows the Garbler to transfer exactly one input label for each wire to the Evaluator without learning which label was sent. This is a crucial component of GC for MPC, and we will use it to authenticate the DV secrets for the Evaluator while keeping them private from the Garbler. There are many protocols of OT, and we are agnostic to the particular protocol.

3 Glock25

Glock25 consists of four novel contributions to achieve practical garbling table size and garbler and evaluator time. First, we modify Pari to have a smaller proof size and simpler evaluator, and we instantiate a DV version of the SNARK using DV-KZG [Orr24]. Next, we instantiate the SNARK using a binary elliptic curve and use an FFT-based multiplication circuit, which in combination with the free XOR [KS08] garbling technique allows us to perform elliptic curve scalar multiplications efficiently. Then, we use a verifiable OT scheme to allow the Verifier to perform only scalar multiplications in the circuit. To make the scheme maliciously secure, we propose a modified CaC procedure which avoids costly validity proofs and reduces the on-chain costs compared to even a single Lamport signature based SNARK Glock. Finally, we instantiate the scheme using a privacy free garbling scheme [FNO14] with free XOR.

This scheme meets our informal security requirements

1. If $\mathcal{G}$ authenticates w in tx_0 such that $\mathcal{C}(w) = 0$, then $\mathcal{E}$ can block release of funds by spending tx_1
2. If $\mathcal{G}$ authenticates w in tx_0 such that $\mathcal{C}(w) = 1$, then $\mathcal{E}$ cannot block release of funds

Assuming no communication failures during setup, this is therefore a secure Glock.

3.1 Designated Verifier SNARK

We adapt the SNARK Pari to have a simpler verifier and smaller proof size, and then replace the KZG polynomial commitment scheme implicit in Pari with the DV KZG commitment scheme of [Orr24]. We will only describe the verifier of the SNARK and defer a complete description appendix A. The verifier is the only part of the SNARK that is present in our Glock. We emphasize that this SNARK is a minor modification of Pari.

3.1.1 Modifications

The first modification we make is how the SNARK handles inputs. In Pari, inputs are passed, as in Plonkish SNARKs, via polynomial $i(X)$ that encodes them as evaluations of some polynomial. That is, for some domain D with values $d_1, d_2, ...$ the polynomial satisfies

$$i(d_j) = x_j \qquad j \in [k].$$

During verification, $\mathcal{V}$ evaluates $i(\alpha)$ at a Fiat-Shamir challenge point, which requires computing Lagrange evaluations. Even this relatively cheap operation is very expensive in a GC. We opt instead pass the inputs in the monomial basis, that is construct

$$i(X) = \sum_{i=0}^{k} x_i X^i.$$

Evaluating this polynomial is significantly cheaper, and incurs negligible overhead for the prover.

The second modification we make to Pari is to combine the quotient polynomial with the arithmetization. In Pari, the relation to prove is arithmetized into a polynomial equation $a(X)^2 = b(X) + z(X)q(X)$ where the structure of SRS constrains how the Prover can construct $a(X), b(X), q(X)$. We combine the polynomials $b(X)$ and $q(X)$ into a single polynomial $r(X) = b(X) + z(X)q(X)$, which we can enforce using the same mechanism as for constraining $a(X)$ and $b(X)$. This saves a scalar from the proof and slightly simplifies the GC.

3.1.2 Verifier Circuit

With these modifications, after applying Fiat-Shamir (FS) [FS87], to verify the proof $\pi = (P, Q, a)$ for statement $\mathcal{R}$ the Verifier must check that

$$e(P - (a[1] + (a^2 + i(\alpha))[\delta]), [1]) = e(Q, [\tau\epsilon] - \alpha[\epsilon]) \qquad \alpha = h(\mathcal{R}, P).$$

Where (τ, δ, ϵ) is the verifier key for the setup. Applying the transformation to a DV SNARK, the verification equation becomes

$$P = (a + (a^2 + i(\alpha))\delta)G + (\tau\epsilon - \alpha\epsilon)Q \qquad \alpha = h(\mathcal{R}, P). \tag{1}$$

This requires 1 hash evaluation to compute α, $2 + k$ scalar field multiplications for $k > 0$ inputs, and an MSM with 2 curve points. One of these curve points is fixed, and using Shamir's trick this MSM can be performed about as efficiently as a single scalar multiplication. Our circuit to garble can thus be defined

$$\mathcal{C}(\tau, \delta, \epsilon; \pi, x) = 1 \text{ if } 1 \text{ otherwise } 0.$$

3.2 Binary Elliptic Curves

We instantiate our SNARK using a binary elliptic curve. The security of binary elliptic curves is more complex than prime field curves. There exist subexponential attacks which become faster than Pollard rho at around degree 310. [Sem15] It is likely the case that Sect233k1 [secb] also called k233, delivers sufficient security for our application given this bound

$$\mathbb{F}_{2^{233}} \simeq \mathbb{F}_2[X]/(X^{233} + X^{74} + 1)$$
$$E_{k233}/\mathbb{F}_{2^{233}} : y^2 + xy = x^3 + 1.$$

This curve supports an efficient endomorphism by the coordinate-wise Frobenius endomorphism $(x, y) \mapsto (x^2, y^2)$ of order 233. This endomorphism is linear, i.e. uses only XOR operations, and so does not increase the size of the GC. Using this, we can perform scalar multiplication using the τ-and-add technique of [AHP05] to avoid doubling operations. Thus, the GC size depends only on the number of curve point additions. This is all true of other, larger Koblitz curves which we could use without modifying the protocol, such as k283.

3.2.1 Binary Field Arithmetic

To perform arithmetic in this curve, we must perform field arithmetic in the field $F = \mathbb{F}_{2^{233}}$. Recall that in a binary field, addition is XOR and is therefore free in our garbling scheme. Multiplication is equivalent to multiplication of polynomials in a quotient ring of $\mathbb{F}_2$.

There are many techniques for efficient field multiplication, including Karatsuba [KO62], Toom-Cook [Too63], and FFT-based methods [SS71]. In all of these methods, we encode the field elements as polynomials with "smaller" coefficients. To encode an element $a \in \mathbb{F}$ we choose an evaluation point e and a degree parameter d and define

$$a(Y) = \sum_{i=q}^{d} \hat{a}_i L_i(Y) \qquad a(e) = a.$$

For larger values of d, we can use smaller values for the coefficients $\hat{a}_i$. To multiply, we first evaluate the polynomials on some set D, multiply the evaluations, and then interpolate the result. The set D must be large enough that the interpolated result faithfully encodes the result of the product but must be chosen so $a(D)$ are as small as possible. Using larger degree and smaller coefficients yields more asymptotically efficient multiplication algorithms, but incurs higher costs for evaluation and interpolation.

For fields of our size, about 256 bits, it is usually not concretely optimal to use the maximal $d = \log_2 |\mathbb{F}|$. This is because the cost of evaluation and interpolation is concretely high. However, in binary fields evaluation and interpolation of polynomials is a purely *linear* operation, i.e. it uses only XOR. As a result, it has no effect on the garbled circuit size. Therefore, we propose using an FFT based multiplication procedure for F over $\mathbb{F}_2$ with degree 233. This is the most aggressive configuration and incurs only about 4 AND gates per bit of field element.

To garble a multiplication of $a, b \in F$ let $a(X), b(X) \in \mathbb{F}^{233}[X]$ be their polynomial representatives. We choose the smallest field $\mathbb{F}_{2^9}$ with enough elements, $2^9 > 2 \times 233$ and let $n = 52 > 2 \times 233/9$. Then choose $d_1, ..., d_{52} \in \mathbb{F}_{2^9}/\mathbb{F}_2$ such that $d_i^{2^j} \neq d_k$ for any i, j, k. We evaluate $a(X), b(X)$ at all the points d_i, and obtain for free there evaluations at all the conjugate points. This reduces the problem to computing 52 products in the field of size 2^9. We can repeat this procedure once again with smaller fields until we have reduced to products entirely within $\mathbb{F}_2, \mathbb{F}_{2^2}, \mathbb{F}_{2^4}$.

3.3 Oblivious Transfer

The first three inputs of the SNARK are the $\mathsf{vk} = (\tau, \delta, \epsilon)$. In the standard Glock design, the Garbler signs input labels for all the inputs to the circuit. Doing this would require the Garbler know the input labels encoding vk. However, since $\mathcal{G} = \mathcal{P}$, if know the input labels encoding vk then they know vk and can forge proofs. Therefore, we need some way to let the Garbler authenticate vk obliviously.

OT protocols provide exactly the mechanism we need for this. For each bit of vk, $\mathcal{E}$ can OT for one of two MACs. This doesn't reveal the bit to $\mathcal{G}$ and ensures that $\mathcal{E}$

learns only one. There are many OT protocols, as well as protocols that generalize the construction in various ways. Any of these will work for our purposes, and we propose using [Mic89].

3.3.1 Verifiability

If $\mathcal{E}$ requests the incorrect labels, then the DV Verifier will fail on valid SNARKs with high probability. This would make the Glock insecure.

To remedy this, $\mathcal{E}$ must prove that they learned the correct input labels, while still keeping them secret. We could use zero knowledge proofs for this purpose in a black box way, but this is complex and costly. Instead, we propose a more efficient construction using a simple sigma protocol.

Each input label for vk will be constructed using an IT MAC. Consider the input labels for τ indexed by $i \in [233]$. $\mathcal{G}$ will sample random values u and v_i in the scalar field of Sect233k1 and let $v = \sum_i v_i 2^i$. They will send to $\mathcal{E}$ the commitments $U = uG, V = vG$ and let the IT MAC keys be

$$\ell_i = (u, v_i).$$

Then, $\mathcal{E}$ will OT for the labels $t_i \in \{0,1\}$ for some value $t = \sum_i t_i 2^i$. Since IT MACs are linearly homomorphic, the Evaluator can compute an IT MAC of the value t from these labels under the key (u, v).

$$\sum_{i=0}^{232} [t_i]_{\ell_i} 2^i = [t]_{(u,v)}.$$

Note that $\mathcal{E}$ can compute exactly one such MAC from the labels that they received during the OT. To prove correct OT, they must prove $t = \tau$. We do this using a zk sigma protocol for the following relation, where H, H_τ are included in pk

$$\mathcal{R}_{OT} = \{(t, x; G, H_\tau, U, V) : xG = tU + V, H_\tau = tG\}.$$

This preserves the secrecy of τ by the zero knowledge property of the sigma protocol. By knowledge soundness, this means that $x = u\tau + v$. Since they learn an IT MAC of the value t, if $t \neq \tau$ then the Evaluator has learned two IT MACs of different values under the same key and can therefore extract (u, v). Thus the scheme is secure up to the security of the OT protocol and the discrete log problem in Sect233k1. We defer a protocol for this to the final section, as it interacts non-trivially with CaC.

3.4 VSS for Malicious Security

Since validity proofs of correct garbling are impractical, we use CaC to make our Glock maliciously secure. $\mathcal{G}$ will send μ garblings of the same circuit, and with high probability $\mathcal{E}$ will either detect a malicious garbling during setup or evaluation will succeed. Our solution requires only a single authentication on chain and reveals a single secret for an invalid SNARK. We do this by secret sharing the outputs *and the inputs*, and verify the correctness of this using a VSS. Using Secp256k1 to instantiate the VSS, we can use Schnorr adaptor signatures to directly leak a verified secret share. These techniques may be of independent interest for other Glocks and for BitVM style protocols.

3.4.1 Outputs

The output label scheme is essentially the classical CaC technique, with a small modification to remove the final Lamport signature verification in tx_1. The Garbler will use a VSS over Secp256k1 to secret share the two output MACs $[b]_o$ into μ shares $[b]_{o_j}$ for $j \in [\mu]$. We will use these outputs for the GCs

$$\mathsf{GC}_j = [\mathcal{C}]_{o_j}^{\ell_j}.$$

During setup, $\mathcal{G}$ will send the coefficient commitments to the verifier. $\mathcal{E}$ can compute commitments to the secret shares from these values $[b]_{Q_j} = [b]_{o_j}$.

Then $\mathcal{E}$ will query a random subset S of $\mu/2$ tables for $\mathcal{G}$ to open by sending all the ℓ_j and o_j keys. From these, the Evaluator can verify GC_j, derive the output labels for each $j \in S$, and verify $Q_b^{(j)}$ correctly commit to these secrets. If any tables or output labels were incorrectly constructed, $\mathcal{E}$ will abort the protocol. Finally, instead of presigning tx_0 to require a Lamport public key, we will presign to it to require a Schnorr signature under $O_0 = o_0 G$.

As in normal CaC, during evaluation $\mathcal{G}$ sends $[w]_{\ell_j}$ to $\mathcal{E}$ for all $j \in [\mu]/S$. $\mathcal{E}$ will use these to evaluate GC_j, and with high probability will learn at least one $[\mathcal{C}(w)]_{o_j}$ for $j \in [\mu]/S$. If the input is invalid, this will be a share of o_0, from which $\mathcal{E}$ can construct o_0 and therefore sign tx_1 under O_0. Thus, with high probability $\mathcal{E}$ will be able to spend tx_1 if $\mathcal{G}$ signs an invalid input. If $\mathcal{G}$ signs a valid input, they will not learn a share of o_0 and won't be able to sign so the Glock is secure.

3.4.2 Inputs

Using the same VSS scheme over Secp256k1, we can also make input secret sharing secure against a malicious Garbler. For each input wire i, $\mathcal{G}$ will choose secrets $\ell_i^{(b)} = [b]_{\ell_i}$ and again secret share these to $\mu/2+1$ out of μ shares $[b]_{\ell_{i,j}}$. $\mathcal{G}$ will use these labels to construct the μ GCs GC_j. They will publish the polynomial commitments, and $\mathcal{E}$ will use these to construct $[b]_{P_{i,j}} = [b]_{o_{i,j}} G$. $\mathcal{G}$ will also send $L_i^{(b)} = \ell_i^{(b)} G$.

In addition to verifying a random subset of GC_j and the output commitments, $\mathcal{G}$ will verify that the input secret shares are valid. If any of these tables are invalid, abort. Instead of using Lamport signatures in tx_0, we use a different projective signature scheme based on adaptor signatures.

Select a unique random public key for each input wire for $\mathcal{G}$ $A_i = a_i G$ and for $\mathcal{E}$ $B_i = b_i G$. $\mathcal{G}$ and $\mathcal{E}$ will prepare two adaptor signatures per input for $\mathcal{G}$ to leak either $[b]_{L_{i,j}}$ for $b = 0, 1$. We will presign tx_0 to require a signature from either A_i, B_i, which must be an adaptor signature.

During evaluation $\mathcal{G}$ will publish tx_0 along with one each adaptor signature revealing $[w]_\ell$. In conjunction with the opened input labels for $j \in S$, this is sufficient for $\mathcal{E}$ to reconstruct $[w]_{\ell_j}$ for all $j \in [\mu]$. This allows $\mathcal{E}$ to evaluate all the remaining labels and learn $[\mathcal{C}(w)]_o$.

Note that this construction generalizes naturally to passing inputs encoded over larger alphabets. Instead of bits, we could encode the circuit input in base 4 or larger without increasing the on-chain costs of the protocol. Using larger alphabets simply increases presigning costs, as we need one adaptor signature per input value per wire. This is in contrast to Lamport signatures whose public keys and verification complexity scale with the size of the input alphabet. In this way, using Schnorr signatures can actually reduce the overall on-chain cost compared to a single Glock with Lamport signatures.

3.5 Protocol

Here we describe the complete Glock construction with DV-Pari, verifiable OT, and modified CaC.

Glock Garble for $\mathcal{R}$

1. $\mathcal{E}$ constructs $\mathsf{vk}, \mathsf{pk} = \mathsf{Setup}(1^\lambda, \mathcal{R})$
2. Send pk to $\mathcal{G}$ with proof of correctness
3. Fix a binary circuit $\mathcal{C}(\mathsf{vk}; \pi, x)$ for verifying modified DV-Pari over Sect233k1
4. $\mathcal{G}$ constructs input and output labels and construct μ GCs GC_j
   - (a) Choose random $u, v = \sum_{i=1}^{233} v_i 2^{i-1}$ and let $\ell_i = (u, v_i)$.
   - (b) Choose random o_0, o_1
   - (c) Let $U = uG, V = vG$
   - (d) Secret share $[b]_\ell$ into $\mu/2 + 1 : \mu$ shares $[b]_{\ell_j}$ and polynomial $f_{i,j}(X)$
   - (e) Secret share $[b]_o$ into $[b]_{o_j}$ shares as well with polynomial $g_j(X)$
   - (f) Construct polynomial commitments $\mathsf{PC} = \left(f_{i,j}^{(b)}(X)G, g_j^{(b)}(X)G\right)_{i\in[n], j\in[\mu]}$
   - (g) Compute V_j from committed shares
   - (h) Garble $\mathsf{GC}_j = [\mathcal{C}]_{o_j}^{\ell_j}$
5. $\mathcal{G}$ sends (U, PC) and $(V_j, \mathsf{GC}_j)_{j\in[\mu]}$
6. $\mathcal{E}$ chooses $S \subset [\mu]$ at random of size $\mu/2$
7. $\mathcal{G}$ sends $\left(\ell_{i,j}\right)_{i\in[n], j\in S}$
8. For each $j \in S$ $\mathcal{E}$ checks that
   - (a) GC_j is correctly constructed
   - (b) VSS commitments correct for $[b]_{\ell_{i,j}}$ and $[b]_{o_{i,j}}$ for $b \in 0, 1, i \in [n], j \in [S]$.
   - (c) If any checks fail, abort
9. $\mathcal{G}$ and $\mathcal{E}$ use OT to send $[\mathsf{vk}]_{\ell_{i,j}}$ for i in vk input and $j \in [\mu]/S$.
10. If $\mathcal{E}$ fails to produce valid sigma protocol for $\mathcal{R}_{OT}$ for any j, abort
11. Construct adaptor signatures and presign $\mathsf{tx}_0, \mathsf{tx}_1, \mathsf{tx}_2$.

Glock Evaluate

1. $\mathcal{G}$ constructs a SNARK π and publishes tx_0 on chain, revealing $[\pi, x]_\ell$
2. Using OT output and keys from setup, $\mathcal{E}$ computes $[\mathsf{vk}, \pi]_{\ell_j}$ for all $j \in [\mu]$
3. Whp $\mathcal{E}$ computes $\mathsf{Eval}([\mathcal{C}]_{o_j}^{\ell_j}, [\mathsf{vk}, \pi]_{\ell_j}) = [\mathcal{C}(\mathsf{vk}, \pi)]_{o_j}$ for at least one j
4. Using keys from setup, derive $[\mathcal{C}(\mathsf{vk}, \pi)]_o$
5. If π is invalid, $\mathcal{E}$ publish tx_1
6. Otherwise, wait time t and $\mathcal{G}$ publish tx_2

4 Future Work

Our Glock is the first known, practical construction, and the first practical instantiation of GC for SNARK verification on bitcoin apart from Delbrag. However, there are many interesting alternative possible construction which merit more research. For example, if it were possible to instantiate a Glock with a non-DV SNARK, we could construct a practical *permissionless* Glock. This likely requires more research into improved circuits for small SNARKs and alternative GC schemes. Alternatively, it might be possible to use other kinds of cryptography than GCs to get Glocks with improved performance.

4.1 Garbling Groth16 with Yao

An alternative candidate Glock construction involves attempting to directly garble the verifier circuit for a Groth16 proof. This involves a large amount of large prime field arithmetic, and yields garbling table sizes orders of magnitude larger than ours. However, this approach would yield a permissionless Glock if practical. Due to the extremely large size of the circuit, it does not yet seem practical, however with alternative garbling schemes or more work on reducing the size of the circuit it may one day be practical.

4.2 Alternative Garbling Schemes

There is a vast literature of garbling schemes. Many of these improve on Yao-style garbling in various ways for different types of circuits. For example, there exist garbling schemes for arithmetic circuits [AIK12] over both integers and finite fields. These might be useful for garbling SNARK verifiers, which naturally reside in fields. There also exist succinct garbling schemes [ILL24] whose GC size does not scale with the size of the circuit. These alternative types of garbling schemes are not known to be practical yet, and more research is needed to apply them to Glocks.

4.3 Programmable Cryptography

A final direction, and perhaps the most interesting, is to replace the GC primitive entirely with a more powerful type of cryptography. For example, as proposed by Chen [Che25] we could use Attribute Based Encryption (ABE) instead of GC. This would allow a kind of "reusable" garbling scheme where a single Prover could use the same setup multiple times. At the moment, it seems that such schemes [BGG$^+$14] are not yet practical.

More speculatively, we could use Witness Encryption (WE) [GGSW13] or even Indistinguishability Obfuscation (iO) [BGI$^+$01] as an alternative primitive. These techniques are even more impractical than ABE, but would allow for substantially more powerful protocols. Some investigation into using WE for bitcoin bridges has been done [Hf22], although more work is necessary.

Acknowledgements

Thanks to Jeremy Rubin early discussion of GC on bitcoin and to Robin Linus. Thanks to the research team at Alpen Labs for helpful discussions and feedback around the DV-Pari Glock construction, and in particular to Aaron Feickert and Barak Shani for review and feedback on the paper. Thanks especially to Ying Tong Lai for extensive discussion of the protocol and review of the scheme and paper.

References

[ADI25] Gal Arnon, Jesko Dujmovic, and Yuval Ishai. Designated-verifier SNARGs with one group element. Cryptology ePrint Archive, Paper 2025/517, 2025.

[AHP05] Roberto M. Avanzi, Clemens Heuberger, and Helmut Prodinger. Minimality of the hamming weight of the tau-naf for koblitz curves and improved combination with point halving. Cryptology ePrint Archive, Paper 2005/225, 2005.

[AIK12] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. Cryptology ePrint Archive, Paper 2012/255, 2012.

[arb] Arbitrum.

[ark23] Ark protocol, 2023.

[bas] Base.

[BCD$^+$14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, , and Pieter Wuille. Enabling blockchain innovations with pegged sidechains, 2014.

[Ber06] Daniel J. Bernstein. Curve25519: new diffie-hellman speed records, 2006.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE, and compact garbled circuits. Cryptology ePrint Archive, Paper 2014/356, 2014.

[BGG17] Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. Cryptology ePrint Archive, Paper 2017/602, 2017.

[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 *Advances in Cryptology—CRYPTO 2001*, pages 1–18. Springer, 2001.

[Bit25] Bitlayer. Zerogc: A garbled circuit scheme for bitvm3 with zero ciphertext per gate, 2025.

[Bot] Botanix. Botanix protocol: An evm equivalent layer 2 on bitcoin.

[But18] Vitalik Buterin. On-chain scaling to potentially 500 tx/sec through mass tx validation, 2018.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (SFCS 1985)*, pages 383–395. IEEE Computer Society, 1985.

[Che25] Weikeng Chen. SoK: BitVM with succinct on-chain cost. Cryptology ePrint Archive, Paper 2025/1253, 2025.

[con23] Connectors, 2023.

[Cor24] Matt Corallo. Stop calling it mev, 2024.

[DMS24] Michel Dellepere, Pratyush Mishra, and Alireza Shirzad. Garuda and pari: Faster and smaller SNARKs via equifficient polynomial commitments. Cryptology ePrint Archive, Paper 2024/1245, 2024.

[Dom] Josh Doman.

[DSS05] Chris Dods, Nigel Smart, and Martijn Stam. Hash based digital signature schemes. In *Cryptography and Coding - IMACC 2005*, volume 3796, pages 96 – 115, Germany, November 2005. Springer Berlin Heidelberg. Conference Proceedings/Title of Journal: Cryptography and Coding, Springer LNCS 3796.

[Eag25] Liam Eagen. A note on bitvm3 rsa garbling, 2025.

[FLB25] Ariel Futoransky, Gabriel Larotonda, and Fadi Barbara. A note on the security of the BitVM3 garbling scheme. Cryptology ePrint Archive, Paper 2025/1291, 2025.

[FNO14] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. Cryptology ePrint Archive, Paper 2014/598, 2014.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO '86*, pages 186–194. Springer, 1987.

[FYG24] Ariel Futoransky, Yago, and Gadi Guy. Bitsnark & grail bitcoin rails for unlimited smart contracts & scalability, 2024.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology–EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 791–809. Springer, 2013.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. Cryptology ePrint Archive, Paper 2013/258, 2013.

[Gro16] J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.

[GSST24] Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Foundations of adaptor signatures. Cryptology ePrint Archive, Paper 2024/1809, 2024.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive*, 2019:953, 2019.

[Hf22] Leona Hioki and 3966f482edb3843a7cdfb9ffa6840023. Trustless bitcoin bridge creation with witness encryption, 2022.

[Hor18] Liam Horne. Counterfactual: Generalized state channels on ethereum, 2018.

[HS23] Ethan Heilman and Armin Sabouri. Bip 0347, 2023.

[ILL24] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct homomorphic MACs from groups and applications. Cryptology ePrint Archive, Paper 2024/2073, 2024.

[JP16] Thaddeus Dryja Joseph Poon. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[KGC$^+$18] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1371–1388, 2018.

[KO62] Anatolii A Karatsuba and Yuri P Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: free xor gates and applications. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. pages 177–194, 2010.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function, 1979.

[LAM$^+$24] Sergio Demian Lerner, Ramon Amela, Shreemoy Mishra, Martin Jonas, and Javier Alvarez Cid-Fuentes. Bitvmx: A cpu for universal computation on bitcoin, 2024.

[LAZ$^+$24] Robin Linus, Lukas Aumayr, Alexei Zamyatin, Andrea Pelosi, Zeta Avarikioti, and Matteo Maffei. Bitvm2: Bridging bitcoin to second layers, 2024.

[Lig22] John Light. Validity rollups on bitcoin, 2022.

[Lin23a] Robin Linus. Bitvm: Compute anything on bitcoin, 2023.

[Lin23b] Robin Linus. Commit to a bit value using a schnorr signature, 2023.

[Lin25] Robin Linus. Bitvm3: Efficient computation on bitcoin, 2025.

[Lip24] Helger Lipmaa. Polymath: Groth16 is not the limit. Cryptology ePrint Archive, Paper 2024/916, 2024.

[LP12] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, 2012.

[Mic89] Mihir Bellare Silvio Micali. Non-interactive oblivious transfer and applications, 1989.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[opt] Optimism.

[Orr24] Michele Orrù. Revisiting keyed-verification anonymous credentials. Cryptology ePrint Archive, Paper 2024/1552, 2024.

[Rab81] Michael O Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard University, Aiken Computation Laboratory, 1981.

[RB23] Steven Roose and Brandon Black. Bip: 346, 2023.

[RO20] Jeremy Rubin and James O'Beirne. Bip 0119, 2020.

[Rub25] Jeremy Rubin. Delbrag, 2025.

[Sch89] C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 239–252, 1989.

[seca] Secp256k1.

[secb] Sect233k1.

[Sem15] Igor A. Semaev. New algorithm for the discrete logarithm problem on elliptic curves. *CoRR*, abs/1504.01175, 2015.

[SHMB20] Jacob Swambo, Spencer Hommel, Bob McElrath, and Bryan Bishop. Bitcoin covenants: Three ways to control the future. *CoRR*, abs/2006.16714, 2020.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[sta] Starknet.

[SW05] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In *Advances in Cryptology–EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[Tea25] Citrea Team. R&d for clementine v2: Eliminating collateral and liquidity requirements with garbled circuits and toop, 2025.

[Tha22] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge.* Foundations and Trends® in Privacy and Security. Now Publishers, 2022.

[Too63] Andrei L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963.

[Whi18] Barry Whitehat. Roll_up, 2018.

[zks] Zksync.

A DV-Pari

The starting point for our DV SNARK is the recent work on Garuda and Pari. These two SNARKs continue work in the direction of Polymath on unifying the KZG [KZG10] and Groth16 [Gro16] style pairing SNARKs, achieving even smaller SNARKs than Groth with simpler verifiers. This is useful for us, as there is a known variant of the KZG PCS that support designated verifiers [Orr24]. There is no such straightforward transformation for Groth16.

All of these SNARKs represent their relations via either QAP [GGPR13] or S(quaring)AP which naturally generalizes to GR1CS [DMS24]. An encoding of a relation $(x, w) \in \mathcal{R}$ in this form consists of a "gate relation" $g : \mathbb{F}^{k+1} \to \mathbb{F}$ and a sequence of matrices $B \in \mathbb{F}^{m\times k}$ and $A_1, ..., A_k \in \mathbb{F}^{m\times n}$ The relation is satisfied if and only if

$$(x, w) \in \mathcal{R} \iff \forall i : g\big((Bx)_i, (A_1w)_i, (A_2w)_i, ..., (A_kw)_i\big).$$

The way that the public input x is handled can vary between implementations, and this will be convenient for us. The difference between a QAP and an SAP is using the gate relation $g(a, b, c) = ab - c$ versus $g(a, b) = a^2 - b$. There is a well known transformation between these two form that exploits the difference of squares identity when the field is not characteristic 2

$$ab = \big((a + b)^2 - (a - b)^2\big)/4.$$

Groth16 encodes these matrices into the SRS by left multiplying by the generators. This allows the prover to perform a verifiable matrix multiplication by simply committing to the witness. This same trick enables Polymath, Garuda, and Pari to avoid relatively more expensive permutation arguments or general matrix multiplication proofs at the cost of a per-circuit trusted setup.

The primary difference between these SNARKs and Groth16 is that they use the Fiat-Shamir transformation to open the resulting commitment. This allows them to evaluate the gate relation on the openings, whereas Groth16 evaluates it in the exponent directly using a pairing. As a result, Groth16 is much more limited in the types of gate relations it can support.

A.0.1 Overview

Our SNARK is a small modification of Pari. Given witness polynomials $a(X), b(X)$ and gate relation $a(X)^2 - b(X) = z(X)q(X)$ rather than opening $a(X)$ and $b(X)$ separately, which requires two scalars, we open $a(X)$ and $r(X) = b(X) - z(X)q(X)$. Note that for an accepting witness we have $r(X) = a(X)^2$ so it is only necessary to provide an evaluation for $a(X)$, saving a field element and yielding a new smallest SNARK. We also propose modifying the way in which public inputs are passed to the SNARK in order to simplify the verifier.

For clarity, we give a complete, self-contained description of our SNARK. Fix a pair of domains $\mathcal{D} = \{d_i\}_{i\in[m]}, \mathcal{D}'$ of size m with vanishing polynomials $z(X)$ and $z'(X)$ respectively, and Lagrange polynomials $L_i(X)$ and $L'_i(X)$ respectively. Assume there exists a pair of efficient transformations Extend which given evaluations of a degree $m-1$ polynomial on one domain, return the evaluations of the same polynomial on the other domain. That is, the Low Degree Extension (LDE).

The starting point for our SNARK is a relation $\mathcal{R} \subset \mathbb{F}^k \times \mathbb{F}^n$ encoded by matrices $A, B \in \mathbb{F}^{m\times n}$ such that

$$(x, w) \in \mathcal{R} \iff (Aw)_i^2 = (Bw)_i + (Dx)_i \text{ where } D_{ij} = d_i^{j-1}$$

The unusual encoding of inputs is done so that the verifier can pass inputs in the monomial basis. There exists a mechanical transformation of any R1CS instance to one of this form with double the number of witness elements and constraints, which we describe in an appendix. For these matrices, we can define polynomial bases

$$\big(a_i(X)\big)_{i=[n]} = A^\top\big(L_i(X)\big)_{j\in[m]} \qquad \big(b_i(X)\big)_{i=[n]} = B^\top\big(L_i(X)\big)_{j\in[m]}.$$

This allows us to transform elementwise product into a polynomial product in the usual way. If the prover knows some w such that the w linear combinations of these bases satisfy the gate relation modulo $z(X)$, then the instance is satisfied

$$a(X) = \sum_{i\in[n]} w_i a_i(X) \qquad b(X) = \sum_{i\in[n]} w_i b_i(X) \qquad i(X) = \sum_{j\in[k]} x_j X^{j-1}$$
$$(x, w) \in \mathcal{R} \iff a(X)^2 = b(X) + i(X) \bmod z(X).$$

Conventionally, we would prove knowledge of the quotient polynomial $q(X)$ and then check that the openings of $a(X), b(X), q(X)$ agree with the gate equation. This is what Pari does. In our case, we instead define the polynomial $r(X) = b(X) + z(X)q(X)$ which allows us to combine the openings of $b(X)$ and $q(X)$ into a single opening, saving a field element and simplifying the verifier. To do this, we simply define an additional polynomial basis.

$$q_i(X) = z(X)L'_i(X)$$

The relation is satisfied iff the prover knows a pair of vectors (w, q) such that

$$a(X) = \sum_{i\in[n]} w_i a_i(X) \qquad r(X) = \sum_{i\in[n]} w_i b_i(X) + \sum_{j\in[m]} q_j L'_j(X)$$
$$(x, w) \in \mathcal{R} \iff \exists \vec{q} : a(X)^2 = r(X) + i(X).$$

Next the verifier will sample a random point $\alpha \leftarrow \mathbb{F}$ for the openings, and the prover will commit to the opening of $a(X)$ and the quotients

$$k_a(X) = \frac{a(X) - a(\alpha)}{X - \alpha} \qquad k_r(X) = \frac{r(X) - r(\alpha)}{X - \alpha}.$$

Since $k_r(X)$ is of larger degree, it will be useful to commit to it using the Lagranges $L''(X)$ for the combined domain $\mathcal{D} \cup \mathcal{D}'$.

A.0.2 Protocol

To construct the proof, we simply use the same *equifficient* polynomial commitment scheme techniques of Pari. Our SRS will be tri-variate, and for the matrices A, B defined previously we define the generators

$$\vec{G}_M = \left[(A^\top \vec{L}(X) + B^\top \vec{L}(X)Y)Z\right] \qquad \vec{G}_Q = \left[z(X)\vec{L}'(X)YZ\right] \tag{2}$$

$$\vec{G}_{K,a} = \left[\vec{L}(X)\right] \qquad \vec{G}_{K,r} = \left[\vec{L}''(X)Y\right] \tag{3}$$

The generators $\vec{G}_M$ and $\vec{G}_Q$ ensure that the prover cannot only commit to polynomials $(a(X), r(X))$ as defined in terms of (w, q)

$$P = \sum_{i \in [n]} a_i G_{M,i} + \sum_{j \in [m]} q_j G_{Q,j} = [(a(X) + r(X)Y)Z].$$

The verifier will sample the challenge α and the prover will commit to the evaluation $a = a(\alpha)$ and KZG quotients together

$$Q = [k_a(X) + k_r(X)Y].$$

Finally, the verifier will check that

$$(P - a[1] - (a^2 + i(\alpha))[Y]) * [1] = Q * ([XZ] - \alpha[Z]).$$

In the standard SNARK case, these products can be performed with pairings. In the DV SNARK case, they will be performed by the verifier directly as scalar multiplications with the toxic waste.

Designated Verifier SNARK
Public input: SRS σ, $x \in \mathbb{F}^k$
Prover private input: Witness $w \in \mathbb{F}^n$
Verifier private input: SRS toxic waste $(X = \tau, Y = \delta, Z = \epsilon)$

1. $\mathcal{P} \to \mathcal{V} : P = \sum_{i \in [n]} w_i G_{M,i} + \sum_{j \in [m]} q_j G_{Q,j}$
2. $\mathcal{P} \leftarrow \mathcal{V} : \alpha \leftarrow \mathbb{F}$
3. $\mathcal{P} \to \mathcal{V} : Q = \sum_{i \in [n]} k_{a,i} G_{K,a,i} + \sum_{j \in [n+m]} k_{r,j} G_{K,r,j}$
4. $\mathcal{V}$ verifies $P - \big(a + (a^2 + i(\alpha))\delta\big)\epsilon G = (\tau - \alpha)\epsilon Q$.

A.1 Proofs

Completeness Follows directly from the protocol.

Knowledge Soundness In the AGM, the prover outputs polynomials $P(X,Y,Z)$ and $Q(X,Y,Z)$ respectively for each of the commitments in the protocol. These polynomials must satisfy the polynomial identity independent of the toxic waste

$$P(X,Y,Z) + (a + (a^2 + i(\alpha))Y)Z = (X - \alpha)ZQ(X,Y,Z).$$

If they did not, the adversary would obtain multiple affine equations in the toxic waste variables $(\epsilon, \delta\epsilon, \tau\epsilon)$. From these with non-negligible probability the adversary could obtain the toxic waste thereby solving the discrete log problem. Thus, the polynomials satisfy the equation with overwhelming probability.

By inspection of the equation, we see that $P(X,Y,Z) = ZU(X,Y,Z)$ because all the other terms in the equation are divisible by Z. Since the SRS only contains elements of degree 1 in Z, we also have $\deg_Z U = 0$ and therefore $\deg_Z Q = 0$. Since the only elements of the SRS with non-zero Z degree are in G_M, we further have that $U(X,Y) \in \langle a_i(X) + b_i(X)Y, q_j(X)Y\rangle_{i,j}$. We can therefore write for some w, q

$$U(X,Y) = a(X) + r(X)Y = \sum_{i\in[n]} w_i(a_i(X) + b_i(X)Y) + \sum_{j\in[m]} q_j q_j(X)Y.$$

Dividing the equation by Z and regrouping terms we have that

$$(a(X) - a) + (r(X) - a^2 - i(\alpha))Y = (X - \alpha)Q(X,Y).$$

By the unique factorization of multivariate polynomials over a field, we find that both $a(X) - a$ and $r(X) - a^2 - i(\alpha)$ are divisible by $X - \alpha$. That is, $v(X) = r(X) - a(X)^2 - i(X)$ vanishes at α. Rewinding over $n + m$ transcripts, we find that $v(X)$ vanishes at more points than it's degree and therefore $v(X) = 0$.

Thus, the adversary knows some w, q such that

$$a(X) = \sum_{i\in[n]} w_i L_i(X) \qquad b(X) = \sum_{i\in[n]} w_i b_i(X) \qquad q(X) = \sum_{j\in[m]} q_j q_j(X) \tag{4}$$

$$a(X)^2 = r(X) = b(X) + i(X) + z(X)q(X). \tag{5}$$

That is, the adversary knows a satisfying witness.