Zero-knowledge key-statement proofs

White Paper

WP0488

Copyright

Authors

Name	Title
Tom Trevethan (thomas@ncrypt.com)	R&D Specialist

Version history

Version	Date	Summary of changes

Related white papers

Reference	Nature of relationship

Contents

1 Abstract

This paper describes a method that enables the efficient zero knowledge verification of composite statements that involve both arithmetic circuit satisfiability and dependent statements about the validity of public keys (*key-statement proofs*) simultaneously. This is achieved by employing the required public key elliptic curve specification within the homomorphic commitment function used to prove circuit satisfiability, essentially proving public key statements corresponding to private keys used as circuit inputs and/or outputs 'for free'. This substantially reduces both the proof size and computational expense of generating proofs for statements involving both circuit satisfiability and elliptic curve key pairs. This method can be easily incorporated into existing discrete-log based zero-knowledge proof protocols for circuit satisfiability which do not require the use of bilinear pairing-friendly elliptic curves (and so is fully compatible with the Bitcoin secp256k1 standard).

We then describe two applications of this type of proof that relate to fair-exchange transactions between two parties on the bitcoin blockchain. The first involves a zero-knowledge contingent payment for the trust-less sale of an outsourced vanity address, which requires a zero-knowledge proof of the equality of a SHA256 hash preimage and an elliptic curve (Bitcoin) secret key. The second involves a privacy protecting cross-chain atomic swap, which requires a proof that a SHA256 hash pre-image is equal to unknown private key (with a supplied public key) multiplied by a supplied nonce.

2 Introduction

The creation of Bitcoin, the first decentralised and permission-less global blockchain, has for the first time enabled a solution to the problem of fair-exchange between two mutually un-trusting parties without the need for third party arbitration or escrow. A general form of this type of fair exchange for the sale of information (or *digital goods*) is embodied in a transaction protocol known as a *Zero-knowledge contingent payments* (ZKCP) [Maxwell 2016]. In a ZKCP, specified data is transferred from seller to buyer only if a payment is confirmed, and the payment from buyer to seller is only completed if the specified data is valid according to the conditions of the sale. The protocol is described in detail in [Campanelli 2017] but it is essentially based on the combination of a hash-time-locked contract (HTLC) with a zero-knowledge proof which *simultaneously* verifies that some encrypted information (the 'digital good') is valid/correct AND that the secret key to decrypt this information is the pre-image of the hash in the HTLC that must be revealed on the blockchain to claim the payment.

The central component of a ZKCP protocol is a *zero-knowledge proof* for a series of dependent statements about data/information validity or correctness, key validity and its corresponding hash value. Such complex composite statements require an efficient zero-knowledge proof system for general computations: in essence this enables one party to run an arbitrary program with secret inputs and then prove to another party that the program accepted the inputs as valid and was executed correctly – without revealing *anything* about the secret inputs or the execution of the program. In the ZKCP examples presented to date, the general purpose zero-knowledge proof system employed has been based on the *succinct non-interactive arguments of knowledge* (SNARKs) framework as implemented in the Pinocchio protocol [Parno 2016] and the C++ libsnark [Libsnark 2016] library.

Zero knowledge SNARKs (zkSNARKs) provide a method of proving, in zero-knowledge, the validity of arbitrary computations that can be expressed as arithmetic circuits. The two main distinguishing properties of zkSNARKs are that they are non-interactive (the prover sends the proof to the verifier in one move) and succinct (the proof is small and easy to verify). However, they have significant limitations:

a) The proof generation is extremely computationally demanding.

b) The proving key is very large and proportional to the circuit size.

c) They depend on strong and un-tested cryptographic assumptions (i.e. the knowledge of exponent assumption and pairing-based assumptions).

d) For any given program (circuit) they require a common reference string (CRS) to be computed by a third party who must be trusted to delete the set-up parameters. Any party with knowledge of the set-up parameters has the ability to create fake proofs.

In this paper, we describe a method to enable the proof a particular class of composite statements that involve relationships with elliptic curve public/private key pairs (based on elliptic curve point multiplications). The construction of zkSNARKs to prove statements that involve arbitrary cryptographic elliptic curve key operations has not been attempted to date (in the literature), but would likely consist of arithmetic circuits with many hundreds-of-thousands (if not millions) of gates (resulting in proof generation times of minutes and proving keys of hundreds of MB in size). To construct a much more efficient system, we propose a scheme where information on elliptic curve public keys is extracted directly from the 'homomorphic hiding' (or *commitment scheme*) used in the construction of proofs for generic circuit satisfiability. For this approach to work, the particular type of elliptic curve involved in the statement must be identical to that used in the circuit commitment scheme – however in the case of statements relating to Bitcoin public keys with zk-SNARKs this is not possible since the SNARK method involves pairing operations and therefore

requires special bilinear pairing-friendly elliptic curves (and the Bitcoin secp256k1 curve is not compatible). As a result, our approach is developed to be compatible with alternative protocols for proving arithmetic circuit satisfiability that do not rely on parings and have fewer cryptographic assumptions (see section 3.3). The resulting protocols additionally have the advantage of being more efficient than zkSNAKS (in terms of both computation and aggregate key/proof size) for trustless exchange applications.

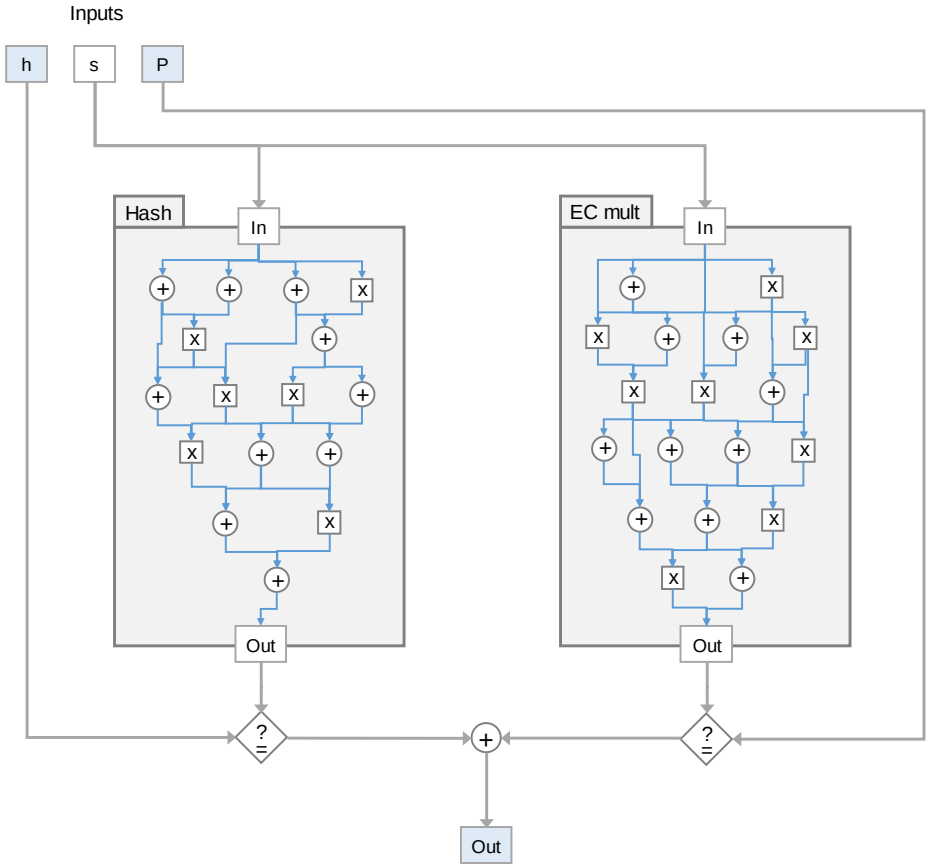

Figure 1. Schematic of a composite circuit for statement 1, containing sub-circuits for both a hash function and elliptic curve multiplication (internal gates are for illustrative purposes only – the real circuits would have 1000s of gates). The circuit checks that the output of the hash is equal to the EC public key: the values highlighted in blue are revealed to the verifier, all other values are encrypted.

An example of the type of statement that our approach is particularly suited and efficient for is as follows:

> **Statement 1**: "Given a hash function (H) output h and an elliptic curve point P (the public key), the pre-image of the hash s (i.e. $h = H(s)$) is equal to the elliptic curve point multiplier (the private key, i.e. $P = s \times G$)[1]"

[1] G is the elliptic curve generator point.

The ability to prove this particular statement in zero-knowledge would enable a number of novel applications, including the trust-less sale of outsourced bitcoin vanity addresses and anonymised, value-hiding cross chain atomic swaps (see applications).

The verification of the truth of statement 1 could be performed by the following (pseudo-code) function, that takes the inputs h, P and s and outputs 1 if the statement is true and 0 otherwise:

```
int verify(h,P,s)
{
      if(h == H(s) && P == s x G) {
            return 1
      }
      else {
            return 0
      }
}
```

Verifying this statement in zero knowledge (i.e. where the prover keeps the value of s secret from the verifier) with the zkSNARK system would require arithmetic circuits for both the hash function AND the elliptic curve point multiplication (see schematic in Fig 1). Arithmetic circuits for the SHA-256 hash function are widely used and optimised, and typically contain less than 30,000 multiplication gates, however there are no examples of arithmetic circuits for cryptographic elliptic curve point multiplication implemented the literature, and in any case they would be complex and contain many more gates. In the method we propose, we only require a full arithmetic circuit for the single hash function, and to explicitly prove (via circuit satisfiability) that s hashes to h. The fact that $s \times G$ also equals P can be extracted from the circuit proof explicitly at negligible extra computational cost by employing the required elliptic curve in the commitment scheme as part of the proof protocol (see Fig 2). We call this operation a *key-statement proof*, which is achieved by a new type of commitment opening procedure we call a *key-opening*.

The remainder of the paper is presented as follows: in the next section the relevant pieces of technical background and prior art are described in detail, then the following section the technical specification of the invention is presented. Finally, we present two potential applications that are made possible by the use of the efficient key statement proof method.

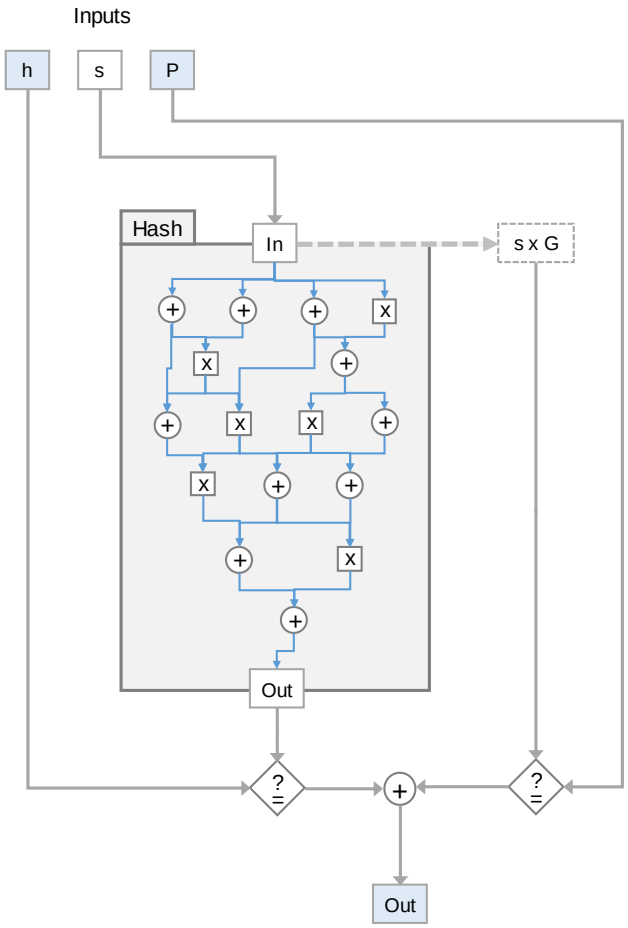

Figure 2. Schematic of an arithmetic circuit for the composite statement 1 employing a key-statement proof and just one arithmetic circuit for the hash function. The circuit checks that the output of the hash is correct and that the public key is equal to the EC encrypted input (a key-statement proof). The values highlighted in blue are revealed to the verifier, all other values are encrypted.

3 Background

This sections details the state-of-the-art and technical background that is required for the implementation of the proposed method.

3.1 Σ-Protocols

Σ (Sigma) protocols are a type of interactive zero-knowledge proof system, that involve a number of *moves* (communications) between the prover and verifier. Usually Σ protocols involve 3 moves: the prover sends an initial commitment to the verifier (a), the verifier then responds with a random challenge (x) and finally the prover answers with a final response, or 'opening' (z). The verifier then accepts or rejects the statement based on the transcript (a, x, z).

Σ protocols can be used to prove knowledge of, or statements about, a *witness* (w) that is known only to the prover. The protocol is zero-knowledge if it does not reveal any information about the witness to the verifier, except for the fact that a statement related to the witness is true [Bootle 2015].

3.2 Pedersen Commitments

Commitment schemes are a central part of many cryptographic protocols, and are a basic component of interactive zero-knowledge protocols for circuit satisfiability. A *commitment* enables a prover to *commit* to a secret value in advance, and then later verifiably reveal (*open*) the secret value. A commitment scheme has two main properties: 1) it is hiding: that is the commitment keeps the value secret, and 2) it is binding: the commitment can only be opened to the originally committed value.

The scheme employed in this invention is the Pedersen Commitment [Bootle 2015]. This scheme involves two elliptic curve generator points: G and F in the group $\mathbb{G}$ of prime order p, known to all parties. The committer generates a secure random number r in the field $\mathbb{Z}_p$, and then computes the commitment (via elliptic curve addition/multiplication) to the secret value s: [2]

$$Com(s,r) = s \times G + r \times F$$

The committer can at a later stage *fully open* the commitment (i.e. it can be verified), by providing the values s and r. The committer can also open the commitment in response to a specific challenge value as part of a Σ protocol (without revealing s or r).

A crucial property of Pedersen commitments is that they are additively homomorphic, meaning that adding (on the elliptic curve) two commitments results in a commitment to the sum of the committed values, i.e.:

$$(s_1 \times G + r_1 \times F) + (s_2 \times G + r_2 \times F) = (s_1 + s_2) \times G + (r_1 + r_2) \times F$$

This homomorphic property is exploited in zero-knowledge proofs of arithmetic circuit satisfiability.

3.3 Proofs of arithmetic circuit satisfiability in zero knowledge

An arithmetic circuit (over a field $\mathbb{Z}_p$) is a virtual construction of arithmetic *gates* that are connected by *wires* (forming a directed acyclic graph), that is capable of performing an arbitrarily complex

[2] Here, × denotes elliptic curve point multiplication.

computation[3]. Each gate has two input wires and one output wire and performs either a multiplication (×) or addition (+) operation on the inputs. A complete circuit has free input wires and free output wires that define the external (circuit) input and output values. We define a *legal assignment* of the values of the wires as those which *satisfy* the circuit, i.e. each wire is assigned a value where the output of each gate correctly corresponds to the product or sum of the inputs (i.e. the gate is *consistent*).

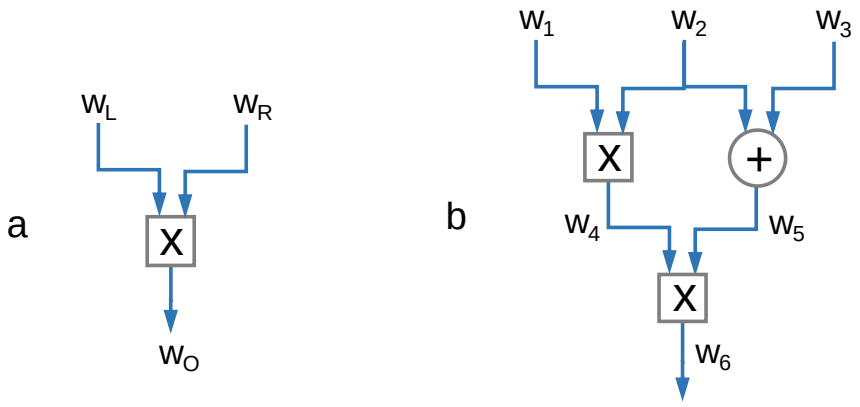

Figure 3. a) Schematic of a multiplication gate with left (w_L) and right (w_R) wire inputs and one wire output (w_O). b) A simple arithmetic circuit with three gates, three input wires (w_1, w_2, w_3), one output wire (w_6) and two internal wires (w_5, w_6).

For a given arithmetic circuit, a prover can *prove* to a verifier that they know a legal assignment for the circuit without revealing the wire values, by first committing to each wire value in the legal assignment (with Pedersen commitments) and then performing special Σ protocols with the verifier for each gate in the circuit (which can be performed in parallel), with the wire values as the witness. These Σ protocols exploit the homomorphic properties of Pedersen commitments, as described below.

To produce the proof (that a circuit is satisfied), initially the prover generates a commitment to each wire w_i in the circuit ($i = 1, \ldots, n$ where n is the number of wires) and sends these to the verifier:

$$W_i = Com(w_i, r_i)$$

3.3.1 Σ_{zero} protocol:

For each addition gate in the circuit, the Σ_{zero} protocol is executed: this involves proving (in zero knowledge) that $w_L + w_R - w_O = 0$ (i.e. that the addition gate is satisfied: the input wires w_L and w_R equal the output wire w_O).

1. The prover generates a commitment to zero: $B = Com(0, r_B)$, and sends to the verifier.
2. The verifier responds with a random challenge value: $x \leftarrow \mathbb{Z}_p$
3. The prover then computes the opening value: $z = x(r_L + r_R - r_O) + r_B$ and sends it to the verifier.

[3] The computation is limited to integer operations and must have no data dependant loops or mutable state.

4. The verifier checks that $Com(0, z) = x \times (W_L + W_R - W_O) + B$ as proof that $w_L + w_R - w_O = 0$

3.3.2 Σ_{prod} protocol:

For each multiplication gate the Σ_{prod} protocol is executed: this involves proving (in zero-knowledge) for each multiplication gate that $w_L \cdot w_R = w_O$ (i.e. that the multiplication gate is satisfied).

1. The prover generates 5 random blinding values: $t_1, t_2, t_3, t_4, t_5 \leftarrow \mathbb{Z}_p$
2. The prover computes $C_1 = Com(t_1, t_3)$, $C_2 = Com(t_2, t_5)$ and $C_3 = t_1 \times W_R + t_4 \times F$ and then sends them to the verifier.
3. The verifier responds with a random challenge value: $x \leftarrow \mathbb{Z}_p$
4. The prover computes the opening values:

$$e_1 = w_L x + t_1$$
$$e_2 = w_R x + t_2$$
$$z_1 = r_L x + t_3$$
$$z_2 = r_R x + t_5$$
$$z_3 = (r_O - w_L r_R) x + t_4$$

   and sends them to the verifier.

5. The verifier then checks the following equalities:

$$Com(e_1, z_1) = x \times W_L + C_1$$
$$Com(e_2, z_2) = x \times W_R + C_2$$
$$e_1 \times W_R + z_3 \times F = x \times W_O + C_3$$

   as proof that $w_L \cdot w_R = w_O$.

3.3.3 Circuit proofs

The Σ_{zero} and Σ_{prod} protocols can be operated in parallel for the verification of each gate in the circuit, and the same verifier challenge value (x) can be used for all gates.

As an example, consider the circuit in figure 3b: for a prover to prove in zero-knowledge to a verifier that they know a legal assignment (i.e. the wire values satisfying the circuit), the prover initially sends the wire commitments $(W_1, \dots, W_6)$ and the Σ protocol commitments for each gate to the verifier (this is one additional commitment for each addition gate and five for each multiplication gate).

The verifier then responds with the random challenge $x \leftarrow \mathbb{Z}_p$, and the prover computes the opening values for each gate (one for each addition and five for each multiplication) and sends them back to the verifier. The verifier then performs the Σ protocol checks to verify that:

$$w_1 \cdot w_2 = w_4$$
$$w_4 \cdot w_5 = w_6$$
$$w_2 + w_3 = w_5$$

and therefore that the commitments $W_1, \dots, W_6$ correspond to *satisfying* wire values $w_1, \dots, w_6$.

If the prover wants to show that, in addition to satisfying the circuit, a particular wire has a particular value, they can fully open the commitments to the relevant wires. In the example, the prover can additionally send the verifier the values w_6 and r_6 (the verifier can then confirm that $W_6 = Com(w_6, r_6)$) to demonstrate that w_6 is the actual output from a particular legal assignment.

The example in figure 3b is a trivial circuit, in practice useful circuits consist of many more gates. Of particular interest is an arithmetic circuit for the SHA-256 hash function - this circuit enables a prover to demonstrate that they know the pre-image (input) to a SHA-256 function that hashes to a particular (output) value, without revealing the pre-image. One of the most efficient implementations of a circuit for the SHA-256 algorithm consists of 27,904 arithmetic gates [Zcash 2016]. To prove knowledge of a SHA-256 pre-image would then require the sending of ~5 MB of data in both the initial commitment and opening rounds of the above protocol, and require ~200,000 elliptic curve operations for both the prover and verifier (taking a few seconds of processor time each). As described in section 3.5, there are some recently published protocols for substantially reducing these computational costs and proof sizes, without introducing any restrictions on the nature of the commitments.

3.4 Efficient zero-knowledge arguments for arithmetic circuit satisfiability without pairings

There are several methods that have been developed to significantly improve the performance of the parallel Σ protocol approach to proving arithmetic circuit satisfiability described in section 3.3. The approaches described in [Bootle 2016] and [Groth 2009] involve *batching* the commitments to circuit wire values to substantially reduce the size of data that must be sent from the prover to the verifier (i.e. reducing the communication complexity). These methods enable proof systems where the communication complexity is reduced from $\mathcal{O}(n)$ to $\mathcal{O}(\sqrt{n})$ or $\mathcal{O}(\log(n))$.

Again, as a comparison for proving the satisfiability of the same SHA circuit, the protocol of [Bootle 2016] has a proving key size of just 5 KB and a key generation time of 180ms. The Proof size is 24KB and takes ~4 s to generate, and the proof also takes ~4 s to verify.

We do not describe these methods in full here, except to state that the main vector batching protocol employed which is described below. This follows the same properties as the standard Pedersen commitment, but committing to n elements ($\mathbf{m} = m_1, \dots, m_n$) only requires the sending of a single group element:

1. The prover and verifier agree on a group element $F \leftarrow \mathbb{G}$
2. The prover generates n random numbers $x_1, \dots, x_n \leftarrow \mathbb{Z}_p$
3. The prover computes the points $K_i = x_i \times F$ (for $i = 1, \dots, n$). These values form a proving key PrK that is sent to the verifier.
4. The prover generates a random value: $r \leftarrow \mathbb{Z}_p$
5. The prover computes the commitment:

$$Com(\mathbf{m}) = r \times F + \sum_{i=1}^{n} m_i \times K_i$$

   and sends it to the verifier.

In the specification, we describe the method for extracting key-statements from both individual gate Σ protocols (embodiment 1) and the batched circuit vector commitments described above (embodiment 2). Therefore, our approach is compatible with each of the stated protocols.

4 Specification

This section describes the specifics of the invention, and its implementation for both batched and un-batched commitment based zero-knowledge proof systems.

Two parties are involved in the zero-knowledge proof protocol: the prover (P) and the verifier (V). The purpose of the protocol is for the prover to convince the verifier that a given statement ($\mathcal{S}$) is true, while keeping information about the witness to the statement secret. The statement consists of an arithmetic circuit ($\mathcal{C}$) with m gates and n wires, as well as dependent assertions about the elliptic curve public key(s) corresponding to one (or more) of the circuit wire values: pk_l (where the sub-script l is the wire index of the key statement. In addition, the statement may also include assertions about fully opened (public) wire values (i.e. public inputs/outputs of the circuit).

The elliptic curve public key(s) specified in the statement correspond to a target elliptic curve specification (which is defined by the full set of elliptic curve parameters: $\mathcal{T} = (p, a, b, G, n, h)$). In the case of Bitcoin script, these parameters are defined by the specification of secp256k1 [SEC 2010]. This specification includes the base generator point G.

In addition to the specifying the base point, the statement must also specify a second point F (where $F = f \times G$ and f is an element of $\mathbb{Z}_p$). The value of f must be provably random (e.g. the Bitcoin genesis block hash), or a 'nothing up my sleeve' number, such as the first 256 bits of the binary representation of π[4].

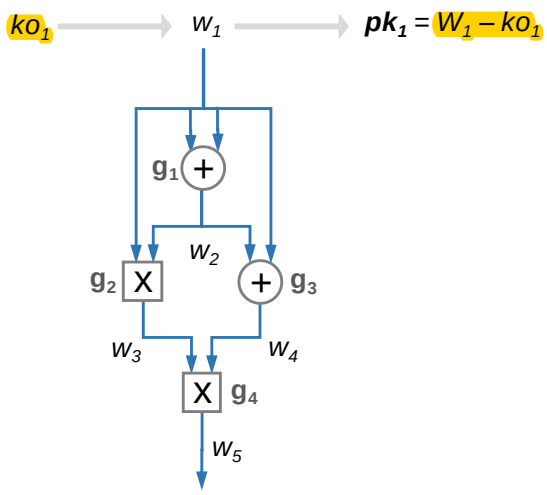

Figure 4. An example circuit with four gates and five wires. One input wire (w_1) has its public key revealed ('opened') from the wire commitment (W_1) with the 'key-opening' value ko_1.

4.1 Embodiment 1: Individual wire commitments

In this embodiment we describe *key openings* when a verifier has individual commitments to each wire in the circuit (i.e. following the Σ protocols for arithmetic circuit satisfiability in section 3.3).

1. Each wire i $(i = 1, \dots, n)$ of the circuit is committed to with a Pedersen commitment:

[4] Allowing the prover a free choice over f could enable them to generate fake proofs.

$$W_i = Com(w_i, r_i)$$

Where:

$$Com(w, r) = w \times G + r \times F$$

2. For the circuit wire l that requires a proof of its corresponding public key (a *key-statement proof*), the prover also sends a *key-opening*:

$$ko_l = r_l \times F$$

3. If a circuit wire j requires being publically revealed (a fully public wire), the prover sends the full opening tuple:

$$(w_j, r_j)$$

4. Each gate of the circuit is then proven satisfied in zero knowledge using the Σ protocols of section 3.3[5].

5. Once the verifier has confirmed that the circuit is satisfied, the verifier then calculates the public key for the wire l via elliptic curve point subtraction:

$$pk_l = Com(w_l, r_l) - ko_l$$

6. The verifier then confirms that each pk_l matches the key(s) specified in the statement (and that the fully opened wires match specified values) to complete the validation.

An explicit example (detailing the individual commitments and validations) of this protocol for the small example circuit shown in Fig. 4 is presented in Appendix 1.

[5] This involves the prover computing and sending the Σ_{zero} and Σ_{prod} commitments (i.e. B or C_1, C_2, C_3 respectively) for each gate, the verifier replying with a challenge value (x), the prover the sending the opening values (z and e values) and the verifier checking against the commitments.

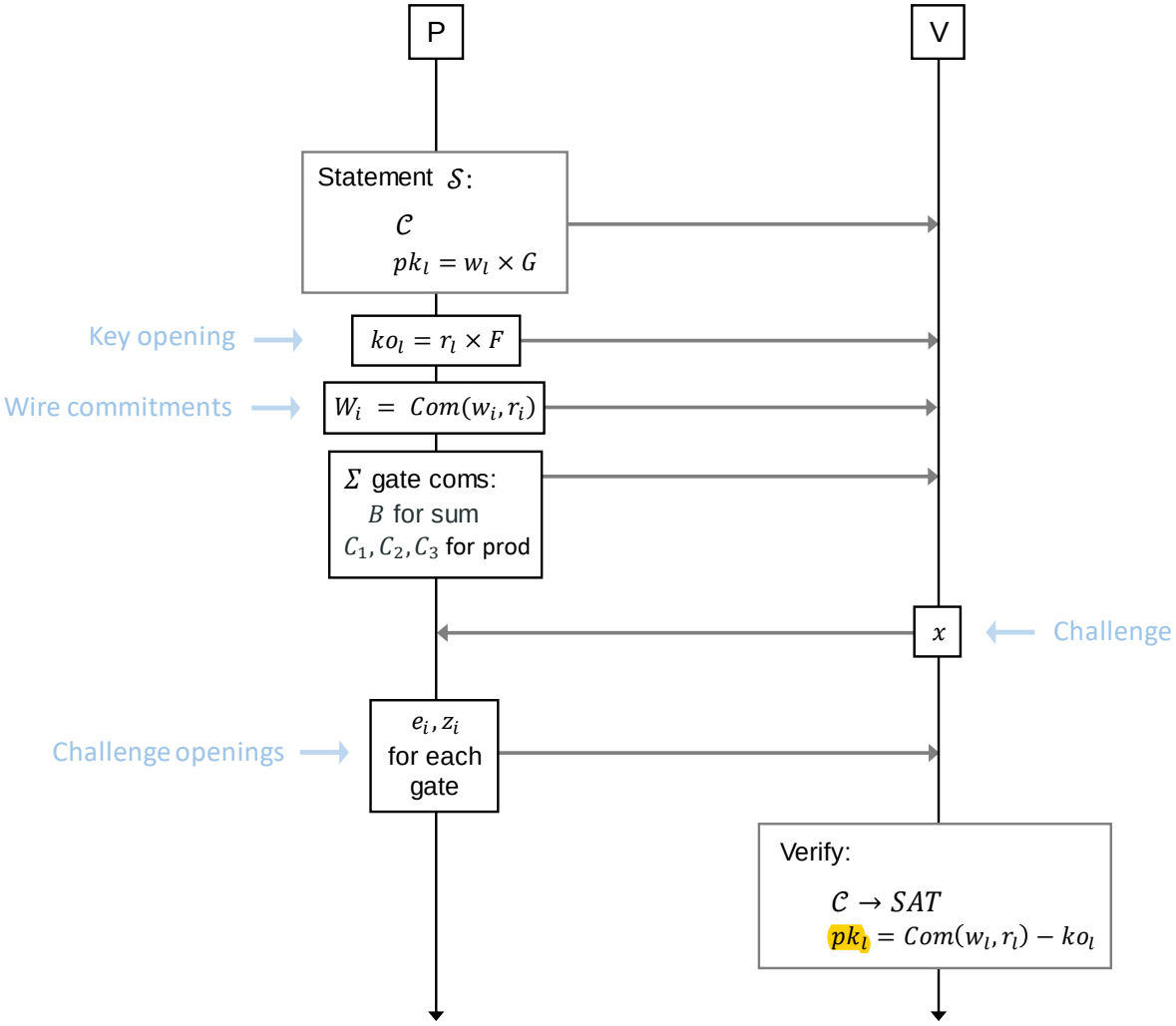

Figure 5. Protocol flow between prover (P) and verifier (V) for the proof of statement $\mathcal{S}$ with embodiment 1. The statement includes the circuit description and that wire l has public key pk_l.

4.2 Embodiment 2: Batched vector commitments

In the case of the compressed proof systems for circuit satisfiability that involve the batching of vector commitments [Bootle 2016, Groth 2009] (as described in section 3.5), the method described below can be used to extract *key-statement proofs* from *batched* circuit wire commitments. Here, we do not describe the full proof protocol, only the generation of the batched wire commitment and the protocol to demonstrate it contains a specified public key.

A batched commitment is generated as follows, where the wire l is to be supplied with a key opening. n wires are batched together in the vector commitment.

1. The prover generates $n-1$ random numbers $x_1, \ldots, x_{n-1} \leftarrow \mathbb{Z}_p$
2. The prover computes the elliptic curve points $K_i = x_i \times G$ (for $i = 1, \ldots, n-1$). These values plus $K_n = G$ form a proving key PrK that is sent to the verifier.
3. The prover generates a random value: $r \leftarrow \mathbb{Z}_p$
4. The prover computes the commitment to the vector $\mathbf{w}$ of wire values w_i (for $i = 1, \ldots, n$) where w_n is to be key-opened:

$$Com(\mathbf{w}) = r \times F + \sum_{i=1}^{n-1} w_i \times K_i + w_n \times G$$

and sends it to the verifier as part of the protocol in [Bootle 2016].

5. The prover also sends the *key opening* for the vector commit:

$$ko_n = r \times F + \sum_{i=1}^{n-1} w_i \times K_i$$

6. The verifier calculates the public key opening of the key-statement wire, via elliptic curve arithmetic:

$$pk_n = Com(\mathbf{w}) - ko_n$$

4.3 Proof of equivalence of a hash pre-image and elliptic curve private key

We here outline the construction of a specific example of a *key-statement zero-knowledge proof* that can be utilised in the applications described in sections 6 and 7.

The statement $\mathcal{S}$ is a more specific version of statement 1 given in section 2:

> $\mathcal{S}$: "Given a SHA-256 hash function (H) with a public output h and a public point P on the secp256k1 elliptic curve, the secret pre-image of the hash, s (i.e. $h = H(s)$) is equal to the elliptic curve point multiplier (i.e. the corresponding private key, i.e. $P = s \times G$)"

In the approach described above, this statement consists of a single arithmetic circuit for the SHA-256 hash function $\mathcal{C}_{SHA256}$ (with n wires w_i ($i = 1, \dots, n$) and m gates) along with an assertion that the input wire (w_1) is the private key for public point P and that the output wire (w_n) is equal to h. i.e.:

> $\mathcal{S}$: $\mathcal{C}_{SHA256}$ is satisfied by the wires $\{w_i\}_{i=1}^{n}$ **AND** $w_1 \times G = P$ **AND** $w_n = h$

Therefore, to fully verify this statement, the prover must demonstrate to the verifier that they know a satisfying assignment to the SHA256 circuit using the secp256k1 based commitment scheme, and then simply provide the *key-opening* for wire 1 (ko_1) and the *full opening* for wire n (w_n, r_n). The verifier does not learn the value of the input wire (w_1), or the values of any of the other wires except for the fully opened output wire w_n.

5 Advantages

The methods described in the previous section enable the zero-knowledge proof of statements that involve elliptic curve public-private key relationships simultaneously with general arithmetic circuit satisfiability. This is achieved with negligible computational expense beyond proving satisfiability of the arithmetic circuit, and avoids the requirement of creating explicit arithmetic circuits for elliptic curve point multiplication operations which would increase the computational expense of the proof substantially.

5.1 Comparison with zkSNARKs

Zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) refer to an implementation of a general purpose proof system for arithmetic circuit satisfiability. In the SNARK framework, a statement encoded as an arithmetic circuit is converted into a construct called a Quadratic Arithmetic Program (QAP), which consists of a set of polynomial equations. The statement can them be proved by demonstrating the validity of this set of equations at a single point. The main advantage of the SNARK method is that the verifier only needs to perform a few elliptic curve (pairing) operations (taking a few ms) and the proof is very small (288 bytes) and independent of the circuit size.

The very small proof and verification time achieved by the SNARK method comes at the expense of a trusted set-up, non-standard cryptographic assumptions and a much heavier computational burden placed upon the prover. The method also requires the use elliptic curve bi-linear pairings. However, the use of computationally feasible bi-linear pairings requires the use of special 'pairing-friendly' elliptic curves. This precludes the use of many standard cryptographic elliptic curve parameter sets, including the Bitcoin secp256k1. Statements involving general elliptic curve point-multiplications must then employ explicit circuits (which may be very large).

By way of comparison with the Σ protocol approach to proving SHA circuit satisfiability described in the previous section, with the SNARK (Pinocchio) framework, the proving key would take ~10 s to generate and be ~7 MB in size, and the proof would also take ~10 s to generate. The proof size would however be 288 B and only take ~5 ms to verify [Bootle 2016]. Additionally incorporating an explicit elliptic curve multiplication (key-statement) into the circuit, would multiply both the proving key size and proof generation time by at least an order of magnitude.

5.2 The Fiat-Shamir heuristic

The protocols described in this paper are interactive proof systems, which require communications to be passed back-and-forth between the prover and the verifier. This can be inconvenient for a ZKCP where the seller and buyer might not be on-line at the same time, and the buyer may wish for a proof to be publically verifiable (maybe as part of an advertisement for the digital goods). In addition, the proofs are only strictly zero-knowledge in the *perfect special-honest verifier* model: that is, we assume the verifier generates a genuine random number as the challenge, and does not choose the challenge value to try and extract information about the witness.

To solve both of these problems, we can apply the Fiat-Shamir heuristic: this simply replaces the random challenge value with the output of a hash of the commitments made by the prover. In the random oracle model (where the output of a cryptographic hash function is considered truly random), the prover cannot cheat, and the verifier can check the challenge value generated.

The Fiat-Shamir heuristic can then convert an interactive proof system into a non-interactive one, and the prover can generate a proof that can be independently and publically verified, off-line.

In the case of the protocol described in section 4.1 and Figure 5, the challenge value (x) is replaced with a value computed by hashing (with for example SHA-256) the concatenation of all of the commitments generated by the prover (i.e. all of the wire commitments and all of the B and C_1, C_2, C_3 commitments for sum and product gates respectively).

6 Application 1: Outsourced vanity address generation

This section describes an application of the specific proof described in section 4.3 to the ZKCP for an outsourced bitcoin vanity address.

Bitcoin addresses are encoded in a human readable alpha-numeric format (Base58 encoding) in order to make them easy to publish, copy and transcribe. The use of this format has led to the popularity of so-called vanity addresses, where the key space is brute-force searched in order to find a private key that results an address that contains a desired string (like a name).

Figure 6. Bitcoin main-net address with vanity pattern

Since deriving a vanity address with a significant pattern can be computationally expensive (for example, the address in figure 6 required the generation of approximately 10^{13} different public keys before a match was found) it is common for the search to be outsourced, and there are several online marketplaces where vanity addresses are commissioned and sold. This can be done securely using the homomorphic properties of elliptic curve point multiplication [JoelKatz 2012].

Although outsourcing the generation is secure, the sale of the vanity address is not trustless. Either the buyer gets the required value before the seller gets paid, or the seller gets paid before releasing the required value, or they must both trust an escrow service. By employing the proof described in section 4.3, the trustless sale of a vanity address via a ZKCP is enabled.

The protocol for this ZKCP is detailed as follows, and involves two parties: The *Buyer* and the *Seller*.

1. The Buyer and Seller agree on the required vanity pattern (Str) and the price (a BTC), and establish a communication channel (which does not need to be secure).
2. The buyer generates a secure random secret key sk_B and corresponding elliptic curve public key $pk_B = sk_B \times G$
3. The buyer sends pk_B to the seller.
4. The seller then performs a search for the required pattern in the Base58 encoded address derived from $pk = pk_B + i \times G$ by changing i.
5. When an address with the required pattern is found, the seller saves the value i, signals to the buyer and sends them $pk_s = i \times G$ and the SHA256 hash $H(i)$.
6. The seller also provides a proof (as in section 4.3) to the buyer that the pre-image to $H(i)$ is the private key corresponding to pk_s (as described in section 4.3).
7. The buyer verifies the proof, and also confirms that the address corresponding to $pk = pk_B + pk_s$ matches the agreed pattern. At this point (by virtue of the proof), the buyer *knows* that learning the value i will enable them derive the full private key for the vanity address ($sk_B + i$), and that particular value i hashes to $h = H(i)$.
8. The buyer then constructs a HTLC transaction Tx_1 which contains an output that contains the agreed fee (a). This output can be unlocked in in two ways:

i. With a signature from the seller and the hash pre-image, i, at any time.

ii. With a signature from the buyer after a specified time (OP_CLTV[6])

9. The buyer then signs and broadcasts this transaction to the blockchain, where it is mined into a block.
10. Once confirmed, the seller can claim the fee in the output of Tx_1 by providing a transaction Tx_2 supplying their signature and the value i to unlock the hash-lock, which is then revealed on the blockchain.
11. The buyer can calculates the final vanity address private key $sk = sk_B + i$, where $pk = sk \times G$
12. If the buyer fails to supply the value i before a specified OP_CLTV time, then the seller can provide their signature to re-claim the fee (to prevent the fee being lost due to an un-cooperative buyer).

The transaction is then fully atomic and trustless: the buyer only gets paid if they provide a valid value i which is revealed publically on the blockchain. Due to the splitting of the private key, this value is of no use to anyone else and does not compromise the security of the full private key.

[6] The CHECKLOCKTIMEVERIFY script op code, which can be used to prevent an output from being spent until a specified time (or block height).

7 Application 2: Privacy preserving cross-chain atomic swaps

A more straightforward type of trustless fair-exchange protocol that leverages blockchain transaction features is the so-called *cross-chain atomic swap* or *atomic trade*. The protocol is used to trade two different cryptocurrency tokens on two different blockchains without a third party centralised exchange. The word 'atomic' in this context refers to the fair exchange property: either the both parties complete their transactions or neither do.

The basic protocol is executed as follows. To be secure, both of the cryptocurrencies used in the swap must have scripting functionality that enables hashed and time-locked contracts. The swap involves two parties: Alice and Bob. In this example, Alice has 1 Bitcoin and has agreed to trade it for Bob's 100 Litecoins.

1. Alice generates a Litecoin public key P_A which she sends to Bob
2. Bob generates a Bitcoin public key P_B which he sends to Alice
3. Alice generates a secure random number x
4. Alice computes the SHA-256 hash of x: $h = H(x)$
5. Alice creates a Bitcoin transaction Tx_A which:
    i. Pays 1 Bitcoin to P_B with a valid signature AND a value that hashes to h
    ii. OR pays 1 Bitcoin back to Alice after 24 hours.
6. Alice broadcasts the transaction to the Bitcoin network.
7. Once Bob observes Tx_A is confirmed on the Bitcoin blockchain, he creates a Litecoin transaction Tx_B which:
    i. Pays 100 Litecoin to P_A with a valid signature AND a value that hashes to h
    ii. OR pays 100 Litecoin back to Bob after 24 hours.
8. Bob broadcasts the transaction to the Litecoin network.
9. Once the transaction has confirmed, Alice can then claim the Litecoin output by providing her signature and the value x.
10. Once Bob observes the value x on the Litecoin blockchain and can then claim the Bitcoin output by providing his signature and the value x.

The protocol is designed so that either both parties get their coins, or neither do. Alice generates the hash value and only she knows the pre-image, but she is required to reveal this pre-image to claim the coins, which then enables Bob to claim his coins. If either party does not follow the protocol to completion, both of them can re-claim their coins after a lock-out period.

7.1 Anonymity with key-statement proofs

One significant feature of the protocol described above is that the transactions on both blockchains are trivially linkable: once confirmed, the unique value x is publically visible on both blockchains, forever. This affects both the fungibility of the coins and the privacy of the transaction. In order to un-link the two transactions, different keys must be used for the outputs on each chain, but in order for the protocol to be secure and trustless, Bob must be given a *proof* by Alice that he will learn the information needed to unlock his coin when she reveals her hash pre-image.

By employing the *key-statement proof* (section 4.3), the hash-locked output on the second blockchain can be converted to a normal pay-to-public-key-hash (P2PKH) output, both disguising

the nature of the transaction and breaking any possible link. This protocol would proceed as follows (again, Alice has 1 Bitcoin and has agreed to trade it for Bob's 100 Litecoins):

1. Alice generates a Litecoin public key P_A (with private key s_A) which she sends to Bob
2. Bob generates a Bitcoin public key P_B (with private key s_B) which he sends to Alice
3. Alice generates a secure random number $x \leftarrow \mathbb{Z}_p$
4. Alice computes the SHA-256 hash of x: $h = H(x)$ and a the elliptic curve public key corresponding to x: $P_x = x \times G$
5. Alice securely sends both h and P_x to Bob.
6. Alice also sends a *key-statement proof* to Bob that the pre-image of h is equal to the private key that generated P_x (as described in section 4.3).
7. Alice creates a Bitcoin transaction Tx_A which:
    i. Pays 1 Bitcoin to public key $P_C = P_B + P_x$
    ii. OR pays 1 Bitcoin back to Alice after 24 hours.
8. Alice broadcasts the transaction to the Bitcoin network.
9. Once Bob observes Tx_A is confirmed on the Bitcoin blockchain, he creates a Litecoin transaction Tx_B which:
    i. Pays 100 Litecoin to P_A with a valid signature AND a value that SHA-256 hashes to h
    ii. OR pays 100 Litecoin back to Bob after 24 hours.
10. Bob broadcasts the transaction to the Litecoin network.
11. Once the transaction has confirmed, Alice can then claim the Litecoin output by providing her signature and the value x.
12. Once Bob observes the value x on the Litecoin blockchain and can then claim the Bitcoin output by providing a signature using the private key for P_C, which is $s_B + x$ from the homomorphic properties of elliptic curve point multiplication.

8 References

Reference	Author, date, name & location
[Campanelli 2017]	Campanelli, Matteo, et al. "Zero-knowledge contingent payments revisited: Attacks and payments for services." *Commun. ACM* (2017).
[Maxwell 2016]	https://github.com/zcash-hackworks/pay-to-sudoku
[Parno 2016]	Parno, Bryan, et al. "Pinocchio: Nearly practical verifiable computation." *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013.
[Libsnark 2016]	https://github.com/scipr-lab/libsnark
[Bootle 2015]	Bootle, Jonathan, et al. "Efficient zero-knowledge proof systems." *Foundations of Security Analysis and Design VIII*. Springer, Cham, 2015. 1-31.
[Groth 2009]	Groth, Jens. "Linear Algebra with Sub-linear Zero-Knowledge Arguments." *CRYPTO*. Vol. 5677. 2009.
[JoelKatz 2012]	https://bitcointalk.org/index.php?topic=81865.msg901491#msg901491
[Bootle 2016]	Bootle, Jonathan, et al. "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting." *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 2016.
[SEC 2010]	Standards for Efficient Cryptography (SEC) (Certicom Research, http://www.secg.org/sec2-v2.pd

9 Appendix 1

Here we describe in detail the protocol for verifying a satisfiability of a simple arithmetic circuit with both a key-statement proof of one of the wires and a full revealing of another wire. The circuit $\mathcal{C}$ is shown in Fig. 7: it has 5 wires w_i ($i = 1, \dots, 5$) and 4 gates g_j ($j = 1, \dots, 4$). Gates 1 and 3 are addition gates and gates 2 and 4 are multiplication gates.

The statement ($\mathcal{S}$) the prover wants to prove the truth of to the verifier is: "I know a satisfying assignment to the circuit $\mathcal{C}$ (i.e. wire values $\{w_i\}_{i=1}^{5}$ that satisfy all the gates), where wire 1 has a public key P (i.e. $P = w_1 \times G$) and wire 5 has a value h (i.e. $w_n = h$)" without revealing the values of wires 1 to 4.

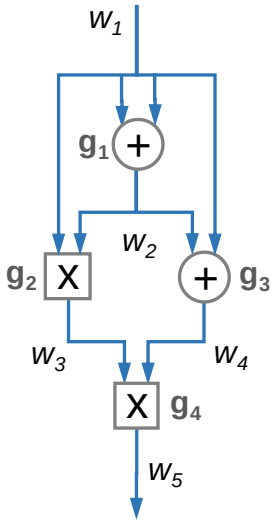

Figure 7. Circuit $\mathcal{C}$.

The prover and verifier agree on the statement (which includes the circuit, the value of wire 5 and the public key of wire 1, along with the elliptic curve and commitment specification). The protocol then proceeds as follows:

1. The prover generates 5 random blinding values ($r_1, \dots, r_5$) and then calculates the 5 wire commitments ($W_1, \dots, W_5$) and sends these to the verifier.
2. The prover computes the key opening for wire 1: $ko_1 = r_1 \times F$ and sends it to the verifier.
3. The prover sends the full opening information for wire 5 (w_5, r_5) to the verifier.
4. For the addition gates (g_1 and g_3) the prover generates a commitment to zero (with a random nonces r_{B1} and r_{B3}) : $B_1 = Com(0, r_{B1})$ and $B_3 = Com(0, r_{B3})$ and sends them to the verifier.
5. For the multiplication gates (g_2 and g_4) the prover generates the commitments as follows:

   For gate **2**: $C_{12} = Com(t_{12}, t_{32})$, $C_2 = Com(t_{22}, t_{52})$ and $C_3 = t_{12} \times W_1 + t_{42} \times F$
   For gate **4**: $C_{14} = Com(t_{14}, t_{34})$, $C_2 = Com(t_{24}, t_{54})$ and $C_3 = t_{14} \times W_3 + t_{44} \times F$

   Where the t_{xx} values are random blinding nonces. The prover sends these commitments to the verifier.

6. The verifier then generates a random challenge value x, and sends it to the prover. (OR the prover can generate value x by hashing a concatenation of all the commitments, via Fiat-Shamir).
7. For the addition gates (g_1 and g_3) the prover computes the following openings and sends them to the verifier:

$$z_1 = x(r_1 + r_1 - r_2) + r_{B1}$$
$$z_3 = x(r_2 + r_1 - r_4) + r_{B3}$$

8. For the multiplication gates (g_2 and g_4) the prover computes the following openings and sends them to the verifier:

$$e_{12} = w_1 x + t_{12}$$
$$e_{22} = w_2 x + t_{22}$$
$$z_{12} = r_1 x + t_{32}$$
$$z_{22} = r_2 x + t_{52}$$
$$z_{32} = (r_3 - w_1 r_2)x + t_{42}$$

$$e_{14} = w_3 x + t_{14}$$
$$e_{24} = w_4 x + t_{24}$$
$$z_{14} = r_3 x + t_{34}$$
$$z_{24} = r_4 x + t_{54}$$
$$z_{34} = (r_5 - w_3 r_4)x + t_{44}$$

9. Finally, the verifier checks the equalities detailed in Figure 9. If these pass, then the proof is verified.

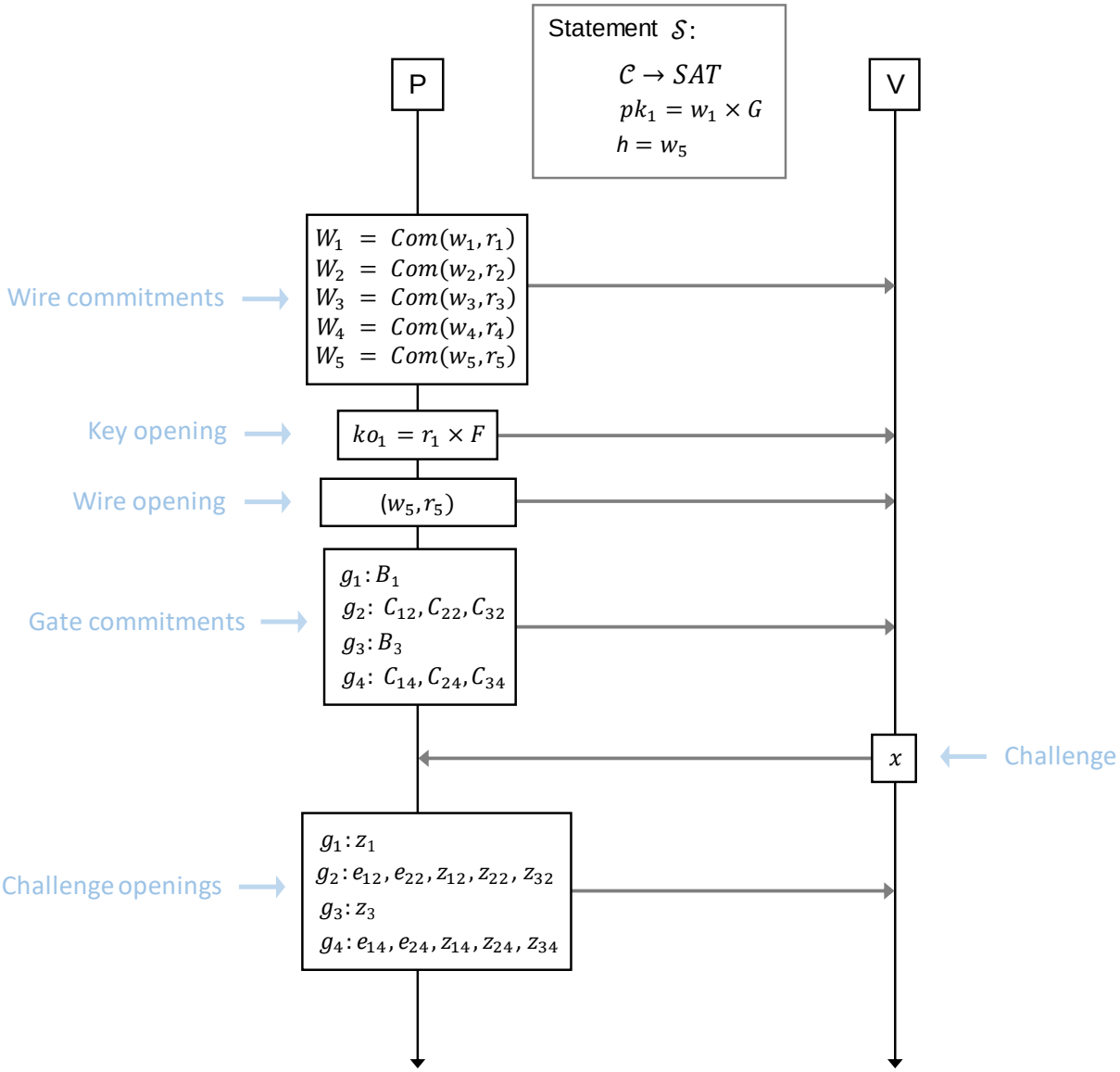

Figure 8. Flow of the interaction between prover and verifier.

Verify:

$\mathcal{C} \rightarrow SAT$:

g_1: $Com(0, z_1) = x \times (W_1 + W_1 - W_2) + B_1$

g_2: $Com(e_{12}, z_{12}) = x \times W_1 + C_{12}$
$Com(e_{22}, z_{22}) = x \times W_2 + C_{22}$
$e_{12} \times W_2 + z_{32} \times F = x \times W_3 + C_{32}$

g_3: $Com(0, z_3) = x \times (W_2 + W_1 - W_4) + B_3$

g_4: $Com(e_{14}, z_{14}) = x \times W_3 + C_{14}$
$Com(e_{24}, z_{24}) = x \times W_4 + C_{24}$
$e_{14} \times W_4 + z_{34} \times F = x \times W_5 + C_{34}$

$$pk_1 = W_1 - ko_1$$

$$W_5 = h \times G + r_5 \times F$$

Figure 9. The checks required to verify the circuit is satisfied (grey box), that wire 1 has the required public key and that wire 5 hash the required value.