Cross Chain Atomic Swaps in the Absence of Time via Attribute Verifiable Timed Commitments

Yacov Manevich
University of Haifa, Israel
yacov.manevich@gmail.com

Adi Akavia
University of Haifa, Israel
adi.akavia@gmail.com

***Abstract*—A Hash Time Lock Contract (HTLC) is a protocol that is commonly used to exchange payments across different blockchains. Using HTLC as a building block for cross blockchain atomic swaps has its drawbacks: The notion of time is handled differently in each blockchain, be it private or public. Additionally, if the swap ends up aborted, the funds are locked in escrow until the safety timeout expires.**

In this work we formulate a new cryptographic primitive: *Attribute Verifiable Timed Commitment* which enables to prove that a timed commitment commits to a value which possesses certain attributes. Using our cryptographic primitive, we describe a new cross chain atomic swap protocol that operates without blockchain derived time and unlike the state of the art, all parties can instantly abort the swap without waiting for the safety timeouts to expire.

In order to prove in zero knowledge that a secret committed to using a timed commitment has a claimed hash value, we employ the "MPC in the head" technique by Ishai et al. and implement our zero-knowledge proof protocol and evaluate its performance. As part of our techniques, we develop a novel and efficient procedure for integer Lower-Than validation in arithmetic circuits which may be of independent interest.

***Index Terms*—Cryptography, Blockchain, Zero-Knowledge Proofs, Cross Chain Swaps, Multi-Party Computation**

1. Introduction

Blockchains are distributed systems that replicate transactions among distrusting and sometimes opposing parties. While it is common practice to exchange assets or currency among participants of the blockchain, engaging in a similar deal across different blockchains is a more involved matter, in which the exchange needs to take place in both blockchains, and ideally without utilizing a trusted third party. Such a protocol is called a cross chain atomic swap.

1.1. Cross Chain Swaps

An atomic swap protocol ensures that if all parties comply with the protocol, a swap takes place and each party exchanges its asset with what the counter-party offers in return. On the other hand, if some party strays from the protocol then the swap ends up aborted, and the system state is left intact. In blockchains, a common solution for atomic swaps is having each party use a Hash Time Lock Contract.

A *Hash Time Lock Contract (HTLC)* is a technique of payment in which a depositor can pay a recipient funds by placing them in an escrow state that can be claimed by the recipient only if the latter presents a pre-image of a cryptographic hash function in a transaction. To prevent the funds from being held in escrow for ever, each HTLC has a time limit (a timeout) after which the funds are returned to the depositor. We outline the standard technique of performing a cross chain atomic swap using HTLC in Section 3.1 in the preliminaries.

1.1.1. Timeouts. An HTLC has a timeout up to which the funds are locked. This is needed in order to enforce that a party will not be able to spend the funds it deposited in escrow while claiming the funds of the counter-party. A downside of using HTLCs as the building block of atomic swaps is that in case the swap ends up aborted (for example, the initiator of the swap has a change of heart) then the funds of both parties are stuck in escrow until their corresponding timeouts expire.

1.1.2. Blockchain Derived Time. Each blockchain (whether it is permissionless or permissioned) has its own different time management and some like Hyperledger Fabric [12], [34] don't even have such. This poses as an obstacle for using HTLCs as an effective building block for cross blockchain swaps. In particular, in some blockchains it is not possible to make a transaction valid only during a certain time frame, or to derive the time from the blockchain height. Therefore, we ask the following question:

Can we build a protocol for cross blockchain atomic swaps without using the blockchain as a source of time?

We answer the question in the affirmative by introducing a new cryptographic primitive which disposes the need to rely on blockchain derived time in cross chain atomic swaps.

1.2. Our Contribution

1.2.1. AVTC. We introduce a new cryptographic primitive that we call an *Attribute Verifiable Timed Commitment* (AVTC). In layman's terms, a timed commitment [15] is a commitment that can be forcibly opened by the receiver without interaction with the sender (committer) by investing a predefined amount of sequential computation. An AVTC can be thought of an enhanced timed commitment where the receiver can be convinced by the sender that the committed secret possesses some attribute, as part of the reception of the commitment.

Our main contribution is the AVTC cryptographic primitive. To construct an AVTC, we augment the timed commitment with an interactive zero knowledge proof for arbitrary properties of the value that is committed to. Our construction prioritizes minimizing computation time for both the prover and verifier and leverages the ”MPC in the head” technique of Ishai et al [30], and entails a novel technique for Lower-Than validation in arithmetic circuits that doesn't require bit decomposition, which may be of independent interest. We implement the zero-knowledge proof of AVTC in C and evaluate its performance under different security configurations.

1.2.2. Blockchain time-agnostic swap. Our second contribution is a novel cross chain atomic swap protocol that operates without blockchain time, which is made possible by using AVTC in a black box manner. We show that our protocol is not only "blockchain agnostic", but has an additional beneficial property: our protocol allows parties to instantly abort the swap if both parties cooperate, while the current state of the art requires parties to wait until the timeouts expire, keeping their funds idly locked in escrow in the meanwhile. The downside of our protocol is that in case the parties do not cooperate in the abort procedure, they need to invest a moderate (several dozens of minutes to a couple of hours) time period of serial computation, whereas in the standard cross chain swap it is sufficient to wait that same time period.

1.3. Overview of Our Techniques

Our cross chain atomic swap protocol entails replacing the time based reclamation condition of the HTLC with revealing a pre-image of a hash function. More specifically, each party gives the counter-party an AVTC and a hash. The counter-party escrows its funds with the reclamation condition of revealing the pre-image of the hash, which it can open only after a predetermined amount of computation steps. The AVTC primitive ensures that the hash pre-image can indeed be revealed after investing the computation steps, which prevents a malicious party from causing the counter party's funds to be locked in escrow forever.

Our AVTC primitive is realized by the timed commitment technique of Boneh and Naor, coupled with an interactive zero knowledge proof between both parties of the swap. The zero-knowledge proof of the AVTC is based on the ”MPC in the head” technique of Ishai et al. [30]. In the zero-knowledge proof, we perform efficient modular exponentiation by employing a novel technique for LSB extraction, which straightforwardly leads to an efficient comparison protocol. The LSB extraction's salient property is the prover acting as a dealer and distributing among the parties blinded plaintext shares of all bits but the LSB itself.

1.4. Paper Organization

The paper is organized as follows: In Section 2 we discuss related work in cross chain atomic swaps and time based cryptography. In Section 3 we outline the state of affairs regarding Hash Time Lock Contracts, give an introduction to Boneh and Naor's timed commitment scheme, and briefly explain the transformation from multi-party computation to zero knowledge proofs ("MPC in the head"), in sections 4 and 5 we describe our protocol for cross chain atomic swap and explain our proposed AVTC primitive. We conclude our work in Section 6.

2. Related Work

2.1. Blockchain Based Time Derivation

In this section we discuss different approaches for time that exist in blockchains, as well as time-related cryptography.

2.1.1. Permissionless Blockchains. Time in permissionless blockchains is often calculated at a very coarse grained level, a fact that forces the parties to have high timeouts (several hours at best): **Bitcoin:** The median time of the last 11 blocks [4], where a block is created on average once per 10 minutes. **Ethereum:** In Ethereum, the block timestamp is set by the node that mined the block, and the validity rules dictate that the timestamp should be monotonously ascending and each block's timestamp is earlier than 15 minutes into the future from the current time.

2.1.2. Permissioned Blockchains. Permissioned blockchains block production rate varies according to the transaction load [34], therefore it is impossible to correlate a number of blocks produced with time. There are no standards of how the notion of time is defined in permissioned blockchains: **Tendermint**: In Tendermint [10], applications can define a "locktime" which is the number of blocks passed after contract creation. However this compromises the safety in the event of an system outage in which no blocks are being created. **Hyperledger Fabric**: Blocks do not carry timestamps [12]. Mitra et al. [34] proposed a mechanism to assign timestamps to the blocks by modifications to post consensus validation. **Sawtooth**: In Hyperledger Sawtooth [8], there is no block timestamp by default, but the nodes can be configured to artificially inject a "Blockchain Info" transaction that contains a timestamp, at the beginning at every block (see [9]).

2.1.3. Scientific Literature. There have been several proposals for cross chain swaps that do not fully depend on blockchain based time derivation. Deneuville et al. [47] proposed an asymmetric protocol in which one blockchain can verify transactions being put on the second blockchain, however, one of the blockchains still relies on blockchain derived time. Interledger [6] proposed a protocol based on connector entities that have access to both blockchains, and is also time reliant for escrow release.

2.2. Time Based Cryptography

The main time-based cryptographic primitive that is used in our work, is the ”Timed Commitment” of Boneh and Naor [15], and is explained in Section 3.2 . We give an overview on other cryptographic primitives that possess time-related guarantees.

2.2.1. Verifiable Delay Functions.

Proofs that verify a computation took a certain amount of steps have been studied in the past in the context of Verifiable Delay Functions (VDF) [14] [38] [46].

Replacing AVTC with a VDF is possible, however it would take away an important advantage of our protocol: interoperability. In our protocol, the computations done in the blockchain are invocations of hash functions, and the heavy lifting is done in the application layer. In contrast, using a VDF would require the blockchain software to be able to validate the VDF proof, which requires introducing new software into the blockchain validation mechanisms, thus hindering adoption and making it impossible to be integrated to conservative blockchains.

2.2.2. Timed Fair Exchange of Signatures.

Garay et al. [22] and Boneh at al. [15] both proposed interactive protocols for timed fair exchange of signatures. Those protocols, however, do not protect the counter-party from a malicious abort that wastes computation time that is linear in the total protocol runtime.

2.2.3. Multi-Party Timed Commitment.

Doweck and Eyal [21] constructed a *Multi-Party Timed Commitment (MPTC)* that enables several parties to jointly commit to a secret to be later opened by a designated party via brute-force computation. The MPTC primitive of [21] does not allow one to prove an attributes the committed secret, unlike our AVTC primitive. The MPTC commitment brute-force opening is trivially parallelizable, unlike our AVTC primitive which is not (under number-theoretic assumptions, see Definition 5). Consequently, we note that in our cross-chain atomic swap protocol, our AVTC primitive cannot be substituted with an MPTC because we require that the commitment brute-force opening cannot be done via parallel computation, as it would harm the security of our atomic swap protocol.

2.2.4. Gage Time Capsules.

Almashaqbeh, Benhamouda, et al. [11] constructed a *Gage Time Capsule (GaTC)* which allows a party to commit to a value that others are able to reveal at a designated total computational cost, and that computation is expected to be parallelized among different parties which are incentivised to try out different random decommitment values, and claim a monetary reward in exchange for their work. While their primitive (GaTC) seems similar to our AVTC primitive, we note the salient differences:

- Brute-forcing a GaTC can be parallelized, and thus, parties with access to parallel computing power gain significant advantage over others. In contrast, brute-forcing an AVTC cannot be parallelized (under number-theoretic assumptions, see Definition 5), and this property is crucial to the security of our novel cross-chain atomic swap protocol described in Section 4.
- The GaTC zero-knowledge proof does not prove any attribute of the secret that is committed to. Rather, it contains only a proof of opening. In contrast, our AVTC primitive, proves that a committed secret, possesses an attribute of the committer's choice.
- The GaTC committer does not prove to a counter-party that the commitment can be opened within a timely manner. Rather, it assumes a semi-honest model on the committer of the GaTC. In contrast, our AVTC primitive only assumes number theoretical assumptions (see Definition 5), and protects against malicious committers by enforcing honest behavior via zero-knowledge proofs. We note that the authors of [11] mention this and state that their semi-honest committer assumption can be eliminated by using zero-knowledge proofs.
- A party that opened a GaTC can prove in zero-knowledge that it did so. While our technique doesn't focus on such an ability, we note that our technique can be trivially extended to facilitate that. Indeed, a party that has forcefully extracted the input for $\mathcal{F}$ (as depicted in **Figure 2**) has the required information to play as the prover of the AVTC.

2.2.5. Homomorphic Time-Lock Puzzles (HTLP).

Malavolta et al. [33] constructed variations of time-lock puzzles that each is homomorphic to an operation[1] such as addition in $\mathbb{Z}_N$ or multiplication in $\mathbb{J}_N$ (where $\mathbb{J}_N$ denotes elements in $\mathbb{Z}_N^*$ with Jacobi symbol +1).

As in the timed commitment, the $HTLP$ has a setup phase that generates public parameters. However, in the timed commitment the sender proves to the receiver that the forced open (denoted as $Solve$ in HTLP) indeed yields the committed value within the claimed number of computation steps. As we'll see below, it is not the case in HTLP. We elaborate on one of the variations of HTLP, Linearly Homomorphic Time Lock Puzzle (LHTLP) as it is used as a building block for higher level applications which we discuss later in this section. A Linearly Homomorphic Time Lock Puzzle (LHTLP) is a tuple of the four algorithms below:

- $pp \leftarrow Setup(1^\lambda, T)$ On input of security parameter λ, hardness parameter T outputs public parameters pp.
- $Z \leftarrow Gen\,(pp, s)$: Output a puzzle Z for a solution s.
- $s \leftarrow Solve\,(pp, Z)$: Recovers s from Z.
- $Z \leftarrow Eval\,(\oplus, pp, Z_1, ..., Z_n)$: Outputs Z s.t $\forall \{Z_i \leftarrow Gen\,(pp, s_i)\}_{i=1}^n$: $Solve\,(pp, Z) = \sum_{i=1}^n s_i\,(mod\ N)$ where N is specified in pp.

Setup samples a safe modulus[2] N, samples $g \in \mathbb{J}_N$ and computes $h = g^{2^T}$ and outputs $pp = (T, N, g, h)$. If h was not honestly constructed then $Solve$ will not output the s that was used as input to Gen. We elaborate on this in Section A in the Appendix. The authors of [33] do mention that the setup should be executed by a trusted party.

2.2.6. Verifiable Timed Signatures.

Thyagarajan et al. [44] constructed a *Verifiable Timed Signature (VTS)* primitive for common elliptic curve based signature schemes such as ECDSA [31]. It allows a sender to commit to a receiver on a signature and later the receiver can forcefully recover the signature without interaction with the sender by executing serial computation. Unlike our AVTC primitive (Definition 10), the VTS construction is an ad-hoc protocol suitable only for signatures, and cannot be used for any arbitrary attribute as in our AVTC primitive.

The VTS construction is based on the Linearly Homomorphic Time-Lock Puzzle (LHTLP) primitive of Malavolta et al. [33] and its high-level idea is as follows: First,

1. The work also shows that a fully homomorphic time lock puzzle is feasible assuming indistinguishability obfuscation.

2. A safe modulus is a product of two primes $p = 2p'+1$, $q = 2q'+1$ where p', q' are also prime

the sender ”encrypts” the signature with secret shares generated with t-out-of-n Shamir’s secret-sharing [42]. Then, it commits to each share with the LHTLP primitive by locking each share in a LHTLP puzzle. The sender and the receiver then engage in a cut-and-choose protocol where the receiver asks to open $t-1$ puzzles and invokes Gen itself to verify the puzzles sent to it were indeed correctly constructed. Additionally there is also a range proof that proves that the time-lock puzzles contain ”small” values.

As noted in our discussion on the Homomorphic Time-Lock Puzzle (HTLP), without a trusted setup assumption, even when validating the Gen operation via cut-and-choose as done in the VTS construction, a malicious sender may fool the receiver into accepting signatures that can never be opened without interaction with the sender. In contrast, our AVTC primitive enjoys a transparent setup.

2.2.7. Universal Atomic Swaps: Secure Exchange of Coins Across All Blockchains. A parallel work to ours by Thyagarajan et al. [43] that appeared online in December 2021 deals with performing cross-chain atomic swaps by substituting blockchain derived time with using the Verifiable Timed Signature (VTS) (Section 2.2.6) also by Thyagarajan et al. [44]. The high level idea of their scheme is that the escrow account on each blockchain is initialized with a public key so that its corresponding private key is secret shared between the two counter-parties. Then, to swap the funds the parties engage in a secure two-party computation protocol to produce a signature (done for each blockchain in parallel). To withdraw from the swap, the parties recover signatures on withdraw transactions prepared as part of the prepare phase.

We note that unlike our solution which relies on the blockchain being able to verify hash puzzles as part of its transaction validation logic (an ability that some blockchains like Monero, lack), the technique of Thyagarajan et al. does not require that, as the time based cryptography locks a signature and not a hash pre-image. It is not clear how the authors address the issue we raise in Section 2.2.5 regarding the trusted setup assumption of the HTLP and VTS primitives that the construction of [43] relies on.

2.3. LSB extraction in Multi-Party Computation

The AVTC primitive we construct uses the ”MPC in the head” zero-knowledge framework (see Section 3.3). An important part of our computation uses LSB extraction of a secret-shared value.

The current state of the art in MPC enables one to extract the LSB [19], [32], [37] but with polynomial complexity in the binary representation length of elements in the underlying field. Such techniques are expensive to be used in the ”MPC in the head” approach that we employ for our zero-knowledge proof, since they blow up the proof size when the field in use is thousands bits long. There are also techniques [18], [40] that allow to extract the LSB with a constant communication complexity, however only for values *known* to be significantly small. In our protocol, unfortunately, the verifier cannot trust the prover to uphold any such invariant.

We devise a novel MPC protocol tailored for zero-knowledge from MPC techniques that extracts an LSB in constant complexity, by making the prover compute the LSB itself, but having the verifier validate the prover.

3. Preliminaries

3.1. Hash Time Lock Contracts

A Hash Time Lock Contract (HTLC) [39] is a technique of payment in which a *depositor* can pay a *recipient* a part of its funds by placing them in an escrow state that can be claimed by the recipient only if it presents a pre-image of a cryptographic hash function in a transaction. To prevent the funds from being held in escrow for ever, each HTLC has a time limit (a timeout) after which the funds are returned to the depositor.

HTLCs are often used as a building block for atomic cross chain swaps, in which Alice can atomically exchange her coins in blockchain A with Bob’s coins in blockchain B. In order for a cross chain atomic swap to be useful, it needs to possess both *safety* and *liveness* properties [29]:

Definition 1 (Safety)**.** For every execution, either both parties claim each other’s funds, or no party claims the counter-party’s funds.

Definition 2 (Liveness)**.** For every execution, no asset of some party is locked in escrow forever.

Below we give the well known HTLC based cross chain atomic swap protocol of [28]:

1) Alice randomly selects a secret $s \leftarrow \{0,1\}^\lambda$ for some security parameter λ and publishes a transaction on blockchain A with a hash lock of $h = \mathcal{H}(s)$ and a time lock of $T + \Delta$ for some $\Delta > 0$.
2) Bob publishes a similar transaction on blockchain B with a hash lock of h (while not knowing s) and a time lock of T.
3) Alice publishes a transaction on blockchain B which includes $s' \in \mathcal{H}^{-1}(h)$ and as a result claims Bob’s coins to her account.
4) Bob, having seen a pre-image s' such that $\mathcal{H}(s') = h$, publishes a transaction on Alice’s blockchain that claims Alice’s coins to his account.

We note that Alice sets a time lock T+Δ compared to T for Bob, because had Alice put her funds in escrow for an equivalent or shorter period of time than Bob - she could have first claimed Bob’s funds into her account and then raced Bob on blockchain A and reclaimed her funds back before Bob had a chance to claim them.

While HTLCs are powerful tools that allow atomic exchange of assets between two or more blockchains, a major limitation in the protocol stems from its safety: If a party put its funds in escrow for a time period T, even if both parties agree to abort the swap, they have to wait until T time passes.

3.2. Timed Commitments

Notations and definitions: We denote by $\xleftarrow{R} \mathbb{X}$ a uniform random selection from a set $\mathbb{X}$ and denote $\lambda \in \mathbb{N}$ to be a security parameter. Let $\epsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a negligible function if for every positive polynomial P: $\exists N \in \mathbb{N}\ s.t\ \forall n > N:\ \epsilon(n) < \frac{1}{P(n)}$.

Definition 3 (Commitment scheme)**.** A protocol between a sender S and a receiver R with two steps:

- $Commit(x, 1^\lambda)$: S has input x, and outputs to R a string $C \in \{0,1\}^\lambda$ and to S a decommitment string $d \in \{0,1\}^\lambda$. We denote by $(d, C) \leftarrow < S(x, 1^\lambda), R(1^\lambda) >$ the experiment in which S and R interact.
- $Open$: R has C and is given x, d by S. R accepts or rejects. We denote by $R(C, x, d) \in \{0,1\}$ its output.

A commitment scheme has two security properties:

- Hiding: For any receiver R which chooses $\{x_0, x_1\} \subset \{0,1\}^\lambda$ and is given $C = Commit(x_b)$ on $x_b \xleftarrow{R} \{x_0, x_1\}$ and outputs $b = 0$ or $b = 1$, it holds that: $Pr\left[R\left(x_0, x_1, 1^\lambda\right) = b\right] - \frac{1}{2} = \epsilon(\lambda)$ for a negligible ϵ.
- Binding: For any $C \in \{0,1\}^\lambda$ there is at most one $x \in \{0,1\}^\lambda$ such that $\forall x' \neq x$, $\forall d'$: $Pr[R(C, x', d') = 1] = \epsilon(\lambda)$ for a negligible ϵ.

Dan Boneh and Moni Naor [15] describe an enhanced commitment primitive called a "Timed commitment" that has a third step:

- *Forced Open(C)*, in which the receiver can invest moderate computation power and time in order to open the commitment C to yield x without any interaction with the sender. The number of computation steps required to execute the forced open step is set by the committer as part of the commit step.

We formally define a timed commitment below:

Definition 4 (Timed Commitment [15])**.** A *timed commitment scheme* $\langle Commit, Open, FOpen\rangle$ is a triplet of PPT algorithms that satisfy: Given a string $x \in \{0,1\}^l$, a difficulty level $k \in \mathbb{N}$, a committer computes a cryptographic commitment $(\mathcal{C}, d) \leftarrow Commit\left(x, k, 1^\lambda\right)$ and convinces the receiver that $\mathcal{C}$ can later be opened without interaction with the committer via $m \leftarrow FOpen(\mathcal{C}, k)$ by executing 2^k *serial* computation steps [3], or by having x revealed to the receiver via $m \leftarrow Open(\mathcal{C}, x, d)$.

The timed commitment primitive, possesses the additional following two properties:

- **Soundness**: The committer proves to the verifier that the forced open step indeed yields the committed value within the claimed number of computation steps.
- **Serial computation**: No parallel algorithm can obtain information about the committed string in time significantly less than it takes to compute successive squarings. In particular, it relies on the *Generalized BBS assumption*.

Definition 5. *Generalized BBS assumption* [15] For $k \in \mathbb{N}$ and $g \xleftarrow{R} \mathbb{Z}_N$, let $W_{g,k} = \left\langle g^2, g^4, ..., g^{2^{2^k}}\right\rangle$. The element $g^{2^{2^{k+1}}}$ is computationally indistinguishable from a random quadratic residue for any parallel random access machine whose running time much less than 2^k.

Aside from the Generalized BBS assumption needed for the serial computation constraint, the hiding property relies on the Quadratic residuosity assumption [13].

3.2.1. The timed commitment protocol. We next elaborate on the timed commitment protocol as it appears in [15]. We also include a full proof for the zero-knowledge proof sub-protocol of timed commitment in Section C, which was not provided in [15].

3. The value k for $k \in \mathbb{N}$ is a difficulty parameter chosen by the committer.

Setup(1^λ)**.** Let λ be a security parameter. The committer (also called the sender) picks P, Q: two λ-bit primes s.t $P \equiv Q \equiv 3 \pmod 4$, and publishes their product $N = P \cdot Q$ to the receiver.

Commit(M, T)**.** The Committer (sender) receives as input $T = 2^k$ for some $k \in \mathbb{N}$ that determines the difficulty of the forced open phase by the receiver, and a message $M \in \{0,1\}^*$ to commit to.

1) The sender picks $h \leftarrow \mathbb{Z}_\lambda$ at random
2) The sender computes $g = h^{(\Pi_{i=1}^{r} q_i^n)} mod\, N$ where $q_1, q_2, .., q_r$ are the first r primes, with a typical setting of r being 128.
3) The sender sends the receiver h, g and the latter verifies them by computing $\mathcal{Q} = \Pi_{i=1}^{r} q_i^\lambda$ and then $g = h^{\mathcal{Q}} \,(mod\ N)$
4) The sender, knowing P, Q computes $u = g^{2^{2^k}} mod\, N$ by computing $a = 2^{2^k} mod\, \phi(N)$ followed by $u = g^a (mod\, N)$.
5) The sender hides M's bits by XOR-ing them with $LSBs$ of successive roots of $u\, mod\, N$, more formally: $S_i = M_i \oplus g^{2^{2^k - i}} mod\, N$ where M_i denotes the i-th bit of M. The commitment string is defined as:
$$C = \langle S, h, g, u\rangle$$
and is sent to the receiver.
6) The sender proves that u is constructed properly, meaning that the receiver can indeed open the commitment in time T by computing repeated squaring modulo N starting from g and until $u = g^{2^{2^k}}$. This is done by constructing the following vector W of length k:
$$W = \left\langle g^2, g^4, g^{16}, g^{256}, ..., 2^{2^{2^i}}, ..., g^{2^{2^k}}\right\rangle (mod\, N)$$
followed by proving in Zero Knowledge for each $i \in [k]$ that for each adjacent pair $(a_i, b_i) \in W$: (g, a_i, b_i) is of the form (g, g^x, g^{x^2}) for some x. This protocol, as well as its proof, can be found in Section B.

Open. Denote $|M|$ as the bit length of M. The sender sends the receiver $v' = h^{2^{2^k - |M|}}$ and the receiver computes $v = (v')^{\Pi_{i=1}^{r} q_i^\lambda}$ which is the $2^{|M|}$'th root of u (taking square roots $|M|$ times from u yields v). Now the receiver can compute: $[M]_i = [S]_i \oplus \left[v^{2^{|M|-i}} mod\, N\right]$ by repeated modular squaring starting from v.

Forced-Open(C)**.** In case the sender never opens the commitment for the receiver, the receiver can compute (with some effort) the committed value M by first computing $v = g^{2^{2^k - |M|}} (mod\, N)$ by using $2^k - |M|$ squaring modulo N to obtain v and proceed as in the open step.

3.3. Zero Knowledge Proofs From Secure Multiparty Computation

We assume the reader is familiar with secret-sharing based multi-party computation schemes such as GMW [26], namely that multiplication a scalar and addition can be done by every party as a local computation, but multiplication, secret share reconstruction and distribution requires communication among parties.

Ishai, Kushilevitz, Ostrovsky and Sahai proved in [30] that Zero Knowledge Proofs can be built from semi-honest MPC protocols in the following manner: Let $L \in NP$ and R_L its corresponding witness relation. Given $x \in L$ and a witness w such that $(x, w) \in R_L$ and an MPC protocol that computes $R_L(x, w) \in \{0, 1\}$, one can build a zero knowledge proof where the prover proves in zero knowledge that indeed $x \in L$ as follows:

1) Let $P_1, ..., P_n$ be n parties. The prover splits w into n random shares w_i such that $w = \bigoplus_{i\in[n]} w_i$ and assigns each party i private input w_i.
2) The prover simulates the MPC protocol for all n parties "in the head" and maintains records of views and message transmissions for all parties.
3) The prover commits to the views and message transmissions using a statistically binding commitment and sends them to the verifier.
4) The verifier randomly asks the prover to reveal the commitments for a subset of the parties.
5) The prover complies and reveals the requested information.
6) The verifier validates the commitments revealed and checks the consistency of the views and message transmissions, and accepts or rejects accordingly.

This result has a strong implication: Given $(x, w) \in R_L$, the construction of a multiparty protocol where w is secret shared among the parties and the circuit checks whether $(x, w) \in R_L$ is a scheme to construct a zero knowledge proof for membership in L.

4. Cross Chain Atomic Swaps in the Absence of Time

We describe a novel protocol for cross chain atomic swaps that utilizes our Attribute Verifiable Timed Commitment (AVTC) primitive. Unlike the state of the art, our protocol operates without blockchain derived time.

Our protocol's main difference from the standard cross chain atomic swap described in the literature in Section 3.1 is that we have each party give the other party a hash puzzle that can be opened in a timely manner. Each party, escrows its funds in a way that reclaims the funds back to the party only if the hash puzzle solution is revealed.

By changing the reclamation condition to a hash verification, we also give the parties a way to immediately reclaim their funds in escrow. While it is possible to construct a condition with the state of the art's HTLC based swap, it would require *adding* a multi-signature condition to the script which would take more space and computation overhead, hence will increase the cost of using such a transaction.

4.1. AVTC as a Time Based Hash Puzzle

Consider a puzzle encoded as a string y so that to solve it, one needs to find a hash pre-image x such that $y = \mathcal{H}(x)$ where $\mathcal{H}$ is a cryptographic hash function. A time based hash puzzle is a puzzle that has an additional way of finding the pre-image x, and this way requires a certain number of computation steps. Indeed, a time based hash puzzle would fit our needs perfectly: Verifying a solution of a time based hash puzzle requires only a single hash invocation on the candidate solution. In addition, a hash puzzle takes up minimal storage space in the blockchain.

Our realization of a time based hash puzzle is via our Attribute Verifiable Timed Commitment (AVTC) primitive that enables a sender to commit to a value M and to prove to a receiver that the message M committed to is a hash pre-image of a specified hash. The receiver can open the commitment without interaction with the sender by executing a predefined number of computation steps (where the number of steps is set by the sender). We define the AVTC time based hash puzzle here, and in Section 5 we generalize the AVTC primitive to any functionality.

Definition 6. AVTC Timed Based Hash Puzzle Let λ be a security parameter and $\mathcal{H}$ be a cryptographic hash function. An AVTC time based hash puzzle is defined by four operations:

1) $Pub, Priv \leftarrow Setup(1^\lambda)$: The sender obtains private parameter $Priv$, both sender and receiver obtain Pub.
2) $(\mathcal{C}, y, \pi) \leftarrow Commit(Priv, T, M)$: Given the setup's private parameters, a difficulty level T and a message M, the sender produces a commitment $\mathcal{C}$, a string $y = \mathcal{H}(M)$ and a proof π that $y = \mathcal{H}(M)$.
3) $M \leftarrow Open(\mathcal{C}, Pub)$: The sender reveals the message M to the receiver.
4) $M \leftarrow ForceOpen(\mathcal{C}, Pub)$: The receiver reveals the message M without interaction with the receiver, by investing T computation steps.

Security model and assumptions. We model the world as an interaction between two mutually distrusting and potentially malicious parties. Each party has an incentive to deceive the counter-party and has no repercussion if caught cheating. We further assume that a party cannot deny the counter-party from transacting on the blockchain indefinitely, but only for a certain known time period. We make the standard requirement that our cryptographic hash function is pre-image resistant [41]. We assume the parties are probabilistic polynomial time Turing machines.

4.2. The Cross Chain Atomic Swap Protocol

We lay out our protocol by describing an interaction between two parties: Alice and Bob, where Alice pays Bob a fee sum on blockchain BC_A in exchange for Bob paying Alice sum on BC_B . The protocol consists of two phases:

1) **Prepare:** Puts transactions on the blockchains in preparation for the swap.
2) Either **Swap:** Transact on both blockchains for each party to claim its payment, or **Withdraw**: Cancel the swap and have each party reclaim the escrow.

Notations: We next elaborate on each of these steps. Let λ be a security parameter and $T = 2^d$ be a difficulty level, typically $d \in [30, 50]$. We denote appending transaction X to blockchain BC_A as $BC_A \Vdash X$. We denote Alice sending a message m to Bob as $A \overset{m}{\to} B$ and similarly, Bob sending m to Alice as $B \overset{m}{\to} A$. For succinctness, we omit specifying Pub and $Priv$ produced in the Setup phase.

Prepare. In steps 1-2 in the prepare phase (depicted in **Figure 1**), Alice produces a timed commitment $\mathcal{C}_A$ for a uniformly random hash pre-image a, and sends the other party (Bob) $\mathcal{C}_A$ and the hash $A = \mathcal{H}(a)$ along with proof that $A = \mathcal{H}(\mathit{ForceOpen}(\mathcal{C}_A))$. The proofs are verified by both parties in step 3, and each party aborts the protocol if the proof verification fails. Then, in step 4 Alice initiates the HTLC by selecting a uniformly random secret x and publishing its hash y in step 5 on BC_A by publishing a transaction that guarantees:

1) If Bob appends a transaction that reveals $\mathcal{H}^{-1}(y)$ before Alice appends a transaction that reveals $b' \in \mathcal{H}^{-1}(B)$ then Bob claims sum to itself on BC_A.
2) If Bob hadn't appended a transaction that reveals $\mathcal{H}^{-1}(y)$ and Alice appended a transaction that revealed $b' \in \mathcal{H}^{-1}(B)$ then Alice claims back sum on BC_A.

Finally, in step 6, Bob finishes the prepare step by appending a transaction on BC_B which guarantees:

1) If Alice appends a transaction which reveals $\mathcal{H}^{-1}(y)$ before Bob appends a transaction which reveals $a' \in \mathcal{H}^{-1}(A)$, then Alice claims sum to herself on BC_B.
2) If Alice did not append a transaction which revealed $\mathcal{H}^{-1}(y)$ and Bob appended a transaction which revealed $a' \in \mathcal{H}^{-1}(A)$, then Bob claims back sum on BC_B.

Swap. Alice appends a transaction to BC_B which reveals x and thus claims sum from Bob, just as done in HTLC. Bob, having now seen x s.t $x \in H^{-1}(y)$, appends a transaction that reveals x to BC_A and thus claims sum from Alice.

Withdraw. As in the standard HTLC based cross chain swap, since Alice initiated the swap she sampled x (line 4 in **Prepare**) and therefore in order to withdraw from the swap, Alice must not commence the **Swap** phase.

The withdraw procedure can be cooperative where the parties exchange information, or non-cooperative where no information exchange takes place.

Cooperative withdraw. If one of the parties decides to abort the swap, first Alice reveals a to Bob and then Bob reveals b to Alice. Each of the parties can reclaim its escrow funds by transacting on its blockchain and revealing the pre-image a or b.

Non-cooperative withdraw. If either party wants to withdraw the funds in escrow without consent from the counter-party, it needs to recover b (for Alice) or a (for Bob) by computing sequentially for a time period that is $T + \Delta$ for Alice, and T for Bob.

Realistic parameter selection. The difficulty level T that Commit is executed with corresponds to some serial execution time. Setting $T = 2^{30}$ amounts to ~ 20 minutes on a modern Intel i7 CPU. The execution time of $\mathit{ForceOpen}$ should be carefully picked according to transaction confirmation time and its correlated probability of transaction reversal [23]. If the confirmation time is lower than the time it takes to compute $\mathit{ForceOpen}$ then the safety of our protocol is impaired

The difficulty level T of the time based hash puzzle should be large enough so that after both Alice

Prepare

```
1: Bob: $b \xleftarrow{R} \{0,1\}^\lambda,\ B = \mathcal{H}(b),$
   $(\mathcal{C}_B, \pi_B) = \mathit{Commit}(b, T+\Delta),\ B \xrightarrow{(B,\mathcal{C}_B,\pi_B)} A$
2: Alice: $a \xleftarrow{R} \{0,1\}^\lambda,\ A = \mathcal{H}(a),$
   $(\mathcal{C}_A, \pi_A) = \mathit{Commit}(a, T),\ A \xrightarrow{(A,\mathcal{C}_A,\pi_A)} B$
3: Alice: Verify $\pi_B$; Bob: Verify $\pi_A$
4: Alice: $x \xleftarrow{R} \{0,1\}^\lambda,\ y = \mathcal{H}(x)$
5: Alice: $BC_A \Vdash \langle B, y, sum \rangle$
6: Bob: $BC_B \Vdash \langle A, y, sum \rangle$
```

Swap

```
1: Alice: $BC_B \Vdash \langle x \rangle$
2: Bob: $BC_A \Vdash \langle x \rangle$
```

Withdraw(cooperative)

```
1: Alice: $A \xrightarrow{a} B$
2: Bob: $BC_B \Vdash \langle a \rangle$
3: Bob: $B \xrightarrow{b} A$
4: Alice: $BC_A \Vdash \langle b \rangle$
```

Withdraw^{Alice}(non-cooperative)

```
1: Alice: $b \leftarrow \mathit{ForceOpen}(\mathcal{C}_B)$
2: Alice: $BC_A \Vdash \langle b \rangle$
```

Withdraw^{Bob}(non-cooperative)

```
1: Bob: $a \leftarrow \mathit{ForceOpen}(\mathcal{C}_A)$
2: Bob: $BC_B \Vdash \langle a \rangle$
```

Figure 1: The phases of our cross chain atomic swap

(and Bob) put their hash puzzles (lines 5-6 in **Prepare**) on blockchain A (and B), Bob cannot race Alice and forcefully open a and withdraw his funds in escrow on blockchain B before Alice's swap transaction (line 1 in **Swap**) is confirmed on blockchain B. Otherwise, once Alice releases the swap transaction that reveals x in her attempt to claim Bob's funds in escrow on blockchain B, Bob can claim Alice's tokens on blockchain A while preventing Alice from claiming his funds on blockchain B. Similarly, if Δ in the difficulty level $T+\Delta$ is too small, if the swap wouldn't end up taking place and Bob will try to withdraw his funds in escrow, Alice might forcefully recover b and reclaim her funds in escrow on blockchain A and race Bob's withdraw transaction.To guarantee Bob has enough time for his withdraw transaction to be confirmed on blockchain B, Δ should be large enough.

Safety boundaries in a mutually distrusted setting. Whether the swap ends up taking place or not, it is possible to safely execute our protocol without having to execute $\mathit{ForceOpen}$. We show that by setting $\Delta > T$ and showing a schedule where none of the parties execute the $\mathit{ForceOpen}$ and argue why the safety property

(Definition 1) is preserved.

0: Bob and Alice execute steps 1-3 in **Prepare** (nothing is written on chain yet)

$\frac{\Delta}{2}$: If Alice's steps 4 and 5 in **Prepare** occur later than $\frac{\Delta}{2}$ then Bob refuses to proceed to step 6 in **Prepare** and Alice never claims his funds.

$\frac{3T}{4}$: If Bob's step 6 in **Prepare** occurs later than $\frac{3T}{4}$ then Alice refuses to proceed to **Swap** and Bob never claims her funds. Else, Alice proceeds to step 1 in **Swap** and claims Bob's funds before time T (where he can reclaim them).

T: If Alice and Bob decide to withdraw, then as per step 1 in **Withdraw(cooperative)**, Alice gives Bob a first. If Alice fails to do so before time T then Bob can start executing **Withdraw(non-cooperative)** which ends before $T + \Delta$ (since $\Delta > T$), hence if Alice chooses to perform **Swap** during that time, she cannot also withdraw her funds in escrow.

4.3. Cross Chain Atomic Swap Protocol Analysis

We prove that our protocol satisfies the *safety* and *liveness* properties from the preliminaries:

Theorem 7 (Safety)**.** *Either both parties claim each others' funds, or no party claims the counter-party's funds.*

Proof. Assume in contradiction that one of the parties claims the counter-party's funds, and split to cases:

1) **Alice claims Bobs' funds**: Claiming Bob's funds involves in revealing $H^{-1}(y)$ to Bob. If Bob couldn't claim Alice's funds on BC_A then it is due to Alice forcefully opening Bob's timed commitment to obtain b, however she needs twice the time than Bob to open his commitment to b, hence Bob had sufficient time to open Alice's commitment to a and reclaim back his funds, preventing Alice from claiming his funds $\Rightarrow$ in contradiction to Alice claiming Bob's funds.
2) **Bob claims Alice's funds**: To claim Alice's funds, Bob has to see $H^{-1}(y)$ which is only revealed by Alice as part of her claiming Bob's funds. If Alice reveals $H^{-1}(y)$ early enough (before Bob has time to forcefully open Alice's commitment and extract a) then it is too late because Alice already claimed his funds.

□

Theorem 8 (Liveness)**.** *No asset of some party is forever locked in escrow.*

Proof. Either no party aborts, and then the swap completes, or some party aborts. If some party aborts, then both parties can reclaim their funds in escrow after a period of time: If Alice decides to abort after the prepare phase, then she can open her commitment to reveal a to Bob which makes him reclaim his funds in escrow, after which Bob can reveal b to Alice to make her reclaim her funds as well. In any case if any of the parties refuses to reveal the commitment, the counter-party can forcefully open it by investing a moderate computation time and thus reclaim back the funds in escrow. □

4.3.1. Calculating the Commitment String Length.

An important question that comes to mind, is, how long should be the hash pre-image prefix committed to by the AVTC primitive? However, a hash can be brute-forced with parallel computation. We show in Section F that having 128 bits of pre-image prefix thwarts even an adversary with hashing power equivalent to the entire Bitcoin network.

5. Attribute Verifiable Timed Commitment

Recall, the timed commitment scheme of Boneh and Naor (Section 3.2) allows Alice to commit to a secret, and to prove to Bob that the commitment can be opened by performing a certain amount of computation.

In this section, we show a novel cryptographic primitive that allows Alice to prove to Bob arbitrary attributes of the secret in zero knowledge. That is, the committer (Alice) plays the role of a prover, and the receiver (Bob) of a verifier. When enhancing the timed commitment with our cryptographic primitive, the receiver accepts the commitment only if the secret committed to possesses attributes agreed by both parties. Hereupon we denote our novel primitive $AVTC$ which stands for *Attribute Verifiable Timed Commitment*.

In Section 5.1 we formally define what the AVTC primitive entails. Then in Sections 5.2 and 5.3 we dive into the construction of the components of the AVTC primitive and analyze its security. In Section 5.4 we discuss evaluation of our publicly available implementation.

5.1. Definition of AVTC

In this section we give our definition for the AVTC primitive. The Timed Commitment satisfies both the hiding and binding properties of a cryptographic commitment scheme (see Section 3.2 in preliminaries), with the exception that it holds only for adversaries limited in the number of their execution steps to be 2^k.

Definition 9 (Values that satisfy $\mathcal{F}$)**.** Let $\mathcal{F} : \{0,1\}^* \rightarrow \{0,1\}$ be a predicate. We define the language which informally means "all values which satisfy $\mathcal{F}$" as $L_{\mathcal{F}} = \{x \mid \mathcal{F}(x) = 1\}$.

An AVTC for a predicate $L_{\mathcal{F}}$ is a timed commitment augmented with a zero knowledge proof that the committed value satisfies F

Definition 10 (Attribute Verifiable Timed Commitment (AVTC) for $\mathcal{F}$)**.** Let $\mathcal{F} : \{0,1\}^* \rightarrow \{0,1\}$ be a predicate. An AVTC for $L_{\mathcal{F}}$ is a triplet of algorithms: $\langle Commit, Open, FOpen\rangle$ that are as in the Timed Commitment (Definition 4), but with an enhanced $Commit$ operation that also guarantees that given a commitment $\mathcal{C} \leftarrow Commit\left(x, 1^{\lambda}\right)$ accepted by the receiver, it holds that $x \in \mathcal{L}_{\mathcal{F}}$. Denote the probability that the receiver R accepts that $\mathcal{C}$ commits to $x \in L_{\mathcal{F}}$ as $Pr\left[\langle S(x), R(\mathcal{C})\rangle = 1\right]$. Let λ be a security parameter and ϵ be a negligible function, and denote $View_R^S(\mathcal{C})$ the view of the receiver with input $\mathcal{C}$ engaging with the sender. The following properties hold:

- **Completeness**: Let $x \in L_{\mathcal{F}}$. Then:
  $Pr\left[\langle S(x), R(\mathcal{C})\rangle = 1\right] = 1$.
- **Soundness**: Let $x \notin L_{\mathcal{F}}$. Then:
  $\exists \epsilon$ s.t: $Pr\left[\langle S(x), R(\mathcal{C})\rangle = 1\right] < \epsilon(\lambda)$.
- **Zero Knowledge**: For every receiver R: $\exists$ PPT simulator $\mathcal{M}$ such that $\forall x \in L_{\mathcal{F}}$: $View_R^S(C) \equiv_s View_R^{\mathcal{M}}(\mathcal{C})$.

5.2. AVTC Construction Overview

We prove the following theorem:

Theorem 11 (AVTC)**.** *For every predicate $\mathcal{F} : \{0,1\}^* \to \{0,1\}$, there exists an AVTC scheme for $\mathcal{L}_{\mathcal{F}}$ as defined in Definition 10.*

To prove the theorem, we first give a high level overview of the AVTC construction. Then, in the following sections, we deep dive into its components and analyze them.

5.2.1. AVTC Protocol.

Our starting point for AVTC is the timed commitment primitive of Boneh and Naor (Section 3.2). We extend its construction to an AVTC by augmenting it with a zero knowledge proof for a predicate $\mathcal{F}$ of the committed value.

A timed commitment $\mathcal{C}$ of a secret $x \in \{0,1\}^l$ is of the form $\mathcal{C} = \langle S, h, g, u \rangle$ where: $h \overset{R}{\leftarrow} \mathbb{Z}_N$, g is deterministically computed [4] from h and $u = 2^{2^k}$ $(mod\ N)$, and the string $S = S_1, ..., S_l$ is defined as:

$$S_i = x_i \oplus LSB\left(g^{2^{2^k - i}}\ (mod\ N)\right)$$

Our protocol involves three proofs:

1) Proving that N is a product of two Blum primes [5]. In Section 5.3.1 it is explained why we enforce a special requirement on the modulus N.
2) Proving that $u = g^{2^{2^k}}$ $(mod\ N)$, which convinces the receiver that it can open the commitment without interaction with the sender within the desired number of computation steps.
3) Proving that $\mathcal{C}$ opens to x s.t $\mathcal{F}(x) = 1$.

In order to prove (1), we employ the scheme of Goldberg et al [25] which yields a non-interactive zero-knowledge proof for a Blum integer. It can be done in a one time setup phase before the AVTC protocol takes place, and the non-interactive proof can be published on a public bulletin board or on the blockchain.

Our protocol for (2) follows the technique of Boneh and Naor as mentioned in Section 3.2 and is found in the appendix in Section 4 along with a complete proof.

In order to prove (3), we prove the following two statements in zero knowledge: [6]

(a) $\forall i \in \{1, ..., l\}$:

$$x_i \oplus S_i = LSB\left(\left(\sqrt[2^{l+1}]{u}\right)^{2^i}\ (mod\ N)\right)$$

(b) $\mathcal{F}(x_1||x_2||...||x_l) = 1$

Our construction for both (a) and (b) is done as a single zero-knowledge proof that is based on the MPC-in-the-head technique of [30].

In the rest of this subsection we give a high level overview of the construction of the zero-knowledge proof for (a) and (b). Then, in Section 5.3 we elaborate all sub-protocols we employ in a top-down manner.

4. We refer the reader to Section 3.2 in the preliminaries, and specifically remind the reader that $g = h^{\left(\Pi_{i=1}^{128} q_i^{\lambda}\right)}$ $(mod\ N)$ where $\{q_i\}_{i=1}^{r}$ are the first r primes and λ is a security parameter.

5. A Blum prime is a prime such that $P \equiv 3\,(mod\ 4)$

6. We denote $||$ as the concatenation operation, and for $X \in \{0,1\}^*$ we denote X_i as the i'th bit of X

Algorithm 1: Validating $(x, w) \in R_{\mathcal{L}}$

```
Input: Witness $w = \sqrt[2^{l+1}]{u}$
 1: $x = \{\}$
    // Initial squaring to ensure $z \in QR_N$
 2: $z = w^2\ (mod\ N)$
    // Concatenate XOR of LSBs
 3: for i=0 to $l - 1$ do
 4:     $x = x || (z_0 \oplus S_i)$
 5:     $z = z^2\ (mod\ N)$
 6: end for
 7: if $\mathcal{F}(x) = 1 \land z = u$ then
 8:     Accept
 9: else
10:     Reject
11: end if
```

Algorithm 1: Lines 3-6 in correspond to (a) which opens the commitment $\mathcal{C} = \langle S, h, g, u \rangle$ where line 7 ensures that successive squaring results in u. As a by-product of opening the commitment we obtain x and in line 7, $\mathcal{F}$ is applied on x to prove (b)

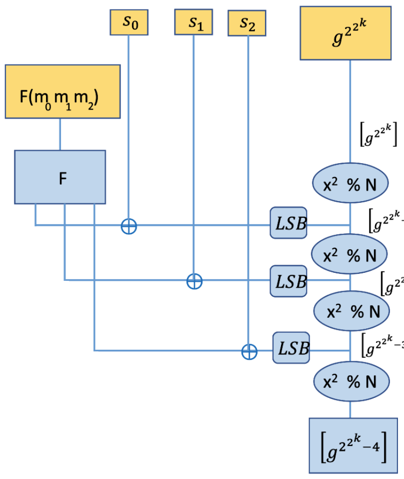


Figure 2: A simple AVTC circuit that verifies an attribute F on a 3-bit sized message m with a commitment string s. Values in blue are secret input, while values in yellow are public input.

5.2.2. Proving $\mathcal{C}$ Opens To x Such That $\mathcal{F}(x) = 1$.

The starting point for our zero knowledge proof that (a), (b) hold is **Algorithm 1** that verifies (a), (b) when x is given as secret input to the prover, and $\mathcal{C}$ is given to both the prover and verifier. Afterwards, we translate **Algorithm 1** to an MPC-friendly algorithm by identifying how each step in **Algorithm 1** can be computed efficiently. Lastly, we build a circuit that will be used to produce a zero-knowledge proof using the "MPC in the head" framework.

5.2.3. Circuit Building Blocks and Configuration.

We construct a layered circuit (seen in **Figure** 2) which is comprised of three main building blocks:

- Squaring modulo N
- LSB extraction
- Field converstion and computation of $\mathcal{F}$

Our circuit executes the open phase of the timed commitment by successively squaring (modulo N) secret shares until a public value u is reached (line 7 in **Algorithm 1**). Each step involves extracting the LSB (while secret-shared) and feeding it into another black-box circuit which computes the desired attribute F (as seen in **Figure 2**). We choose an arithmetic circuit over a prime field p and require that $p > N^2 + n \cdot 2^{\mu}$ for some security parameter μ (i.e $\mu = 100$) and number of parties n. [7] In case $\mathcal{F}$ is most efficiently computed using a boolean circuit, we also perform share conversion from arithmetic shares to boolean shares.

5.2.4. Squaring Modulo N. We devise **Algorithm 2** which verifies that two secret shared values $[a]$ and $[b]$ are congruent modulo N by verifying that the equality $[a] = [b] + [t] \cdot N$ holds, for some $t < N$. Since the equality needs to hold modulo N but the circuit's arithmetic operations are modulo p we require that $p > N^2$.

5.2.5. LSB Extraction. Each squaring and modulo reduction step in the layered circuit is followed by LSB extraction which is then XOR-ed and fed to a component that computes $\mathcal{F}$. Our novel technique for LSB extraction is shown in Section 5.3.5.

5.2.6. Computation of $\mathcal{F}$. Once we have extracted each bit of the committed message x, we forward it into a component which validates that $\mathcal{F}(x) = 1$. However, this component might often be implemented as a circuit in a different field, such as a boolean circuit. To that end, we shall convert our arithmetic shares of bits into boolean shares via **Algorithm 6** which is based on the share conversion technique of [20].

5.2.7. Construction Overview Summary. The AVTC protocol builds on top of Boneh and Naor's timed commitment scheme [15] and its main distinguishing part is a zero-knowledge proof that the commitment is opened to a value that satisfies the predicate. We use the MPC-in-the-head technique of Ishai et al. [30] which we bootstrap with an MPC protocol in the semi-honest setting.

In the next subsection, we will deep dive into each of the components of the aforementioned MPC protocol, discuss our design choices and analyze our scheme.

5.3. AVTC Component Design

In this section we explain in detail our AVTC construction, namely its low level building blocks, and give reasoning behind each design choice. We first describe in our design choice for efficiently opening the timed commitment in the arithemtic circuit while retaining the binding property. Then, we dive into the details of the design of the circuit's subcomponents and analyze them.

5.3.1. Opening a Timed Commitment with a Circuit. We recall the timed commitment (Section 3.2) encoding: $\mathcal{C} = \langle S, h, g, u \rangle$ where $u = g^{2^{2^k}} \pmod N$ for a difficulty parameter k, and $g = h^{\Pi_{i=1}^{r} q_i^{\lambda}}$ where q_i are all primes less than some upper bound r. We wish to prove that the secret x that we commit to, indeed satisfies $\mathcal{F}(x) = 1$. A straightforward way of achieving this, is to execute the *Open* phase of timed commitment in our arithmetic circuit, extract the committed value x, and then compute $\mathcal{F}$ on x. To prevent the prover from equivocating by supplying a different root than of what was used to compute the commitment string $\mathcal{C}$, the timed commitment technique of Boneh and Naor entails the sender sending the receiver $v' = h^{2^{2^k - |x|}}$ and the receiver computing $v = (v')^{\Pi_{i=1}^{r} q_i^{\lambda}}$ which yields $\sqrt[2^{|x|}]{u}$, which is then successively squared to obtain the bits of x.

Proving integrity of the exponentiation of v' by $\Pi_{i=1}^{r} q_i^{\lambda}$ in a cicruit is quite expensive, as the current state of the art [19] utilizes bit decomposition which is computationally expensive. Therefore, we would like to side-step it while retaining binding to the committed value. To that end, we seek out a method that allows the sender of the commitment to convince the receiver that there is a single $2^{|x|}$ root of u without proving integrity of the expensive exponentiation over v' which is done in the *Open* phase in the original paper of Boneh and Naor [15].

We leverage the fact that our modulus N is a product of two Blum primes, [8] and also prove to the verifier using the scheme of Goldberg et al [25] that it is indeed the case. We mention the following theorem which will help us prove that the committer can only prove $\mathcal{F}(x) = 1$ for the same x that the timed commitment committed to.

Theorem 12. *Let N be the product of two Blum integers, and let $a \in QR_N$. Then a has exactly one square root in QR_N.*

Proof. Theorem 9 in [13]. □

Let $w = \sqrt[2^{|x|+1}]{u} \; (mod\ N)$ be a witness used in our zero knowledge proof for which we prove that successively squaring and reduction modulo N starting from w yields u. Indeed, the intermediate value in our zero knowledge proof that is obtained after taking w and squaring it (modulo N) once is $\sqrt[2^{|x|}]{u}$, which is clearly in QR_N, hence (by Theorem 12) there is a single $2^{|x|}$ root of u. As we shall see next, the binding property easily follows from the uniqueness of $\sqrt[2^{|x|}]{u}$:

Corollary 13. *Let $u \in QR_N, w = \sqrt[2^{|x|+1}]{u} \; (mod\ N)$.*

The series $\left\{ w^{2^i} \; (mod\ N) \in QR_N \right\}_{i=1}^{|x|+1}$ is unique.

Proof. By construction, every element in the series is in QR_N, therefore by Theorem 12, for $1 \leq i \leq |x|$ every such element w^{2^i} is a unique square root of another element $w^{2^{i+1}}$ also in QR_N where for $i = |x| + 1$ we get u itself. □

Hence, the witness to the zero knowledge proof is going to be $\sqrt[2^{|x|+1}]{u}$, and the prover will prove that repeated squaring modulo N starting from the witness, reaches u in $|x| + 1$ iterations.

5.3.2. Squaring and Modulo Reduction. Reducing a secret shared number modulo N is an expensive operation because it either requires bit decomposition [36] (decomposing the number to shares of bits), or employing schemes that require random bitwise sharing [35]. Both bit decomposition and bitwise sharing in a field big enough to contain our modulus N means thousands of shares each thousands bits in size. We sidestep this inefficiency by recalling that our circuit only needs to validate the

7. In Section 5.3.5 we show why $p > N^2 + n \cdot 2^{\mu}$ is required

8. A Blum prime is a prime such that $P \equiv 3 \; (mod\ 4)$

Algorithm 2: SquareModNExtractLSB

```
Input: $[x_i]$ (from previous layer) and pre-computed:
    $\mathcal{I}_1 = \left[x_i^2 \ \% \ N\right], \mathcal{I}_2 = \left[\frac{x_i^2 - (x_i^2 \ \% \ N)}{N}\right]$
Output: $\left[x_i^2 \ \% \ N\right]$ (to next layer), $\left[LSB\left(x_i^2 \ \% \ N\right)\right]$ to
    sub-circuit which computes $\mathcal{F}$.
1: $[a] = [x_i] \cdot [x_i]$
2: $[b] = \mathcal{I}_1$    // $\left[x_i^2 \ \% \ N\right]$ pre-computed
3: $[t] = \mathcal{I}_2$    // $\left[\frac{x_i^2 - (x_i^2 \ \% \ N)}{N}\right]$ pre-computed
4: $[y] = [a] - [b] - [t] \cdot N$
5: $y = Reveal([y])$
6: if $y \neq 0$ then
7:     abort
8: end if
9: $z = VerifyLowerThan\left([b], N\right)$
10: $w = VerifyLowerThan\left([a], N^2\right)$
11: $u = LessThanHalfOrder\left([t] \cdot N\right)$
12: if $z \cdot w \cdot u \neq 1$ then
13:     abort
14: end if
15: $[b_0] = ComputeLSB\left([b]\right)$
16: Forward $[b]$ to next level
17: Forward $[b_0]$ towards computation of $\mathcal{F}$
```

computation, and the prover is free to pre-compute all intermediate values of the computation and feed them into the circuit as secret shared inputs.

Specifically, instead of computing $a \ mod \ N$, the prover pre-computes b (denoted as $\mathcal{I}_1$ in **Algorithm 2**) and t (denoted as $\mathcal{I}_2$ in **Algorithm 2**) such that it holds that: $b = a - t \cdot N$, and both b and t are given as input to the parties, which verify that the equality $[a] = [b] + [t] \cdot N$ holds.

At a first glance, validating that $[a] = [b] + [t] \cdot N$ seems trivial. However, the hardness lies in ensuring that a cheating prover doesn't secret share b, t that satisfy the condition but make the computation secretly overflow the field p of the circuit. By initializing the secret shared values b, t with numbers that overflow the finite field p, a cheating prover may manipulate the LSBs extracted from each layer in the circuit and make the verifier think that a committed value has an attribute that it doesn't posses. Therefore, we additionally make **Algorithm 2** ensure that:

1) $b < N$
2) $a < N^2$
3) $t \cdot N < \frac{p}{2}$

Ensuring (1), (2) is done via **Algorithm 3** and ensuring (3) is done via **Algorithm 4**, where both are comparison algorithms and are possible to be computed efficiently thanks to our novel LSB extraction algorithm.

Algorithm 2's correctness immediately follows from the following lemma:

Lemma 14. *For an equality $a = b + t \cdot N$ where $\{a, b, t, N\} \subset \mathbb{Z}_p$ for $p > 4$ and $N^2 < p$, if $b < N$, $a < N^2$ and $[t] \cdot N < \frac{p}{2}$ then there is a single b that satisfies the equality.*

Proof. Assume in contradiction there are b, b', t, t' that satisfy the equality, i.e $a = b + t \cdot N$ and $a = b' + t' \cdot N$. Then it holds that $b + t \cdot N = b' + t' \cdot N$, thus we get: $(\bigstar)$ $b - b' = N \cdot (t' - t)$. Since $N^2 < p$, so $N < \sqrt{p} < \frac{p}{2}$, therefore $b < N < \frac{p}{2}$ and thus $b + t \cdot N < p$. And since $a < N^2$, then $a < p$ as well. Therefore, both sides of the equality do not overflow p and $(\bigstar)$ holds over $\mathbb{N}$, meaning that without loss of generality, $b > b'$ and $t' > t$. It follows that $t' - t \geq 1$, thus $b - b' > N \Rightarrow b > N$ in contradiction to the fact that $b < N$. □

Algorithm 3: VerifyLowerThan

```
Input: $[v], u$
Output: 1 if $v < u$, otherwise aborts
1: $a = \boldsymbol{LessThanHalfOrder}\left([v]\right)$
2: if $a \neq 1$ then
3:     abort
4: end if
5: $[w] = p - [v]$
6: $[x] = [w] + u$
7: $b = \boldsymbol{LessThanHalfOrder}\left([x]\right)$
8: if $b \neq 1$ then
9:     abort
10: end if
11: return 1
```

Algorithm 4: LessThanHalfOrder

```
Input: $[v]$
Output: 1 if $v < \frac{p}{2}$, otherwise aborts
1: $[w] \leftarrow 2 \cdot [v]$
2: $[b] \leftarrow \boldsymbol{ComputeLSB}\left([w]\right)$
3: $b \leftarrow Reveal\left([b]\right)$
4: if $b \neq 0$ then
5:     abort
6: end if
7: return 1
```

5.3.3. Validating Lower-Than.

We give a novel and efficient construction to a secure Multi-Party protocol which validates that a secretly shared value is lower than a public value. Our protocol doesn't doesn't require edaBits like the recent work of [32] and unlike the aforementioned state of the art, has constant rounds. The protocol uses **Algorithm 4** in a black box manner for validating that a secret shared value is less than $\frac{p}{2}$. The protocol and **Algorithm 4** both draw on the ideas from [18] and [37].

The protocol (**Algorithm 3**) receives as input a secret shared value $[v]$ and a public constant $u < \frac{p}{2}$ and returns 1 if and only if $[v] < u$, otherwise aborts.We prove its correctness:

Lemma 15. *For every $u < \frac{p}{2}$, **Algorithm 3** returns 1 if it is input with $[u] < u$, otherwise it aborts.*

Proof. **Algorithm 3** checks if $[v] > \frac{p}{2}$. If so, then $v > u$, therefore it aborts. Otherwise, both v and u are smaller than $\frac{p}{2}$, therefore $(\bigstar)$ $v > u \Leftrightarrow u - v \,(mod \ p) > \frac{p}{2}$. In lines $5 - 6$ we compute $[x] = u - [v] \,(mod \ p)$ and then check if it's less than $\frac{p}{2}$ (line 7). If it is the case, then it follows from $(\bigstar)$ that $v < u$. □

5.3.4. Validating that a value is less than $\frac{p}{2}$.

We show how to validate that $[v] < \frac{p}{2}$. The protocol relies on the fact that if $v < \frac{1}{2}p$ then multiplying it by 2 yields a number that is less than p, and then there is no wrap-around p and the LSB of the result becomes 0 for an odd p. The protocol is shown in **Algorithm 4** and uses the **ComputeLSB** protocol.

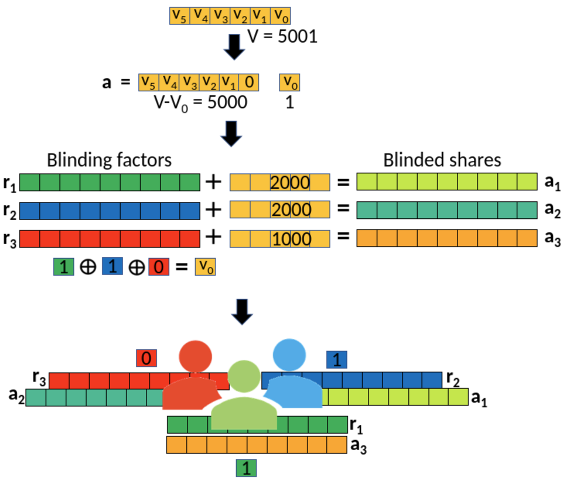


Figure 3: The prover secret-shares v into plaintext shares and then blinds each share with a blinding factor given to one of the remaining two parties (1-secure MPC).

Algorithm 5: ComputeLSB Algorithm

```
Input: $[v]$
Output: $[v_0]$, the LSB of $v$
    Offline phase (performed by the prover):
 1: Set $a = v - v_0$ and $b = v_0$
 2: Set $a = a^{(1)} + a^{(2)} + \ldots + a^{(n)}$ where $\forall i : a^{(i)}$ is even,
    and set $b = b^{(1)} \oplus b^{(2)} \oplus \ldots \oplus b^{(n)}$
 3: $\forall i \in [n]$ :
    $r^{(i)} \overset{R}{\leftarrow} \{2 \cdot x \mid x \in (0, 2^{|N|+\mu-1})\}$; $e^{(i)} \overset{R}{\leftarrow} \{0,1\}$
 4: $\forall i \in [n]$ :
    Give party $i :< a^{(i)} + r^{(i)}; r^{(i+1)}; b^{(i)} \oplus e^{(i)}; e^{(i+1)} >$

    Online phase (performed by the simulated parties):
 5: $\forall i \in [n]$ :  If $\left\{ \left(a^{(i)} + r^{(i)}\right), r^{(i+1)} \right\} \nsubseteq \mathfrak{S}$, abort
 6: $\forall i \in [n]$ :  If $\left\{ \left(b^{(i)} \oplus e^{(i)}\right), e^{(i+1)} \right\} \nsubseteq \{0,1\}$ abort
 7: $\forall i \in [n]$ :  $\left[a^{(i)} + r^{(i)}\right] = SecretShare\left(a^{(i)} + r^{(i)}\right)$
 8: $\forall i \in [n]$ :  $\left[r^{(i+1)}\right] = SecretShare\left(r^{(i+1)}\right)$
 9: $\forall i \in [n]$ :  $\left[b^{(i)} \oplus e^{(i)}\right] = SecretShare\left(b^{(i)} \oplus e^{(i)}\right)$
10: $\forall i \in [n]$ :  $\left[e^{(i)}\right] = SecretShare\left(e^{(i)}\right)$
11: $[v_0] = \bigoplus_{i=1}^{n} \left[b^{(i)} \oplus e^{(i)}\right] \oplus \bigoplus_{i=1}^{n} \left[e^{(i)}\right]$
12: $[z] = [v] - \sum_{i=1}^{n} \left[a^{(i)} + r^{(i)}\right] + \sum_{i=1}^{n} \left[r^{(i)}\right] - [v_0]$
13: $z = Reconstruct\left([z]\right)$
14: if $z \neq 0$ then
15:     abort
16: end if
17: return $[v_0]$
```

5.3.5. Extracting the LSB of a Secret Shared Value.

We describe our novel protocol LSB extraction protocol shown in (**Algorithm 5**). The protocol extracts the LSB of a secret shared value v and is suited to the MPC in the head setting where a prover, knowing v, can act as a dealer and distribute shares in an offline phase before the online phase takes place among the parties. At the end of the online phase, parties obtain a secret sharing of the LSB of v, $[v_0]$.

Our contribution: We construct a novel constant-round protocol for LSB extraction of a secret shared value that is *claimed* (but not *known*, as in [18], [40]) to be smaller than the field size. Our protocol immediately leads to constant round efficient comparison validation protocols in arithmetic circuits and do not require bit decomposition or binary share assisted schemes.

Notations: We denote $[x]$ to be a secret sharing of x among n parties. When referring to an operation done by every party, we shall denote $\forall i \in [n]$. Let $\mathfrak{S} = \left\{2 \cdot \alpha \mid \alpha \in \left(0, 2^{|N|+\mu-1}\right)\right\}$: be the set of legal blinded plaintext shares in our algorithm.

The technique: For simplicity, we first describe the technique for a semi-honest *1-secure* setting (which assumes parties do not collude), as shown in **Figure 3**. A modification to a semi-honest k-secure case, where parties may collude appears later on. In the *offline phase*, the prover first represents v as a sum of an even number a and the LSB of v denoted by b: $v = a + b$ (line 1). Then, it splits a into n shares $a^{(1)} + a^{(2)} + \ldots + a^{(n)} < p$ and it splits b into bits such that: $b^{(1)} \oplus b^{(2)} \oplus \ldots \oplus b^{(n)}$ (line 2). Finally, it statistically hides each share $a^{(i)}$ with an *even* uniformly random variable $r^{(i)} \overset{R}{\leftarrow} \{2 \cdot x \mid x \in (0,\ 2^{|N|+\mu-1})\}$ (line 3) where $|N|$ is the number of bits in N, n is the number of parties and μ is a security parameter such that $p > N^2 + n \cdot 2^{\mu}$. Similarly, it blinds each share $b^{(i)}$ with a random bit $e^{(i)}$. The prover sets each party i's input (line 4) to be the following plaintext shares: $a^{(i)} + r^{(i)}$; $r^{(i+1)}$; $b^{(i)} \oplus e^{(i)}$; $e^{(i+1)}$.

For a k-secure MPC protocol, each share $a^{(i)}$ is blinded with k blinding factors, e.g in a 2-secure MPC protocol with 3 parties, the share $a^{(1)}$ is blinded with $r^{(1,2)}$ which is given to party 2 and with $r^{(1,3)}$ which is given to party 3.

In the *online phase*, each party verifies that its plaintext share is in the appropriate range (line 5) and the blinding factor for $a^{(i)}$ is even, and secret shares the blinded share and the blinding factor among all parties (lines 7-10), after which the parties compute $\left[a^{(i)} + r^{(i)}\right], \left[r^{i+1}\right], \left[b^{i} \oplus e^{(i)}\right], \left[e^{i+1}\right]$ and finally extract the sharings of the LSB (line 11) after which they jointly validate the plaintext shares given to them in the offline phase (lines 12-15).

5.3.6. LSB extraction security proof.

We formally prove the correctness and privacy of the LSB extraction protocol. We give three theorems and prove them in Section D. For simplicity of notation, we set the number of parties to be three.

Definition 16 (ComputeLSB protocol)**.** Let $\overrightarrow{x} = (x_0, x_1, x_2)$ be shares distributed by a dealer and define the LSB extraction functionality $\mathfrak{f} : \left(\mathbb{Z}_p\right)^3 \rightarrow \left(\mathbb{Z}_p\right)^3$ where $\mathfrak{f}_i\left(\overrightarrow{x}\right)$ is the output of party i, and let π to be the LSB extracton protocol (**Algorithm 5**) and $VIEW_i^{\pi}\left(\overrightarrow{x}\right)$ the view of party i during the execution of π.

Theorem 17 (LSB extraction completeness)**.** *On input $[v]$ with an honest dealer, ComputeLSB outputs $[v_0]$ (secret shares of the LSB of v) and never aborts.*

Theorem 18 (LSB extraction privacy)**.** *There exists a PPT $\mathcal{S}$ such that for a single semi-honest party $j \in \{0,1,2\}$: $\left\{\mathcal{S}\left(x_j, \mathfrak{f}_j\left(\overrightarrow{x}\right)\right)\right\} \equiv_s \left\{VIEW_j^{\pi}\left(\overrightarrow{x}\right)\right\}$*

Theorem 19 (LSB extraction soundness)**.** *ComputeLSB always either aborts or correctly extracts $[v_0]$.*

Algorithm 6: BooleanizeLSB

```
Input: $[v_0]$
Output: $[v_0]_B$
1: $\forall i \in [n] : [r_i] = SecretShare\left(Rand\left(\frac{p}{n+1}\right)\right)$
2: $\forall i \in [n] : [b_i]_B = SecretShare\left([r_i \ \% \ 2]\right)$
3: $[b]_B = \bigoplus_{i\in[n]} [b_i]_B$
4: $[r] = \sum_{i\in[n]} [r_i]$
5: $[x] = [r] + [v_0]$
6: $x = Reconstruct\left([x]\right)$
7: $[v_0]_B = LSB(x) \oplus [b]_B$
```

Combining the three theorems above yields that an honest prover distributes plaintext shares in a fashion that ensures that the verifier always accepts, and the output to the next gate in the circuit is indeed the LSB of the secret shared input value.

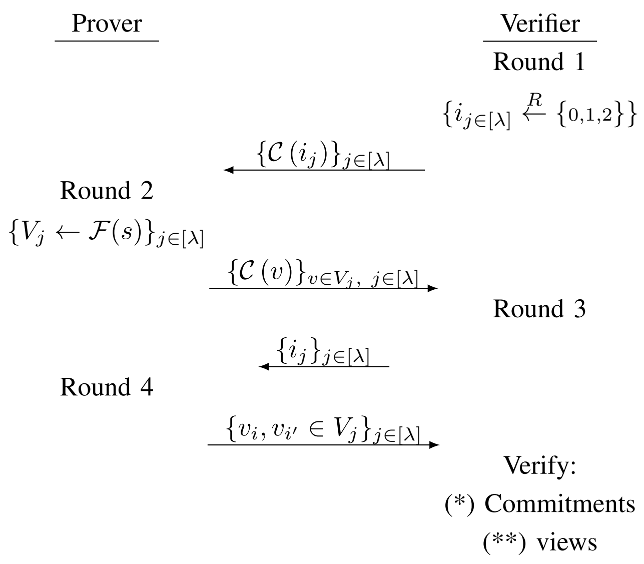
Additionally, the protocol achieves statistical zero knowledge, since the view of every single semi-honest party participating in the protocol is statistically indistinguishable from the simulated view.

Lastly, if the dealer is dishonest, the MPC protocol of the LSB extraction always aborts. It immediately follows that the LSB extraction protocol does not degrade the soundness of the zero knowledge proof, as the probability of a cheating prover to commit to an incorrect protocol execution does not depent on whether it entails an LSB extraction or not.

5.3.7. Conversions between different fields. After extracting the LSB of each layer in our circuit, we show in **Algorithm 6** how to convert it from a sharing modulo p to the domain of the predicate $\mathcal{F}$. Since our predicate of choice in our cross chain swap protocol (Section 4) is SHA256 comparison, we explain how we convert the arithmetic sharing to a boolean sharing, and denote the shares in the target domain as $[\cdot]_B$. Our technique follows [20], and its main idea is to produce masking factors in both the arithmetic domain and the boolean domain such that the LSB of both masking factors is identical (lines 1-2). Then, the masking factor in the arithmetic domain, masking the secret shared LSB (line 5), is revealed (line 6) and the LSB is used to cancel out the mask in the arithmetic domain (line 7).

5.3.8. Instantiating the MPC Protocol To Be Used In The "MPC In The Head" Framework. We instantiate an MPC protocol using a semi-honest 2-secure[9] GMW [26] where the arithmetic part of our circuit uses an arithmetic GMW, and the boolean part of our circuit for computing $\mathcal{F}$ uses a boolean GMW. Our conversion from an MPC protocol to a zero-knowledge proof using the "MPC in the head" framework follows [30] and it is outlined in Section 5.3.9. We elaborate on our salient difference from the state of the art, the secure multiplication, as secret sharing and addition are done in the standard manner. To multiply $[z] = [x] \cdot [y]$, we use an Oblivious Linear Evaluation (OLE) oracle $\mathfrak{O}$: Every pair of parties i, j use

9. A 2-secure protocol can withstand collusion of two parties out of the remaining three



Prover
Verifier
Round 1
$\{i_{j\in[\lambda]} \xleftarrow{R} \{0,1,2\}\}$
$\{\mathcal{C}(i_j)\}_{j\in[\lambda]}$
Round 2
$\{V_j \leftarrow \mathcal{F}(s)\}_{j\in[\lambda]}$
$\{\mathcal{C}(v)\}_{v\in V_j,\ j\in[\lambda]}$
Round 3
$\{i_j\}_{j\in[\lambda]}$
Round 4
$\{v_i, v_{i'} \in V_j\}_{j\in[\lambda]}$
Verify:
(*) Commitments
(**) views

Figure 4: The MPC in the head interactive protocol

the OLE oracle $\mathfrak{O}$ to compute blinded products of each party's inputs:
1) Every party i samples a random value $r_{i,j} \in \mathbb{Z}_p$ for each party $j \neq i$ and sends the share x_i and $r_{i,j}$ to $\mathfrak{O}$.
2) party j sends to $\mathfrak{O}$ the share y_j
3) The OLE oracle $\mathfrak{O}$ sends party j $s_{i,j} = x_i \cdot y_j + r_{i,j}$.

Afterwards, $[z]$ is defined by having the blinding factors sent by party i, subtracted: $z_i = x_i + y_i + \sum_{j=1, j\neq i}^{n} (s_{j,i} - r_{i,j})$. When simulating $\mathfrak{O}$, the prover commits to the inputs and outputs of $\mathfrak{O}$. When verifying the multiplication gate, the prover reveals two views to the verifier, and the latter verifies they are consistent with the commitments. Note, that since we employ a 2-secure GMW, the prover reveals to the verifier the views of two out of the three parties. Thus, we only need to reveal the de-commitment of the party which its view is not revealed, as the verifier can compute the expected de-commitments of the rest of the parties. This saves us significant bandwidth, as our de-commitments are elements in $\mathbb{Z}_p$ and have thousands of bits each. In each iteration, the verifier opens two out of the three views committed by the prover, therefore the probability of the verifier accepting an incorrect proof with λ rounds is $\left(\frac{1}{3}\right)^{\lambda}$. Since our zero-knowledge protocol is interactive, k doesn't need to be large, as in the honest-verifier non-interactive zero-knowledge, because the prover cannot use computation to predict the verifier's challenge.

5.3.9. Obtaining Zero-Knowledge From MPC. We describe an interactive protocol (**Figure 4**) between the sender of the commitment, playing as the prover, and the receiver of the commitment, playing as the verifier. Denote v_i as the view of party $i \in \{0, 1, 2\}$ in an MPC protocol executed on secret input s.

The view includes private input, random tape, and incoming messages received during the protocol from foreign parties. Denote $\mathcal{C}$ a commitment operation and define $V_j = \bigcup_{i\in\{0,1,2\}}\{v_i\}$ as the union of all views in the j'th simulation of MPC protocol and let λ be a security parameter. The zero knowledge proof is a four round protocol:

Round 1: The verifier samples λ indices uniformly at

random and commits to them using a perfectly hiding commitment, and sends the commitment to the prover.
Round 2: The prover simulates the MPC protocol for $\mathcal{F}$ on secret input s, λ times and commits (using a perfectly binding commitment) to each view (of each party) separately, and sends the λ triplets of commitments to the verifier.
Round 3: The verifier decommits the vector of λ indices $\{i_j\}_{j\in[\lambda]}$ sent in the first round.
Round 4: The prover checks the decommitments of the verifier, and then replies with the views that correspond to other two remaining requested indices $\{\{0,1,2\} \setminus \{i_j\}\}_{j\in[\lambda]}$. The verifier then: (*) verifies that the commitments match the received views, (**) simulates the execution of each party that its view was sent by the verifier. Specifically, the verifier ensures that the messages that the party should send, match the commitments on the received messages of the other parties.

5.3.10. Zero knowledge Proof For Membership In $L_{\mathcal{F}}$. After proving the correctness and privacy of our LSB extraction protocol (Sections 5.3.5, 5.3.6), as well the correctness and privacy of all other algorithms instantiated with the GMW [26] scheme, we can now prove our main result for this section, Theorem 11. The proofs of completeness, soundness and zero-knowledge are found in Section E.

5.4. Evaluation

We implement the AVTC zero-knowledge proof used for our cross chain atomic swap protocol from Section 4, which requires proving in zero knowledge that an AVTC commitment string, commits to a SHA256 pre-image. We re-use code from ZKBoo [24] for the SHA256 circuit, and implement the AVTC arithmetic circuit and share conversion in 3,500 lines of C code available online [1].

We use the GNU MP library [5] for our arithmetic share addition and multiplication, and OpenMP [7] for parallel processing of soundness amplification. We experiment on an Intel i7 2.6Ghz with 12 cores and 32GB RAM with different numbers of rounds for soundness amplification, and in each configuration we compute the network bandwidth sent from the prover to the verifier (see **Table 1** and Section G) and estimate the latency for a 300 Mbps network.

5.5. Comparison to Other Schemes

In this section we discuss alternative options for our AVTC implementation. We consider other interactive Zero-Knowledge proofs and also non-interactive ones such as ZK-SNARKs and discuss trade-offs.

5.5.1. Lowering the network bandwidth. A recent promising zero-knowledge proof scheme that has both fast proving time and low bandwidth is [45]. It is a flavor of ”MPC in the head” where the prover and verifier run a two-party MPC protocol (with malicious security) and unlike the original ”MPC in the head”, they only do a single pass over the circuit. However, the protocol assumes a designated verifier because a preliminary setup phase must take place between the prover and verifier prior to the protocol. In contrast, the standard ”MPC in the head” approach enables the prover to pre-compute the views of the parties and their commitments long before it comes into interaction with the verifier, and then when the verifier asks for opening of the commitments just send the pre-computed de-commitments.

#Rounds	Security	Prover time	Verifier time	Total bandwidth	Network Latency
26	2^{40}	7.8s	4.9s	577 MB	15s
19	2^{30}	5.5s	3.2s	421 MB	11s
13	2^{20}	3.3s	2s	288 MB	8s

TABLE 1: Performance evaluation of AVTC where the attribute $\mathcal{F}$ is the SHA2_256 function

5.5.2. Quantitative comparison to ZK-SNARKs. General purpose zero-knowledge proof systems vary from simulations of MPC protocols (fast proving time, but high bandwidth) to non-interactive arguments of knowledge (slow proving time, low bandwidth). The analysis of [3] compared different systems and showed that the Groth16 [27] scheme is the fastest in both prover time and verfier time, having an order of magnitude difference from the slowest scheme (Bulletproofs [17]) in their analysis.

To that end, we implement [2] AVTC with the Groth16 ZK-SNARK library of Gnark [16] instantiated with the BN-254 curve. In particular, we implement gadgets for 4096 bit number addition and 4096 bit number multiplication and then use them in the successive modular squaring using pre-computed hints similar to **Algorithm 2** in Section 5.3.2. Finally, the LSBs of each layer are fed into a SHA256 gadget we implemented. We benchmark the Groth16 implementation on an AWS c5a.8xlarge machine with a 64GB RAM and 32 CPUs. The time it takes to run the trusted setup of the 14 million constraints circuit is 3 minutes and 38 seconds. The prover runs in 54 seconds and verifier in 1.4 milliseconds. The large number of constraints is due to the need to represent 4096 bit numbers as arrays of smaller numbers (we pick 120 bit words), since the order of the BN-254 curve is only 254 bits long. However, we use hash pre-images of only 96 bits in the Groth16 in contrast to 128 bits in the ”MPC-in-the-head” AVTC. The reason is that the more bits in the hash pre-image, the more layers of successive modular squaring we have. Unlike in our ”MPC-in-the-head” construction where the number of iterations is the dominating factor, in the Groth16 ZK-SNARK it is the depth of the circuit.

6. Conclusions

In this work we explored whether cross chain atomic swaps based on Hash Timed Lock Contracts can be constructed without reliance on a time source, and answered the question in the affirmative by providing a protocol which replaces the well known blockchain based timeouts used for safety with timed commitments. We enforced honest party behavior for our protocol via our novel AVTC primitive, which allows to prove arbitrary attributes for Boneh and Naor’s timed commitment.

While there is plenty of room for efficiency improvements in our construction, as it requires substantial network bandwidth, we make up for it in minimal prover time, and in not requiring a trusted setup.

References

[1] AVTC MPC in the head implementation. https://github.com/AVTC-paper/AVTC-MPC.

[2] AVTC ZK-SNARK implementation. https://github.com/AVTC-paper/AVTC-SNARK.

[3] Benchmarking zero-knowledge proofs with isekai: https://sikoba.com/docs/skor_isekai_benchmarking_201912.pdf.

[4] Bitcoin time locks: https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki.

[5] GMP library. https://gmplib.org/.

[6] Interledger https://interledger.org/interledger.pdf.

[7] OpenMP library. https://www.openmp.org/.

[8] sawtooth: https://sawtooth.hyperledger.org/.

[9] sawtooth: https://sawtooth.hyperledger.org/faq/transaction-processing/#why-is-there-no-timestamp-in-a-transaction-header-or-block.

[10] Tendermint htlcs: https://github.com/nomic-io/htlc.

[11] G. Almashaqbeh, F. Benhamouda, S. Han, D. Jaroslawicz, T. Malkin, A. Nicita, T. Rabin, A. Shah, and E. Tromer. Gage mpc: Bypassing residual function leakage for non-interactive mpc. *Proceedings on Privacy Enhancing Technologies*, 2021:528 – 548, 2021.

[12] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. *CoRR*, abs/1801.10228, 2018.

[13] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo random number generator. *SIAM J. Comput.*, 15(2):364–383, May 1986.

[14] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. *IACR Cryptol. ePrint Arch.*, 2018:712, 2018.

[15] D. Boneh and M. Naor. Timed commitments. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 236–254, 2000.

[16] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie. Consensys/gnark: v0.6.4, Feb. 2022.

[17] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[18] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In J. A. Garay and R. De Prisco, editors, *Security and Cryptography for Networks*, pages 182–199, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[19] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.

[20] I. Damgård and R. Thorbek. Efficient conversion of secret-shared values between different fields. *IACR Cryptol. ePrint Arch.*, 2008:221, 2008.

[21] Y. Doweck and I. Eyal. Multi-party timed commitments. *ArXiv*, abs/2005.04883, 2020.

[22] J. A. Garay and C. Pomerance. Timed fair exchange of standard signatures. In R. N. Wright, editor, *Financial Cryptography*, pages 190–207, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[23] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 3–16, New York, NY, USA, 2016. Association for Computing Machinery.

[24] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1069–1083, Austin, TX, 2016. USENIX Association.

[25] S. Goldberg, L. Reyzin, O. Sagga, and F. Baldimtsi. *Efficient Noninteractive Certification of RSA Moduli and Beyond*, pages 700–727. 11 2019.

[26] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[27] J. Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Report 2016/260, 2016. https://ia.cr/2016/260.

[28] M. Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 245–254, New York, NY, USA, 2018. ACM.

[29] M. Herlihy, B. Liskov, and L. Shrira. Cross-chain deals and adversarial commerce. *Proc. VLDB Endow.*, 13(2):100–113, Oct. 2019.

[30] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 21–30, New York, NY, USA, 2007. ACM.

[31] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *Int. J. Inf. Secur.*, 1(1):36–63, aug 2001.

[32] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh. Rabbit: Efficient comparison for secure multi-party computation. *IACR Cryptol. ePrint Arch.*, 2021:119, 2021.

[33] G. Malavolta and S. Thyagarajan. *Homomorphic Time-Lock Puzzles and Applications*, pages 620–649. 08 2019.

[34] A. Mitra, C. Gorenflo, L. Golab, and S. Keshav. Timefabric: Trusted time for permissioned blockchains. 2021.

[35] C. Ning and Q. Xu. Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 483–500, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[36] C. Ning and Q. Xu. Constant-rounds, linear multi-party computation for exponentiation and modulo reduction with perfect security. In D. H. Lee and X. Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 572–589, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[37] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *International Workshop on Public Key Cryptography*, pages 343–360. Springer, 2007.

[38] K. Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[39] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[40] T. I. Reistad. Multiparty comparison - an improved multiparty protocol for comparison of secret-shared values. In *SECRYPT*, 2009.

[41] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. Roy and W. Meier, editors, *Fast Software Encryption*, pages 371–388, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[42] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[43] S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sánchez. Universal atomic swaps: Secure exchange of coins across all blockchains. Cryptology ePrint Archive, Report 2021/1612, 2021. https://ia.cr/2021/1612.

[44] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder. Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1733–1750, New York, NY, USA, 2020. Association for Computing Machinery.

[45] C. Weng, K. Yang, J. Katz, and X. Wang. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091. IEEE, 2021.

[46] B. Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.

[47] J.-Y. Zie, J.-C. Deneuville, J. Briffaut, and B. Nguyen. Extending atomic cross-chain swaps. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 219–229. Springer, 2019.

Appendix A. LHTLP construction

We elaborate on our observation from Section 2.2.5 on the trusted setup requirement of the Homomorphic Time-Lock Puzzles (HTLP) by going into details on the Linear Homomorphic Time-Lock Puzzle (LHTLP) construction and showing that a setup phase that is executed dishonestly leads to incorrect recovery of the puzzle. The construction of the LHTLP is as follows:

- $Setup\left(1^{\lambda}, T\right)$: Sample two safe primes p, q and set $N = p \cdot q$, and sample $\bar{g} \overset{R}{\leftarrow} \mathbb{Z}_N^*$ and set $g = -\bar{g}^2\,(mod\ N)$. Compute $h = g^{2^T}$ and output $pp = (T, N, g, h)$.
- $Gen\,(pp, s)$: Let $r \overset{R}{\leftarrow} \{1, ..., N^2\}$ and compute $u = g^r\,(mod\ N), v = h^{r \cdot N} \cdot (1+N)^s \left(mod\ N^2\right)$ and output $Z = (u, v)$.
- $Solve(pp, Z = (u, v))$: Let $w = u^{2^T}\ (mod\ N)$. Output $s = \left(\left(\left(v \cdot w^{-N}\right) mod\ N^2\right) - 1\right) \cdot N^{-1}$
- $Eval\left(\oplus, pp, \{Z_i = (u_i, v_i)\}_{i=1}^n\right)$: Let $\bar{u} = \prod_{i=1}^n u_i\,(mod\ N)$, $\bar{v} = \prod_{i=1}^n v_i\left(mod\ N^2\right)$ and output $Z = (\bar{u}, \bar{v})$.

At a first glance, the Homomorphic Time Lock Puzzle (HTLP) construction seems similar to the Timed Commitment primitive devised by Boneh and Naor (see **Definition 4**). We note however, an important difference: In the timed commitment, the sender proves to the receiver that the forced open (called $Solve$ in HTLP) indeed yields the committed value within the claimed number of computation steps. It is not the case in LHTLP (or any of the other variations of HTLP). Moreover, even if Gen was computed honestly by the sender, or that the receiver verifies the correct computation of Gen via an interactive proof, there is no guarantee that the $Setup$ phase was constructed correctly. Indeed, in the LHTLP construction, the $Solve$ operation computes $w = u^{2^T} = g^{r \cdot 2^T}$ via successive modular squaring and then if h was constructed honestly in the $Setup$ phase, then $h = g^{2^T}$ and then $v = g^{r \cdot 2^T \cdot N} \cdot (1+N)^s$ which fits the aforementioned w. However, if h was not honestly constructed in $Setup$ then $Solve$ will not output the s that was used as input to Gen. The authors of [33] do mention that the $Setup$ should be executed by a trusted party. We note that unlike the HTLP where the $Setup$ phase outputs T, N, g, h, the $Setup$ phase of the the timed commitment outputs only the modulus N.

Appendix B. Proving that $u = g^{2^{2^k}} \quad (mod\ N)$

We describe the protocol executed as part of the $Commit$ phase of AVTC which convinces the receiver that the commitment can be forcefully opened in 2^k sequential steps. Recall, the prover can compute $u = g^{2^k}$ by computing the exponent $e = 2^{2^k}\ (mod\ \phi\,(N))$ and then $u = g^e\,(mod\ N)$. In contrast, the receiver, not knowing the factorization of N, has to resolve to 2^k successive squaring and modulo reductions.

The prover is going to send the verifier a vector W of the form

$$W = \left\langle g^2, g^4, g^{16}, g^{256} ..., g^{2^{2^i}}, ..., g^{2^{2^k}} \right\rangle (mod\ N) = \left\langle g^{2^{2^i}}\ (mod\ N) \right\rangle_{i=1}^{k}$$

along with a zero knowledge proof that for every two consecutive elements a, b in W it holds that: $\exists x \middle| a = g^x\,(mod\ N) \wedge b = a^x\,(mod\ N)$, or alternatively: $\exists x \middle| a = g^x\,(mod\ N) \wedge b = g^{x^2}\,(mod\ N)$.

The proof convinces the verifier that the last element in W is indeed the desired $u = g^{2^{2^k}}\ mod\ N$, and is computed from g by repeated squaring followed by modulo reductions of N. Note that the receiver learns $g^{2^{2^{k-1}}}$, however $\left(g^{2^{2^{k-1}}}\right)^{2^{2^{k-1}}} = g^{2^{2^{k-1}} \cdot 2^{2^{k-1}}} = g^{2^{2^k}} \equiv_{(mod\ N)} u$. Therefore, given the vector W, the receiver still needs to perform an exponential (2^{k-1}) number of squarings followed by modulo reduction by N.

Let λ be a security parameter and denote $[\lambda] = \{0, 1, .., \lambda\}$ and q the order of $g \in Z_N^*$. Observe the interactive zero knowledge protocol of Boneh and Naor for inclusion in $L = \{(a_i, b_i) \middle| \exists x\ s.t\ a_i = g^x\,(mod\ N) \wedge b_i = g^{x^2}\,(mod\ N)\}$:

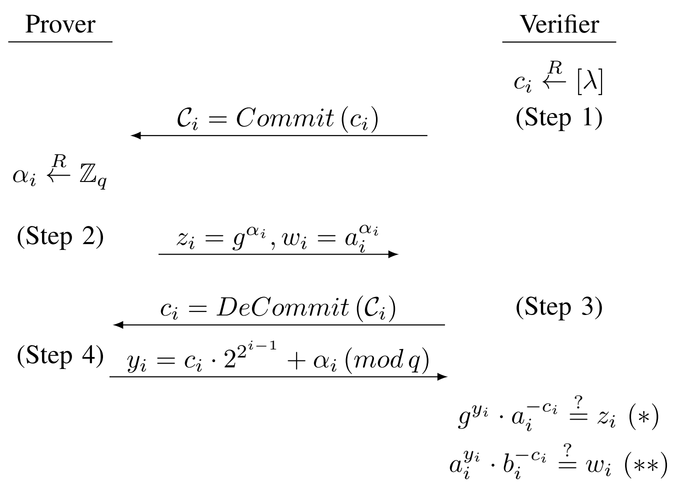

Prover
Verifier
$c_i \overset{R}{\leftarrow} [\lambda]$
(Step 1)
$\mathcal{C}_i = Commit\,(c_i)$
$\alpha_i \overset{R}{\leftarrow} \mathbb{Z}_q$
(Step 2)
$z_i = g^{\alpha_i}, w_i = a_i^{\alpha_i}$
$c_i = DeCommit\,(\mathcal{C}_i)$
(Step 3)
(Step 4)
$y_i = c_i \cdot 2^{2^{i-1}} + \alpha_i\,(mod\,q)$
$g^{y_i} \cdot a_i^{-c_i} \overset{?}{=} z_i\ (*)$
$a_i^{y_i} \cdot b_i^{-c_i} \overset{?}{=} w_i\ (**)$

Figure 5: Proof that $\exists x \middle| a_i = g^x\,(mod\ N) \wedge b_i = g^{x^2}\,(mod\ N)$

Each step of the protocol depicted in **Figure 5** is executed in parallel for all $i \in [k]$:

Step 1:The verifier samples $\{c_i\}_{i \in [k]} \in [\lambda]$ uniformly at random, and sends a commitment to each c_i using an perfectly hiding commitment scheme to the prover.

Step 2:The prover, in turn, samples $\{\alpha_i\}_{i \in [k]}$ and sends $z_i,\ w_i$ for all $i \in [k]$.

Step 3:The verifier decommits to all $\{c_i\}_{i\in[k]}$ and the prover aborts if one of the decommitments turn out to be false.
Step 4:Finally, the prover sends y_i, and the verifier checks that conditions (*) and (**) hold. A formal proof of the protocol, which wasn't included in the original paper of Boneh and Naor, can be found in the next subsection.

Appendix C. Proof for the Boneh-Naor Zero Knowledge Protocol

C.1. Completeness

Observe the interaction between an honest prover and a verifier for an arbitrary $i \in [1,k]$: First the prover sends $z_i = g^{\alpha_i}$ and $w_i = a_i^{\alpha_i}$ and finally sends $y_i = c_i \cdot 2^{2^{i-1}} + \alpha_i (mod\ q)$. The verifier computes: $g^{y_i} \cdot a_i^{-c_i} = g^{c_i \cdot 2^{2^{i-1}} + \alpha_i} \cdot a_i^{-c_i} = \left(g^{2^{2^{i-1}}}\right)^{c_i} \cdot g^{\alpha_i} \cdot a_i^{-c_i} = (a_i)^{c_i} \cdot g^{\alpha_i} \cdot a_i^{-c_i} = z_i$, and $a_i^{y_i} \cdot b_i^{-c_i} = a_i^{c_i \cdot 2^{2^{i-1}} + \alpha_i} \cdot b_i^{-c_i} = \left(g^{2^{2^{i-1}}}\right)^{2^{2^{i-1}} \cdot c_i + \alpha_i} \cdot b_i^{-c_i} = \left(g^{2^{2^i}}\right)^{c_i} \cdot \left(g^{2^{2^{i-1}}}\right)^{\alpha_i} \cdot b_i^{-c_i} = b_i^{c_i} \cdot a_i^{\alpha_i} \cdot b_i^{-c_i} = w_i$ as required, therefore it always accepts the proof.

C.2. Soundness

Let (a_i, b_i) for some $i \in [1,k]$ such that $\nexists x \middle| a_i = g^x\ (mod\ N) \wedge b_i = g^{x^2}\ (mod\ N)$ and assume in contradiction that $\forall i \in [1,k]$ a cheating prover P^* can send (z_i, w_i), get a challenge c_i and then send y_i and convince the verifier that $(a_i, b_i) \in L$ with non negligible probability. Define the following set C:

$$C = \left\{ c_i \middle| \exists y_i\ s.t\ g^{y_i} \cdot a_i^{-c_i} = z_i \wedge a_i^{y_i} \cdot b_i^{-c_i} = w_i \wedge (a_i, b_i) \notin L \right\}$$

The set C, informally, is the set of all challenges from an honest verifier c_i such that there exists a response from the prover y_i such that the verifier accepts a tuple (a, b) not in L.

Let's split into cases in accordance to C's cardinality:

- $|C| = 0$: By definition, we get complete soundness.
- $|C| = 1$: To cheat, the prover will guess c_i, pick y_i and send $z_i = g^{y_i} \cdot a_i^{-c_i}$ and $w_i = a_i^{y_i} \cdot b_i^{-c_i}$ and the verifier would accept. In that case, the probability of the prover to convince the verifier of $(a_i, b_i) \notin L$ is $\frac{1}{q}$, because it needs to guess the challenge c_i which has q (the order of $g \in Z_N^*$) different possibilities despite the verifier picking $c_i \in [\lambda]$, and every challenge is selected uniformly at random.
- $|C| > 1$: There are at least 2 such challenges: $c_i \neq c_i'$. We'll see that if that is the case, then $\forall i \in [1,k]$: $(a_i, b_i) \in L$ in contradiction to the assumption.
  The following equations hold:

$$(1)\ \ g^{y_i} \cdot a_i^{-c_i} = z_i$$

$$(2)\ \ a_i^{y_i} \cdot b_i^{-c_i} = w_i$$

$$(3)\ \ g^{y_i'} \cdot a_i^{-c_i'} = z_i$$

$$(4)\ \ a_i^{y_i'} \cdot b_i^{-c_i'} = w_i$$

From (1) and (3) we get

$$g^{y_i} \cdot a_i^{-c_i} = g^{y_i'} \cdot a_i^{-c_i'} \Rightarrow g^{y_i - y_i'} = a_i^{c_i - c_i'}.\ (5)$$

and from (2) and (4) we get:

$$a_i^{y_i} \cdot b_i^{-c_i} = a_i^{y_i'} \cdot b_i^{-c_i'} \Rightarrow a_i^{y_i - y_i'} = b_i^{c_i - c_i'}\ (6)$$

Denote $\gamma = y_i - y'$ and $\delta = c_i - c_i'$, thus from (5) we get $g^\gamma = a_i^\delta$ and from (6) we get $a_i^\gamma = b_i^\delta$. Thus, $g^\gamma = a_i^\delta \Rightarrow g^{\gamma \cdot \delta^{-1}} = a_i$ and $a_i^\gamma = b_i^\delta \Rightarrow a_i^{\gamma \cdot \delta^{-1}} = b_i$. Denote $\beta = \gamma \cdot \delta^{-1}\ (mod\, q)$. Hence $g^\beta = a_i$ and $a_i^\beta = b_i$, so $(g, a_i, b_i) = \left(g, g^\beta, g^{\beta^2}\right)$, thus $(a_i, b_i) \in L$ in contradiction to the assumption.

C.3. Zero Knowledge

We show that the protocol in **Figure** 5 satisfies the zero-knowledge property. For this purpose we construct a simulator S that upon receiving input $(a_i, b_i) \in L$ produces a view that is indistinguishable from the verifier's view in a run with an honest prover, or more formally: $\left\{View_{V^*}^{P}(a_i, b_i)\right\}_{x \in L} \equiv_s \{S(a_i, b_i)\}$.

1) The simulator sets the random tape $\hat{\mathfrak{T}}$ of V^* by randomly sampling from $\{0,1\}^\tau$ where τ denotes an upper bound on the running time of V^*
2) The simulator waits for the commitments given in step 1 by the verifier
3) The simulator samples $\alpha_i \xleftarrow{R} \mathbb{Z}_N$ and sends $\hat{z}_i = g^{\alpha_i}, \hat{w}_i = a_i^{\alpha_i}$.
4) The simulator waits for the decommitments from the verifier
5) The simulator rewinds the verifier to right after its first step
6) The simulator samples $\hat{y}_i \in \mathbb{Z}_N$ uniformly at random and computes
$$\hat{z}_i = g^{\hat{y}_i} \cdot a_i^{-c_i}, \hat{w}_i = a_i^{\hat{y}_i} \cdot b_i^{-c_i}$$
7) The simulator sends $\hat{z}_i, \hat{w}_i$, waits for the decommitment to c_i from the verifier
8) The simulator sends $\hat{y}_i$

We next show that $\left\{View_{V^*}^{P}(a_i, b_i)\right\}_{x \in L} \equiv_s \{S(a_i, b_i)\}$. The view of the verifier V^* consists of the public input $(a_, b_i)$, its random tape $\mathfrak{T}$, and the messages received $\mathfrak{M}$, or more formally: $View_{V^*}^{P} = \langle (a_i, b_i), \mathfrak{T}, \mathfrak{M} \rangle$.
The messages $\mathfrak{M}$ sent by the real prover in **Figure** 5 are z_i, w_i in step (2) and y_i in step (4), while $\hat{\mathfrak{M}}$ consists of $\hat{z}_i, \hat{w}_i$ sent in steps (3),(7) and of $\hat{y}_i$ sent in step (8).

First, observe that the only message sent in the protocol that is related to the public input (a_i, b_i) is the first message containing $w_i = {a_i}^{\alpha_i}$. The simulator sends this message twice (before and after rewinding) in steps (3) and (6). We will show that the first time the simulator sends the message $(\hat{z}_i, \hat{w}_i)$ in step (3), its joint distribution with the public input distributes statistically close to the joint distribution of the corresponding first message and public input in the real protocol. The last message sent in the protocol (both by the real prover and the simulator) is independent of the public input. We will show that the last message (step 8) the simulator sends distributes statistically close to the corresponding message sent by the real prover. For the second message sent by the simulator, and corresponds to the first message sent in the real protocol, we will show that the exponents are statistically close, and conclude that the messages are also statistically close.

Statistically close messages: We show that the messages sent by the simulator in steps (3) and (8) are statistically close to their corresponding messages in the real protocol:

- **Step (3)** $(\hat{z}_i, \hat{w}_i) \in \mathbb{Z}_N \times \mathbb{Z}_N$ since $\alpha_i \xleftarrow{R} \mathbb{Z}_N$ and $\hat{z}_i = g^{\alpha_i}, \hat{w}_i = a_i^{\alpha_i}$, while the real prover sends $(z_i, w_i) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$. The messages are therefore picked in the same manner in both the simulated protocol and the real protocol, except from the distributions the exponents are sampled from. Therefore we will show that: $\mathbb{Z}_N \equiv_s \mathbb{Z}_N^*$.
- **Step (7)**: Notice that $\hat{z}_i = g^{y_i} \cdot a_i^{-c_i}, \hat{w}_i = (\hat{z}_i)^{2^{2^i}}$ while $z_i = g^{\alpha_i}$ and $w_i = (z_i)^{2^{2^i}}$. Therefore, the pairs z_i, w_i and $\hat{z}_i, \hat{w}_i$ distribute like z_i and $\hat{z}_i$. Since $a_i = g^{2^{2^i}}$, $\hat{z}_i = g^{y_i - c_i \cdot 2^{2^i}}$. If we show that $\Delta(U_N, U_N^*)$ is negligible, it will follow that $\Delta\left(z_i = g^{\alpha_i}, \hat{z}_i = g^{y_i - c_i \cdot 2^{2^i}}\right)$ is also negligible. To see why, assume in contradiction there exists a distinguisher $\mathcal{D}$ that distinguishes between $\{g^{\alpha_i}\}_{\alpha_i \in U_N^*}$ and $\left\{g^{y_i - c_i \cdot 2^{2^i}}\right\}_{y_i \in U_N^*}$. We can use $\mathcal{D}$ to build a distinguisher $\mathcal{D}'$ that distinguishes U_N from U_N^*: Upon input x, the distinguisher $\mathcal{D}'$ returns $\mathcal{D}(g^x)$.
- **Step (8)**: The simulator sends $\hat{y}_i \in \mathbb{Z}_N$ picked uniformly at random while the real prover sends a $y_i \in \mathbb{Z}_N^*$ also picked uniformly at random. Note that y_i, $\hat{y}_i$ are independent of the public input, and the correlation between $\hat{y}_i$ and $\hat{z}_i$ and $\hat{w}_i$ sent after rewinding is identical to the correlation of y_i with z_i, w_i in the corresponding real protocol. Therefore, as in step (3), it will suffice to show $\mathbb{Z}_N \equiv_s \mathbb{Z}_N^*$.

We denote a random variable that is sampled from $\mathbb{Z}_N$ as U_N, similarly U_N^* is sampled from $\mathbb{Z}_N^*$. We now next the indistinguishably of the random variables U_N and U_N^*. We are interested in measuring $\Delta(U_N, U_N^*)$ (the statistical distance between $\mathbb{Z}_N$ and $\mathbb{Z}_N^*$).

$$\Delta(U_N, U_N^*) = \frac{1}{2} \sum_{a=0}^{N-1} \left| Pr[a \in \mathbb{Z}_N] - Pr[a \in \mathbb{Z}_N^*] \right| \leq$$

$$\leq \frac{1}{2} \left(\sum_{a \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*} \left| \overbrace{Pr[a \in \mathbb{Z}_N]}^{=\frac{1}{N}} - \overbrace{Pr[a \in \mathbb{Z}_N^*]}^{0} \right| + \right.$$

$$\left. + \sum_{a \in \mathbb{Z}_N^*} \left| \overbrace{Pr[a \in \mathbb{Z}_N]}^{=\frac{1}{N}} - \overbrace{Pr[a \in \mathbb{Z}_N^*]}^{=\frac{1}{\phi(N)}} \right| \right) = (\bigstar)$$

Notice that:

$$|\mathbb{Z}_N^*| = \phi(N) = (P-1) \cdot (Q-1) = P \cdot Q - (P+Q) + 1$$

while $|\mathbb{Z}_N| = P \cdot Q$, therefore, $|\mathbb{Z}_N \setminus \mathbb{Z}_N^*| = P + Q - 1$.
Thus: $(\bigstar) = \frac{1}{2} \cdot \left(\frac{P+Q-1}{N} + \left(\phi(N) \cdot \left(\frac{1}{\phi(N)} - \frac{1}{N} \right) \right) \right) = \frac{1}{2} \cdot \left(\frac{P+Q-1}{N} + 1 - \frac{\phi(N)}{N} \right) = (\blacklozenge)$.
Notice that $\frac{\phi(N)}{N} = \frac{P \cdot Q - (P+Q) + 1}{N} = 1 - \frac{P+Q-1}{N}$, hence $1 - \frac{\phi(N)}{N} = \frac{P+Q-1}{N} \Rightarrow (\blacklozenge) = \frac{P+Q-1}{N}$.
Recall that both P and Q are λ bits of size. Hence, $\frac{P+Q-1}{N} \approx 2^{-\lambda}$, thus $\Delta(U_N, U_N^*)$ is negligible, and therefore $\mathbb{Z}_N \equiv_s \mathbb{Z}_N^*$ as claimed.

As we've shown that $\langle (a_i, b_i), \mathfrak{T}, \mathfrak{M} \rangle \equiv_s \langle (a_i, b_i), \hat{\mathfrak{T}}, \hat{\mathfrak{M}} \rangle$, we conclude that:

$$\left\{ View_{V^*}^{P}(a_i, b_i) \right\}_{x \in L} \equiv_s \{ S(a_i, b_i) \}$$

Appendix D. Proofs for AVTC sub-protocols

Proof of Theorem 17 ((LSB extraction completeness)). An honest dealer distributes only even plaintext shares and even blinding factors in lines 1-4, therefore the parties do not abort in lines 5,6. Furthermore, an honest dealer distributes plaintext shares and blinding factors such that for each blinding factor, there exist a plaintext share that was built using that blinding factor (line 4). It follows that in line 13, $[z]$ opens to zero, and the protocol does not abort. Finally, from construction of the plaintext shares of the LSB and its blinding factors, the parties actually compute a secret sharing of the LSB in line 11, and the protocol outputs it in line 17 since it does not abort.

□

Proof of Theorem 18 (LSB extraction privacy). The simulator $\mathcal{S}$ evaluates the circuit of **Algorithm 5** backwards, starting from the shares of $[v_0]$ of parties i and j. Note that the only place where secret shares are reconstructed, is when z is revealed to be zero. Therefore, in line 13, the simulator $\mathcal{S}$ samples three random shares z_1, z_2, z_3 and has each party broadcast the its share. Then, to simulate line 12 :

$$[z] = [v] - \sum_{i=1}^{n} \left[a^{(i)} + r^{(i)} \right] + \sum_{i=1}^{n} \left[r^{(i)} \right] - [v_0]$$

the simulator first samples v and then samples blinded plaintext shares and blinding values such that:

$$0 = v - \sum_{i=1}^{n} \left(a^{(i)} + r^{(i)} \right) + \sum_{i=1}^{n} r^{(i)} - v_0$$

where $\left(a^{(i)} + r^{(i)} \right)$ is a single value.

Afterwards, the simulator $\mathcal{S}$ secret-shares the blinded plaintext shares and blinding values, and adds to the incoming messages of parties i and j the secret shares accordingly.

□

Proof of Theorem 19 (LSB extraction soundness). Assume in contradiction that the protocol does not abort in spite of a dishonest dealer. It follows from lines 5,6 that for every party, $a^{(i)} + r^{(i)}$ is even, and $b^{(i)} + e^{(i)}$ is a bit. Since each blinding factor $r^{(i+1)}$ is even and $e^{(i+1)}$ is a bit, it follows that $a^{(i)}$ is even and $e^{(i)}$ is a bit. Furthermore it follows from line 5 that there is no overflow in the additions in line 12 since $p > N^2 + n \cdot 2^{\mu}$. Since we assumed the protocol didn't abort, $[z]$ was reconstructed to reveal zero in line 13 and there was no overflow (otherwise zero cannot be revealed). However, in line 12 the parties added and subtracted even values which did not change the LSB of the result, therefore the LSB of $[v]$ had to be $[v_0]$ which is returned, in contradiction to how a dishonest dealer should have operated. □

Appendix E. AVTC scheme for every predicate

Proof of Theorem 11 (Completeness). If the sender and receiver are honest, then indeed x, the secret input that $\mathcal{F}$ is evaluated on, has the required attribute, and every simulation of the MPC protocol of $\mathcal{F}$ on x will always yield $\mathcal{F}(x) = 1$. Furthermore, all commitments open correctly by both the sender and the receiver, and views are consistent with the messages sent due to the prover honestly emulating all executions of $\mathcal{F}$, therefore the receiver accepts.

□

Proof of Theorem 11 (Soundness). Since $x \notin L_{\mathcal{F}}$, the sender cannot simulate the MPC protocol of $\mathcal{F}$ honestly, and thus it has to cheat at at least a single party simulation among the n simulations. Observe that the sender commits to views in round 2, and reveals views selected by the verifier in round 4. The probability that a cheating sender will cheat in one of the three party simulations which the receiver is not going to ask to reveal in round 4 for all λ indices is $\left(\frac{1}{3}\right)^{\lambda}$, which is negligible for a large enough λ.

□

Proof of Theorem 11 (Zero Knowledge). We describe a simulator $\mathcal{M}$ and show that: $View_R^{S(x)} \equiv_s View_R^{\mathcal{M}}$. The simulator $\mathcal{M}$ uses an auxiliary function $\mathfrak{F}$ to construct the views and outgoing messages for each party i:

1) $\mathcal{M}$ receives a vector of commitments for indices from the verifier, sent in round 1.
2) $\mathcal{M}$ sends a commitment to zeroes in round 2.
3) Verifier decommits to indices $\{i_j\}_{j\in[\lambda]}$ it sent.
4) $\mathcal{M}$ rewinds the verifier back to right after round 1.
5) $\forall j \in [\lambda]$, $\mathcal{M}$ sends to R in round 2: $\{\mathcal{C}\left(\mathfrak{F}\left((i_j-1) mod\ 3\right)\right), \mathcal{C}\left(\mathfrak{F}\left((i_j+1)(mod\ 3)\right)\right), \mathcal{C}\left(0\right)\}$
6) $\mathcal{M}$ Receives indices sent in round 3, and decommits to views and outgoing messages forged by $\mathfrak{F}$.

The construction of $\mathfrak{F}$ is defined for the other two parties that are not i, meaning parties $i-1$ and $i+1$ that their view and outgoing messages are forged. Their output depends on the action performed by the party (gate type, or operation), the input to that action (gate input, private input) as well as the random tape. We define each action's output by emulating the protocol *backwards* gate by gate:
Share reconstruction: The public output $\mathcal{F}(x)$ for which the verifier accepts, is split into three uniformly random shares $s_{i-1\ mod\ 3}, s_i, s_{i-1\ mod\ 3}$ and let the parties of the views $i-1$, $i+1$ send their shares to each other. This is possible thanks to the fact that $\mathfrak{F}$ emulates the MPC protocol backwards, so the gates that define the shares of the output reconstruction have not yet been defined at this stage.
Multiplication: We recall the definition of the OLE based multiplication gate: To multiply $[z] = [x] \cdot [y]$, we use an Oblivious Linear Evaluation (OLE) oracle $\mathfrak{O}$: Every pair of parties i, j use the OLE oracle $\mathfrak{O}$ to compute blinded products of each party's inputs:
1) Every party i samples a random value $r_{i,j} \in \mathbb{Z}_p$ for each party $j \neq i$ and sends the share x_i and $r_{i,j}$ to $\mathfrak{O}$.
2) party j sends to $\mathfrak{O}$ the share y_j
3) The OLE oracle $\mathfrak{O}$ sends party j: $s_{i,j} = x_i \cdot y_j + r_{i,j}$.
Afterwards, $[z]$ is defined by:

$$(\bigstar)\ z_i = x_i + y_i + \sum_{j=1,j\neq i}^{n} (s_{j,i} - r_{i,j})$$

For a party $j \in \{i-1, j+1\}$ and w.l.o.g $j = i+1$, upon having a share z_j that is a result of a multiplication gate, we sample its inputs $x_j, y_j \in \mathbb{Z}_p$, sample the random tape entry $r_{j,i-1}$ to be sent to party $i-1$ and then do the same for party $i-1$. Then, we compute the OLE outputs to parties $j, i-1$ and sample random values for the OLE with party i (its view is not revealed) such that $(\bigstar)$ holds.
Secret sharing: Let s^j be a share that is in possession of party $j \in \{i-1, i+1\}$ at some point in the backward evaluation. By definition, the share was sent by a party in $\{j+1, j-1\}$. We split into two cases: (i) The share was sent by party i which its view isn't revealed. In that case, we add s^j to the incoming messages of j. (ii) The share was sent by the remaining party, one that is not i and also not j and its view is revealed. In that case, we sample a random value s' according to its domain (it might be a plaintext share, or a regular share in $\mathbb{Z}_p$) and then split s' into three parts: $\{s'_1, s'_2, s^j\}$ such that $s'_1 + s'_2 + s^j = s'\ (mod\ p)$, and then we set the output of the gate whose output is used as input to the secret sharing operation, and recursively call $\mathfrak{F}$.
LSB extraction: The LSB extraction involves additions (local operations), secret sharing, multiplications and share reconstructions, which were all defined previously for $\mathfrak{F}$.

□

Appendix F. Hash Pre-Image Prefix Length

A malicious party in our cross-chain atomic swap protocol is incentivised to find a pre-image for of a cryptographic hash $\mathcal{H}$ with the secret prefix of length d, as it can then claim its funds in escrow faster than the specified timeout, and then claim both the counter party's funds and not lose its own funds in escrow.

Security model and assumptions: We assume a model where malicious parties are computationally bound. Specifically, we assume that any malicious party that participates in the cross-chain atomic swap protocol, has hashing power less or equal to the entire hashing power of the Bitcoin network. We also put an upper bound to the delay used for any of the two parties in our protocol, which will be a single day.

We are only interested in protecting honest parties that follow the protocol, therefore we assume that they sample their pre-image prefix uniformly at random from $\{0,1\}^d$ as dictated by our protocol (see Section 4). We assume the standard cryptographic assumption on the hash function [41] we use, in particular that it is pre-image resistant, and let its input be 256 bits.

Analysis: Denote the daily hashing rate of a malicious party to be r. For simplicity, we model the computation by giving the malicious party oracle access to $\mathfrak{R}$:

$$\mathfrak{R} : \mathbb{N} \rightarrow \{(x, \mathcal{H}'(x)) \mid x \in \{0,1\}^d\}$$

where $\mathcal{H}'(x) = \mathcal{H}(x || \overbrace{0..0}^{256-d})$, a function that pads the input with zeroes and then applies $\mathcal{H}$. Informally, $\mathfrak{R}$ receives a number $r \in \mathbb{N}$ and then outputs r zero-padded pairs of pre-image prefixes and hashes, where each padding is of length $256 - d$.

We need to ensure that d will be large enough so that with high probability, none of the unique pre-image prefixes guessed by a malicious party during an upper bound time limit of a day, is the one sampled by the party that produced the AVTC commitment. We note that even though we require a suffix of $256 - d$ zeroes, we do not make it easier for an adversary to attack our scheme, as any adversary that can find a pre-image with a hash with a specific suffix of zeroes, finds a pre-image in the general case. The malicious party sends its daily hashing rate r to $\mathfrak{R}$ and receives back r distinct pairs: $\{(x_i, \mathcal{H}'(x_i))\}_{i=1}^{r}$. The probability that one of the pairs matches the hash that revealing its pre-image releases the escrow, is $\frac{r}{2^d}$.

Bitcoin average hash rate is 140 exa-hashes per second. If we upper bound the timeouts to a day, it sums up to $140 * 10^{18} \cdot 60 \cdot 60 \cdot 24 = 2^{83}$ double hash operations per day, or 2^{84} hash operations per day. Hence, if we take $d = 128$ then the probability of a malicious party that has access to a hashing rate that is equal to the entire Bitcoin network, is: $\frac{2^{84}}{2^{128}} = 2^{-44}$.

Appendix G. Extended experiments

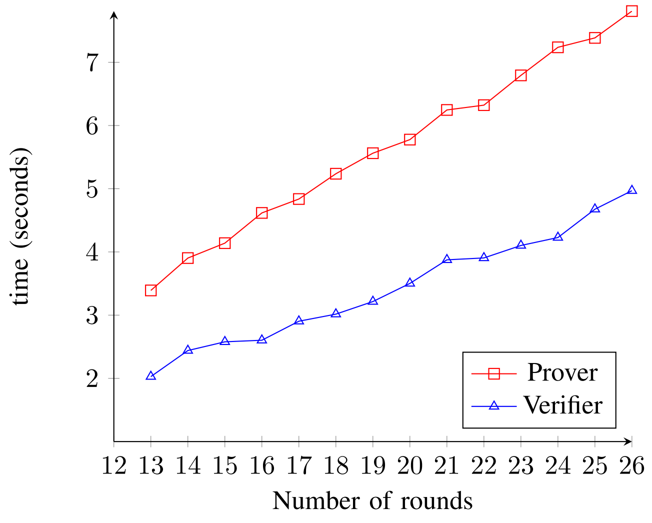


Figure 6: Performance measurement of SHA256-AVTC

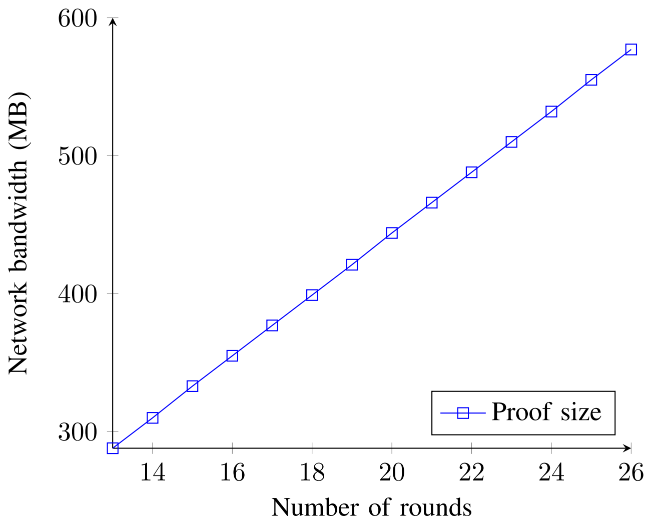


Figure 7: Network bandwidth of SHA256-AVTC

We show further measurements of our "MPC in the head" implementation of the SHA256 AVTC. We measure the performance (proving time and verification time) in **Figure 6**. The bandwidth sent from the prover to the verifier (in the de-commitment phase) as a function of with the number of rounds is shown in **Figure 7**. The correlation is linear since for every round, the prover de-commits all secret shares and OLE inputs/outputs in two of three views.4