

gOTzilla: Efficient Disjunctive Zero-Knowledge Proofs from MPC in the Head, with Application to Proofs of Assets in Cryptocurrencies

Foteini Baldimtsi
foteini@gmu.edu
George Mason University

Panagiotis Chatzigiannis
pchatzig@gmu.edu
George Mason University

S. Dov Gordon
gordon@gmu.edu
George Mason University

Phi Hung Le
ple13@gmu.edu
George Mason University

Daniel McVicker
dmcvicke@gmu.edu
George Mason University

ABSTRACT

We present gOTzilla, a protocol for interactive zero-knowledge proofs for large disjunctive statements of the following format: given publicly known circuit C , and set of values $Y = \{y_1, \dots, y_n\}$, prove knowledge of a witness x such that $C(x) = y_1 \vee C(x) = y_2 \vee \dots \vee C(x) = y_n$. These type of statements are extremely important for the proof of assets (PoA) problem in cryptocurrencies where a prover wants to prove the knowledge of a secret key sk that associates with the hash of a public key $H(pk)$ posted on the ledger.

gOTzilla is based on the MPC in the head (MPCitH) paradigm and is based on the observation that if we restructure the proof statement to an equivalent of proving knowledge of (x, y) such that $(C(x) = y) \wedge (y = y_1 \vee \dots \vee y = y_n)$, then we can reduce the disjunction of equalities to 1-out-of- N oblivious transfer (OT). We additionally provide a concrete, efficient extension of our protocol for the case where C combines algebraic and non-algebraic statements (which is the case in the PoA application). We achieve an asymptotic communication cost of $O(\log n)$ plus the proof size of the underlying MPCitH protocol. While related work has similar asymptotic complexity, our approach results in concrete performance improvements. We implement our protocol and provide benchmarks. Concretely, for a set of size 1 million entries, the total run-time of our protocol is 14.89 seconds using 48 threads, with 6.18 MB total communication, which is about 4x faster compared to the state of the art when considering a disjunctive statement with algebraic and non-algebraic elements.

1 INTRODUCTION

A zero-knowledge (ZK) proof [27] allows a prover \mathcal{P} to convince a verifier \mathcal{V} that a statement x is true without revealing any further information. ZK proofs have numerous applications: they are used as a building block in various cryptographic constructions such as secure multiparty computation [26], signatures [11] and anonymous credentials [16] just to name a few, and more recently they have been used as a core component in privacy-preserving cryptocurrencies [13].

ZK proofs can be constructed generically for any NP language [26], however such generic constructions are usually not efficient. In order to achieve practical constructions, customized ZK proofs have been designed for specific languages

(e.g. particular algebraic statements), or with specific optimization goals (e.g. proof size succinctness or non-interactiveness). Many different approaches have been proposed, each with different trade-offs on the types of supported languages, efficiency goals and underlying assumptions. In terms of efficiency, the tradeoffs appear in the prover complexity, the verifier complexity, and the communication costs.

Our focus. In our work, we focus on zero-knowledge proofs that can efficiently support very large *disjunctive* statements. We are inspired by the Proof of Assets (PoA) problem in UTXO based cryptocurrencies where a prover (usually some exchange or other organization) wishes to convince a verifier that it knows the respective private keys of *at least* a certain number of coins on the blockchain, without revealing which those coins are. In Bitcoin, and other cryptocurrencies with similar structure, PoA can be expressed as a *disjunctive* ZK proof where the statement is a set of hashed public keys (often called addresses), and the witness is one or more secret keys that correspond to some of the public keys. The challenge when computing a ZK proof for PoA is bifold: (a) the size of the statement grows with the total size of the Bitcoin UTXO set, which has hundreds of millions of elements, and (b) the statement is a combination of an algebraic circuit – the discrete log relation between (sk, pk) – and a Boolean hash function, since the prover needs to prove that it knows the secret key for one of the hashed public keys. Concrete protocols for the PoA application have been designed in the literature [5, 21], but as explained in related work (Section 1.1), they fall short in addressing the two main design challenges simultaneously.

Our Construction. Focusing on the first challenge of dealing with very large statements, we begin with a simple observation. We are interested in statements of the following form. Given some publicly known circuit C , and some publicly known set of values $Y = \{y_1, \dots, y_n\}$, the prover wishes to prove that it knows a witness x such that $C(x) = y_1 \vee C(x) = y_2 \vee \dots \vee C(x) = y_n$. A simple restructuring of this statement allows us to remove the n copies of C , greatly reducing the statement size. The prover witness is modified to be a pair of values, (x, y) , such that $(C(x) = y) \wedge (y = y_1 \vee \dots \vee y = y_n)$. As we discuss below, this provides significant improvement even for existing proof systems that support disjunctions, as

the size of an equality circuit is much smaller than $|C|$.^{*} Once re-written in this form, we are able to reduce the disjunction of equalities to 1-out-of- n OT.

We build a ZK proof using the MPC in the head paradigm (MPCitH). Our main observation is that when proving that y is equal to one of the elements in the public set Y , it suffices to enforce constraints on the Prover’s set of inputs to the MPC in the head. We do that by having the Verifier prepare an encoding of all n possible inputs y_i for the MPCitH and having the Prover select a single input encoding obliviously, using 1-out-of- n OT. The Verifier creates these encodings such that the portions of the encoded input that are revealed to the verifier in the opened MPCitH views are identical for all y_i . This ensures that the view can be safely opened for verification, without revealing the index i . We implement 1-out-of- n OT using Private Information Retrieval (PIR). Note that PIR is a relaxation of 1-out-of- n OT, in that it potentially allows the receiver to learn more than one value. We strengthen PIR to OT by performing a zero-knowledge proof on the Prover’s PIR query to show that it is well-formed (i.e. a valid ciphertext encrypting a query for only a single database element). In our 1-out-of- n OT protocol (as well as in the rest of the protocol) we enforce honest Verifier behavior by committing to, and later revealing, the Verifier’s random tape, allowing the Prover to check that the Verifier’s messages have all followed the protocol. We can only do this because the Verifier’s inputs are random challenges that do not require privacy beyond the end of the protocol; for general purpose 1-out-of- n OT, this approach cannot be used for ensuring the honest behavior of the sender. By building 1-out-of- n OT from PIR, we achieve communication complexity of $O(\log n)$, which allows us to achieve an overall communication cost of $O(\log n) + \Pi_{proofsize}$ where $\Pi_{proofsize}$ denotes the proof size of the MPCitH protocol. We present our protocol in Section 4.

ZK Proofs on Mixed Statements. An efficient disjunctive proof however, is not enough to efficiently prove the concrete PoA statement. One could convert our algebraic statement (on the relation between (sk, pk)) to a Boolean circuit, but as we discuss later in Section 6 these would result in a circuit with millions of gates. A number of works examined the problem of efficiently combining algebraic to non-algebraic statements in a ZK proof. Chase et al. [18] provided one of the first techniques based on Garbled Circuits, MACs and Oblivious Transfer which was further optimized in [5], and later Backes et al. [8] presented a technique on an MPCitH Σ -protocol inspired from ZKBoo [24]. In Section 5 we present an extension of our disjunctive proof for mixed statements based on Oblivious Transfer and MPC-in-the-head, rather than the less efficient Garbled Circuit approach of Chase et al. [18]. Finally, in the Appendix we show an extension of our mixed statements protocol that also proves the value that corresponds to the secret key (i.e. witness) of the Prover (a property needed towards designing a PoA protocol), which utilizes the same technique. **Evaluation Results.** We evaluate gOTzilla on top of an existing PIR implementation, namely SealPIR [2], by implementing

^{*}Generically, a proof system that efficiently supports disjunctions might not support the conjunction of C with the disjunction of n equalities. In practice, existing systems seem to handle this change without significant complication.

our techniques to derive a 1-out-of- n OT protocol, while treating the underlying MPCitH protocol as a black box. Our results are presented in detail in Section 6, where we show that we can prove knowledge over a disjunctive statement of $n = 2^{20}$ elements in 14.89 seconds, with 6.18 MB of communication and 10 seconds network latency at the worst case, for statistical security parameter $\lambda = 40$. Our evaluation shows a significant improvement in total protocol run-time over Mac’n’Cheese [10] the state of the art in disjunctive proofs, which has similar asymptotic costs to us as discussed in the Related Work section below.

1.1 Related Work

We provide a short, non exhaustive, overview of common ZK proof types for disjunctive proofs and mixed statements, as well as an overview of solutions specific to the PoA problem. **Standard ZK Techniques.** Σ -protocols form a well studied class of efficient protocols specifically for algebraic statements, such as discrete logarithms and roots [29, 36], while garbled-circuit approaches were used to efficiently prove Boolean circuits [32]. We note that if one attempted to use a Σ -protocol to prove a statement about a function represented as a Boolean (or arithmetic) circuit C , both proving and verification costs would grow linearly with the size of the circuit (a simple SHA256 evaluation would result in tens of thousands of exponentiations) which makes them prohibitive for a PoA like statement.

ZK-SNARKs (Succinct Non-Interactive Arguments of Knowledge) [12, 28, 35] is another well studied type of ZK proofs that received a lot of attention the last few years due to their use in private cryptocurrencies [13].

Their goal is to offer constant, succinct proof sizes and short verification times. In particular, ZK-SNARKs can be verified in time that is linear in the length of the input x , rather than the length of the circuit C . However, they suffer from large prover overhead, since they require the prover to perform a large number of public-key operations that is proportional to the size of the circuit representing the statement. ZK-SNARKs are better suited for Boolean or arithmetic circuits and while they could be used for algebraic statements, they would require circuits with thousands or millions of gates for a simple computation like an exponentiation exploding the prover’s cost. Finally, many ZK-SNARKS constructions, require an additional trust assumption. Namely, to guarantee soundness, they need a common reference string (CRS) that is generated ahead of time by a trusted party (or a distributed protocol). Some recent works [6, 12, 33, 37, 38] use techniques such as interactive oracle proofs (IOP), vector oblivious linear evaluation (VOLE) and the MPC in-the-head paradigm and do not rely on a setup phase, yet, they still impose very high computational costs on the prover side thus are not directly relevant to the considered PoA application.

Disjunctive ZK Proofs. A number of related works have examined the general problem of building efficient disjunctive ZK proofs. Following the seminal work by Cramer et al. on constructing standard disjunctive proofs [20], Stacked Garbling [30] proposed a garbled-circuit approach for creating a disjunctive proof with sublinear communication complexity,

based on Jawurek et al. [32]. Later, Stacking Sigmas [25] provided a generic compiler for reducing communication complexity (i.e “stacking”) of disjunctive Sigma-protocols satisfying a specific “stackable” property, and is compatible with recent MPCitH style ZK protocols such as KKW [33] and Ligerio [6].

Recently, Mac’n’Cheese [10] proposed a new, VOLE based approach to build generic zero-knowledge proofs for disjunctive statements of the form $(x, i) : C_i(x) = y_i$ for $i \in \{1, \dots, n\}$ with communication cost of $\max_i \{|C_i|\} + \log n$. In the general case where each C_i is a different circuit, both the prover and verifier have to execute all branches to construct or verify the proof, causing the total computation cost to be $O(\sum_{i=1}^n |C_i|)$. In the special case where all the circuits C_i are identical, using our observation above that restructures the disjunctive portion, their construction can be slightly modified to improve the computational cost to $O(n + |C|)$.

In Table 1 we provide a comparison of basic techniques for disjunctive statements. We note that although asymptotically we might have similar performance as Mac’n’Cheese [10] or Stacking Sigmas [25] combined with a suitable MPCitH protocol II, we have significant concrete improvements. In Section 6.1 we present a concrete comparison of our disjunctive protocol with Mac’n’Cheese [10] to showcase our improvement by 4x in runtime for the case of “mixed” disjunctive statements. We note that there is no available implementation of Stacking Sigmas [25] for a direct comparison, however Stacking sigmas is expected to be more expensive than Mac’n’Cheese concretely due to its underlying techniques (Stacking sigmas relies on commitments with elliptic curve operations which are more expensive than VOLE used in Mac’n’cheese). We note that a caveat of our approach is that we generally have larger memory requirements as opposed to Mac’n’Cheese the prover and verifier are not required to store the entire proof statement in memory. However, as we discuss in Section 6.1, gOTzilla can optimize RAM usage by generating the required data on the fly as needed to improve our storage costs.

Proof of Assets. Provisions [21] was the first attempt to create an assets proof on behalf of a cryptocurrency exchange, as a part of a general solvency proof. The proof was constructed using standard Σ -protocols, which however was only compatible with unhashed public keys (P2PK), thus severely limiting both the anonymity set and its practical use. Agrawal et al. [5] made proving assets with hashed public keys possible as part of a zk-SNARK-based protocol combined with Σ -protocols (in a CRS model based on Pinocchio [35]), tailored for mixing arithmetic and boolean components. However, in addition to the setup assumptions, this approach is not efficient for large disjunctive statements (the size of the UTXO list in Bitcoin is in the order of hundreds of millions) as as both the computational and space requirements scale linearly with its size, its has expensive concrete costs for the prover because of the underlying SNARKs and range proofs.

2 PRELIMINARIES

2.1 Notation.

We denote an n -dimensional vector $\mathbf{v} = \{v_1, \dots, v_n\}$. By $a||b$ we denote concatenation of elements a and b . A probabilistic polynomial-time (PPT) algorithm B with input a and output b is written as $b \leftarrow B(a)$. By \bigoplus we define a bit-wise XOR operation. We define the statistical security parameter by λ .

2.2 Basic Cryptographic Building Blocks

Commitment Schemes. A commitment protocol between a committer and a receiver consists of two phases: a committing phase where the committer on input a message m and public parameters pp , samples randomness r and computes a commitment $C_m \leftarrow \text{Com}(pp, m, r)$, and a de-committing or opening phase where the committer de-commits C_m to m . A commitment scheme should be *hiding*, i.e. C_m should not reveal any information about m and *biding*, i.e. it should be hard for the committer to find m' such that $\text{Com}(pp, m, r) = \text{Com}(pp, m', r')$ with $m' \neq m$. In the rest of the paper, we imply inputs pp and r and omit them for simplicity.

MACs. A circuit-based one-time Message Authentication Code (MAC) on x is defined as $t = ax + b$ where a and b are randomly sampled by the verifier, and can be opened after the prover has committed to t [18].

Homomorphic Encryption. An (additive) homomorphic encryption scheme is public-key encryption scheme equipped with an operation \boxplus over the ciphertext space such that for any two plaintexts a, b , $\text{Dec}(\text{Enc}(a) \boxplus \text{Enc}(b)) = a + b$.

2.3 Zero-knowledge Proofs

A zero-knowledge (ZK) proof π enables a prover P who holds some private witness w for a public instance x and an NP-relation R , to convince a verifier V that some property of w is true i.e. $R(x, w) = 1$, without V learning anything more. To denote a ZK proof statement we use the Camenisch-Stadler notation [17] as $\pi = \{(w) : R(x, w) = 1\}(x)$.

DEFINITION 1. A zero-knowledge proof between P and V for an NP relation R must satisfy the following properties:

- **Completeness:** If $R(x, w) = 1$ and both players are honest V always accepts.
- **Soundness:** For every malicious and computationally unbounded P^* , there is a negligible function $\epsilon(\cdot)$ s.t. if x is a false statement (i.e. $R(x, w) = 0$ for all w), after P^* interacts with V , $\Pr[V \text{ accepts}] \leq \epsilon(|x|)$.
- **Zero Knowledge:** For every malicious PPT V^* , there exists a PPT simulator \mathcal{S} and negligible function $\epsilon(\cdot)$ s.t. for every distinguisher D and $(x, w) \in R$ we have $|\Pr[D(\text{View}_{V^*}(x, w)) = 1] - \Pr[D(\mathcal{S}) = 1]| \leq \epsilon(|x|)$.

Composed statements. ZK proofs can be composed as follows: (1) AND composition $\pi_1 \wedge \pi_2$ which can be easily constructed by sequential or parallel proving of underlying assertions, and (2) OR composition $\pi_1 \vee \pi_2$ which can be constructed by proving knowledge for the one and simulating knowledge for the other, without revealing which of the two is actually proved and which is simulated.

Table 1: Asymptotic comparison of disjunctive ZK proof systems for n statements for a single circuit C . NI = Non-Interactive. Π denotes an MPCitH protocol, $\Pi_{Runtime}$ and $\Pi_{Proofsize}$ denote Runtime and Proofsize of Π , respectively.

	No setup	NI	Prover Runtime	Proof size
Ligero [6]	✓	✓	$O((n + C) \cdot \log(n + C))$	$O(\sqrt{n + C })$
Π + Stacking Sigmas [25]	✓	✓/✗	$O(n) + \Pi_{Runtime}$	$O(\log n) + \Pi_{Proofsize}$
Mac'n'cheese [10]	✓	✗	$O(n + C)$	$O(\log n + C)$
gOTzilla	✓	✗	$O(n) + \Pi_{Runtime}$	$O(\log n) + \Pi_{Proofsize}$

Mixed statements. Let f and g be non-algebraic and algebraic relations with public instances y and z respectively. A ZK proof on a mixed statement has the generic form $\{(w) : f(y, w) = 1 \wedge g(z, w) = 1\}(y, z)$.

2.4 Oblivious Transfer

1-out-of-2 oblivious transfer (OT) is a fundamental functionality in secure computation between a sender S that holds two values v_0, v_1 and a receiver R . At the end of the protocol, the receiver learns exactly one of the sender values while the sender learns nothing. *1-out-of- n OT* is a generalized version of 1-out-of-2 OT where the sender has n values, and the receiver learns one of them. In Figure 1 we describe the ideal functionality for 1-out-of- n OT.

Functionality $\mathcal{F}_{OT}^{1:n}$
Functionality $\mathcal{F}_{OT}^{1:n}$ communicates with sender S and receiver R , and adversary \mathcal{A} .
(1) Upon receiving input (sid, v_1, \dots, v_n) from S where $v_i \in \{0, 1\}^\kappa$, record (sid, v_0, \dots, v_n) .
(2) Upon receiving (sid, i) from R where $i \in \{1, \dots, n\}$, send v_i to R . Otherwise, abort.

Figure 1: The 1-out-of- n OT functionality.

2.5 MPC in the Head

We use the MPC-in-the-Head paradigm introduced by Ishai et al. [31]. An MPC protocol $\Pi_{\mathcal{F}}$ is an interactive protocol between m parties P_1, \dots, P_m to securely compute some function \mathcal{F} on the joint input of all parties. In MPCitH a single party simulates the execution of all m parties locally and records transcripts of the interaction between the simulated parties. These simulated views can later be selectively opened to prove statements about the inputs of the simulated parties.

Formally, we require the following properties for $\Pi_{\mathcal{F}}$ to be an admissible protocol for MPCitH:

DEFINITION 2. Let $\Pi_{\mathcal{F}}$ be an MPC protocol for a functionality $\mathcal{F}(x_1, \dots, x_m)$.

- We say $\Pi_{\mathcal{F}}$ realizes \mathcal{F} with correctness if for all possible inputs the probability that the output of any party P_j running (semi-honest) $\Pi_{\mathcal{F}}$ is different from $\mathcal{F}(x_1, \dots, x_m)$ is negligible in λ .
- We say $\Pi_{\mathcal{F}}$ realizes \mathcal{F} with t -privacy if for all sets of (semi-honest) corrupt parties $I \subset \{P_1, \dots, P_m\}$ s.t. $|I| \leq t$ there exists a PPT simulator \mathcal{S} s.t. for all inputs the set of views $\{view_j\}_{j \in I}$ is statistically indistinguishable from $\mathcal{S}(I, \{x_j\}_{j \in I}, \mathcal{F}(x_1, \dots, x_m))$.

We model the local simulation of the MPCitH protocol with the black-box functionality Π_{MPCitH} in Figure 2.

Π_{MPCitH}
Input: An m -party MPC protocol $\Pi_{\mathcal{F}}$ implementing the functionality \mathcal{F} which takes as input (X_j, Y_j) from each party P_j and outputs to all parties $\mathcal{F}(\oplus_{j=1}^m X_j, \oplus_{j=1}^m Y_j)$ with completeness and $(m-1)$ -privacy, (X_j, Y_j) for $j = 1, \dots, m$
Protocol: Run a simulation of $\Pi_{\mathcal{F}}$ as follows: <ul style="list-style-type: none"> • Set each party P_j's input as (X_j, Y_j) • Sample P_j's initial randomness r_k • Set $view_j \leftarrow \{X_j, Y_j, r_j\}$ • Execute the steps $\Pi_{\mathcal{F}}$, adding each message received by P_j to $view_j$ • Add the output to each $view_j$
Output: $(view_1, \dots, view_m)$

Figure 2: MPC-in-the-Head subroutine

3 OBLIVIOUS TRANSFER FROM PRIVATE INFORMATION RETRIEVAL

A Private Information Retrieval (PIR) protocol [34] between a receiver R and a server S which owns a database D consisting of items y_1, \dots, y_n , enables R to retrieve some item y_i from D without S learning any information about i . Intuitively, PIR is similar to a 1-out-of- n OT protocol, with the main difference being that it only protects the privacy of R 's input and assumes semi-honest behavior for both parties. In this section we construct a protocol for 1-out-of- n OT built on top of SealPIR [7]. Note that this construction cannot generically be applied to arbitrary PIR protocols, as it relies on properties of SealPIR's construction.

Privacy against semi-honest receiver. SealPIR is constructed from the additive homomorphic encryption scheme BFV [14, 23] based on Ring-LWE. As privacy is not a concern in a PIR protocol, SealPIR packs many y_i to fully utilize the large plaintext supported by Ring-LWE, and computes $f(b, y) = \sum_{j=1}^{n/k} b_j \cdot Y_j$ where $Y_j = (y_{1+(j-1)k}, \dots, y_{jk})$, k is the number of y_i 's that can be fitted into one plaintext, $b_j = 1$ if the selected item is in $[1 + (j-1)k; jk]$ and $b_j = 0$ everywhere else. In the protocol, b_j 's are encrypted and compressed by the receiver, decompressed by the sender who sends back the encrypted $f(b, y)$. Finally the receiver decrypts the ciphertext and obtains $f(b, y) = Y_j$.

We observe that without packing the y_i , the protocol actually computes $f(b, y) = \sum_{j=1}^n b_j \cdot y_j$ and realizes a semi-honest

1-out-of- n OT protocol (with less efficiency if the plaintext has too much empty space).

Security against a malicious receiver. To achieve security against malicious receivers, after sending its query the receiver performs a zero-knowledge proof that the query is “well-formed” (i.e. is an encryption of a plaintext with exactly 1 nonzero index). We describe this protocol in figure 3.

First, $\Pi_{ZK}^{WellFormed}$ guarantees that the encrypted query c_v is a correctly-constructed ciphertext of a known plaintext with bounded noise using techniques proposed by Chen et al. [19]. The rest of the protocol proves that the underlying plaintext is a well-formed.

The server creates a challenge by sampling a random element r_i and random polynomial Q_i and an $(n - 1)$ -out-of- n shamir-sharing of Q_i (denote the vector of shares as q_i). The server then homomorphically computes $c_i := r_i \cdot c_b + Enc(q_i)$ and sends c_i to the receiver.

If the query is well-formed, then the decryption of c_i will have enough unmodified shares to reconstruct Q_i . More specifically, the decrypted plaintext will contain $n - ||b||$ shares of Q_i where $||b||$ is the number of nonzero elements in the plaintext query. Hence if the query contains > 1 non-zero elements the receiver is unable to reconstruct Q_i . We repeat this process (in parallel) to achieve the desired level of soundness.

Security against a malicious sender. To make $\Pi_{ZK}^{WellFormed}$ malicious-secure, we observe that the server only needs to keep Q_i and r_i private until the receiver has sent its response. Additionally, once the receiver knows Q_i , r_i and the randomness used to encrypt q_i it can deterministically recompute the honestly-generated c_i to verify honest behavior of the server.

Along with the challenges c_i , the server now sends a commitment to a PRG seed s from which all other random values are sampled. The receiver commits to its responses, then the server opens the seed to the receiver. The receiver recomputes the challenges and verifies that they match what the server originally sent. If so, the receiver opens its responses to the server.

For the overall 1-out-of- n OT protocol we apply a similar methodology. First the server commits to its PRG seed and database input, and later opens this commitment so that the receiver may check for honest behavior. In order to preserve the receiver’s input privacy, this check must occur before any computations based on the received value are revealed to the sender. To preserve soundness, it must occur after all relevant prover computations have been run and their outputs committed. Because of this, our implementation of $\Pi_{OT}^{1:n}$ only attains security against semi-honest S and malicious R . When using $\Pi_{OT}^{1:n}$ in a larger protocol, we augment it with the PRG trick to reach malicious S security. As an optimization, we use a single seed for all instances of 1-out-of- n OT, allowing the server to reveal the databases for all instances simultaneously.

THEOREM 1. *Protocol $\Pi_{ZK}^{WellFormed}$ is a Zero Knowledge proof that an encrypted PIR query $Enc(\mathbf{b})$ satisfies the condition: $\nexists(i \neq j)$ s.t. $b_i \neq 0 \wedge b_j \neq 0$.*

PROOF (SKETCH). *Soundness:* If the prover cheats by setting more than one entry of \mathbf{b} to be non-zero, it will not have enough information to reconstruct \widetilde{Q}_i . As a_{ij} are sampled

uniformly at random, $Q_i(0) = a_{i,0}$ is also uniformly random. In order to pass the check, the prover can only guess $a_{i,0}$ and has the probability of $1/t$ to guess it correctly. In overall, the chance that the verifier passes the check if it cheats is $t^{-\sigma} < 2^{-\lambda}$.

Zero-knowledge: It is clear that the encryption of \mathbf{b} preserves the privacy of the selection index. This is due to the property of the encryption scheme. In the protocol, the prover only reveals $Q'_i(0)$ after seeing the seed used to generate the challenges by the verifier. Thus, the verifier has no way to deviate from the protocol without being caught. If the verifier abides to the protocol, it learns nothing from the answers. If the verifier cheats, it will not see the answers, thus, there is no risk of leaking information to the verifier due to selective failure attacks. \square

$\Pi_{ZK}^{WellFormed}$

Setup. Ring-LWE scheme with parameters (N, t, q) where N is the degree of the cyclotomic polynomial, t the plaintext modulus, and q the ciphertext modulus. The prover has the key pair (sk, pk) , while the verifier has the public key pk . σ is the soundness amplifier such that $t^{-\sigma} < 2^{-\lambda}$.

Prover’s input. $\mathbf{b} \in R_t[X]/(X^N + 1)$ and $k \in [0, N)$ such that $b_k \neq 0$ and $b_i = 0 \forall i \neq k$.

Commm input. $c_b = Enc(pk; \mathbf{b})$ where $\mathbf{b} \in R_t[X]/(X^N + 1)$.

Protocol.

- (1) The prover sends a proof on the validity of ciphertext c_b which includes that the Ring-LWE noise is bounded.
- (2) The verifier samples a random seed $s \in \{0, 1\}^\kappa$. For $i \in \{1, \dots, \sigma\}$ the verifier samples $r_i, Q_i(X) \leftarrow PRG(s)$ where $r_i \in \mathbb{Z}_t$, $Q_i(X) = \sum_{j=0}^{n-1} a_{ij}X^j$, $a_{ij} \leftarrow \mathbb{Z}_t$, and computes $\mathbf{q}_i = (Q_i(1), \dots, Q_i(N)) \in \mathbb{Z}_t^N$. It uses the additive homomorphic property of Ring-LWE to compute $c_i \leftarrow Enc(pk, r_i \cdot \mathbf{b} + \mathbf{q}_i)$. After that, the verifier sends $Com(s)$ and c_i to the prover.
- (3) The prover decrypts c_i , obtains $c_i = r_i \cdot \mathbf{b} + \mathbf{q}_i$, interpolates Q'_i from the points (j, c_{ij}) , where $j \in \{1, \dots, N\}$, $j \neq k$. It then sends $Com(Q'_i(0))$ to the verifier.
- (4) The verifier decommits s to the prover.
- (5) The prover verifies that c_i is correctly generated. If so, it decommits $Q'_i(0)$ to the verifier.
- (6) The verifier checks that $Q'_i(0) = a_{i,0}$.

Figure 3: Zero-knowledge proof to prove that the encrypted ciphertext c_v is well formed and at most one of $b_i \neq 0$.

THEOREM 2. *Let Com be a binding and hiding commitment scheme, and let $SealPIR$ be the protocol described in [7], modified to not use any packing. Then the protocol described in Figure 4 implements $\mathcal{F}_{OT}^{1:n}$ with security against a malicious receiver and semi-honest sender in the Com -hybrid model.*

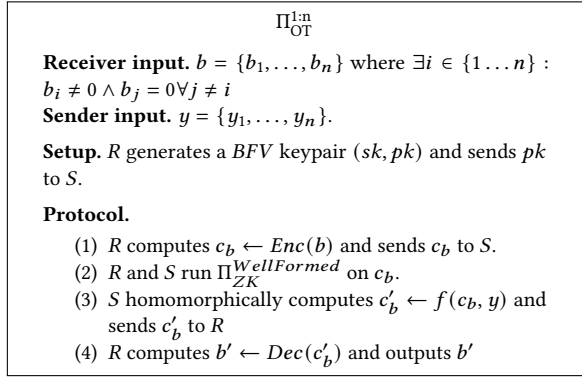


Figure 4: 1-out-of- n OT protocol

PROOF. (Sketch) The security against semi-honest senders follows directly from the semantic security of the cryptosystem and the zero-knowledge property of $\Pi_{ZK}^{WellFormed}$.

As for the security against malicious receivers, the soundness of $\Pi_{ZK}^{WellFormed}$ ensures that b is a valid query. Given that b is a valid query (i.e. only one i is nonzero) the sender's response $c'_b = f(c_b, y) = Enc\left(\sum_{j=1}^n b_j \cdot y_j\right) = Enc(b_i \cdot y_i)$.

We construct a simulator S which interacts with R and the ideal functionality (shown in figure 5). The indistinguishability of R 's view when interacting with S in the real world versus R 's view when interacting with S in the ideal world follows directly from the soundness of the $\Pi_{ZK}^{WellFormed}$ ZKPoPK subprotocol [19].

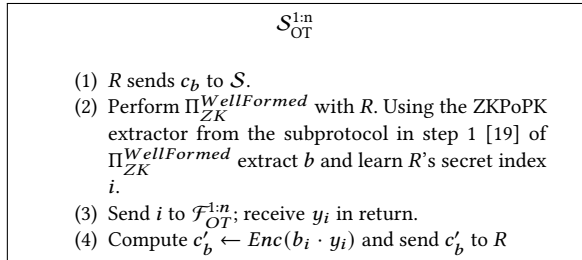


Figure 5: Malicious-receiver simulator for $\Pi_{OT}^{1:n}$ □

4 DISJUNCTIVE PROOFS FROM 1:N OT

4.1 MPC-ItH Disjunctive Proof

Towards our goal of constructing an efficient disjunctive proof, we first define a helper function $ShareAt(m, x, J, r)$ which pseudorandomly samples m -out-of- m additive secret shares of x from the seed r , outputting a vector of values $\{X_1, \dots, X_m\}$ such that $\bigoplus_{j=1}^m X_j = x$ (we omit m as explicit input to $ShareAt$ for the rest of the paper). In addition, $ShareAt$ has the property that for a fixed index $J \in \{1, \dots, m\}$ and r , $ShareAt(m, x, J, r)$ and $ShareAt(m, x', J, r)$ will have identical outputs at every index except the J -th index. This function is shown in Figure 6.

We define our main protocol $\Pi_f^{MPCitH-OR}$ in Figure 7. The common input is a function f , and a set of values y_1, \dots, y_n .

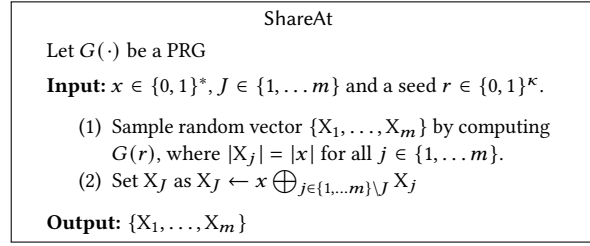


Figure 6: Secret sharing with specific offset index

The Prover wishes to prove, in zero knowledge, that it knows input (x, y) such that $f(x, y) = 1 \wedge y \in \{y_1, \dots, y_n\}$. We let $\Pi_{\mathcal{F}}$ denote an m -party protocol for securely computing $f(\bigoplus_j x_j, \bigoplus_j y_j)$. The verifier, \mathcal{V} , begins by generating and committing to a PRG seed s from which all further random values are sampled. This seed will allow the prover to confirm the verifier's honest behavior using the technique described in section 3.

Next, the verifier \mathcal{V} generates τ different sets of input encodings – each containing m shares – for each of the y_i common input values. This results in a 3D table of shares $(n \times m \times \tau)$, denoted Y_{ij}^k , as shown in Figure 8. This 3D table is divided into n 2D “slices” of dimension $(m \times \tau)$, each corresponding to an input y_i . These slices are encodings of y_i such that all slices are identical in every position except for 1 per row (τ non-identical positions total). The non-identical position for the k -th row is denoted ε_k .

Then, prover \mathcal{P} selects a “slice” of this 3D table through the 1-out-of- n OT protocol; the choice of the slice is its secret input, ℓ .

For each row of the slice (i.e. a chunk which was generated from the same value of k and adds up to y_ℓ) \mathcal{P} generates additive shares of x and computes MPCitH views using the shares of y_ℓ from the slice and the newly-generated shares of x as input. Once all views are generated, \mathcal{P} commits these views to \mathcal{V} .

Finally, \mathcal{V} reveals the committed seed s to \mathcal{P} , who re-generates the whole 3D table, and verifies the honest behavior of the verifier. \mathcal{P} aborts if there is a mismatch, otherwise \mathcal{P} decommits the MPCitH views (except Party ε_k) to \mathcal{V} who verifies the honest execution of the MPCitH protocol, and accepts the proof.

Note that since the slices are identical in each row except for at ε_k (i.e. the unrevealed MPCitH input) the privacy of the MPCitH protocol protects the prover from revealing its choice of index ℓ .

THEOREM 3. *Let $m \geq 3$, let $f(Y, X)$ be the function defined in Figure 7, let Com be a binding and hiding commitment scheme, let $\Pi_{\mathcal{F}}$ implement f with correctness and $(m - 1)$ -privacy, and let $\Pi_{OT}^{1:n}$ implement $\mathcal{F}_{OT}^{1:n}$ with security against a malicious receiver and a semi-honest sender. Then the protocol described in Figure 7 is a zero-knowledge proof protocol for the language $\{(y_1, \dots, y_n), x) : \exists \ell \in \{1, \dots, n\} \text{ such that } f(x, y_\ell) = 1\}$ in the $(Com, \mathcal{F}_{OT}^{1:n})$ -hybrid model.*

PROOF. Zero Knowledge: First we claim that the security of the protocol in the presence of a dishonest verifier reduces to

$$\Pi_f^{\text{MPCitH-OR}}$$

Setup. Let $f(x, y)$ be some function, and let \mathcal{F} be an m -party functionality that takes input (X_j, Y_j) from each party \mathcal{P}_j and outputs $f(\bigoplus_{j=1}^m X_j, \bigoplus_{j=1}^m Y_j) \stackrel{?}{=} 1$ to all parties. Let $\Pi_{\mathcal{F}}$ be an m -party protocol that securely realizes \mathcal{F} with correctness and $(m-1)$ -privacy.

Common inputs. τ total number of repetitions, n values $\{y_1, \dots, y_n\} \in \{0, 1\}^\kappa$, and m , the number of parties involved in the MPC protocol, run in the head of the Prover, and $m^{-\tau} < 2^{-\lambda}$, λ is a security parameter.

Prover's input. $x \in \{0, 1\}^*$ such that $f(x, y_\ell) = 1$ for some $\ell \in \{1, \dots, n\}$.^a

- (1) Verifier \mathcal{V} generates random seed s and sends $C_s \leftarrow \text{Com}(s)$ to the prover \mathcal{P} . Throughout the rest of the protocol, all randomness of the Verifier is generated by applying a PRG, $G(s)$. (We will, imprecisely, refer to these as “random” values.)
- (2) \mathcal{V} uses the random seed s to sample the following values uniformly at random:
 - (a) $\varepsilon_k \xleftarrow{\$} \{1, \dots, m\}$ for $k \in \{1, \dots, \tau\}$.
 - (b) $r_k \xleftarrow{\$} \{0, 1\}^\kappa$ for $k \in \{1, \dots, \tau\}$.
- (3) For $i \in \{1, \dots, n\}, k \in \{1, \dots, \tau\}$, \mathcal{V} compute vector $\{Y_{i,1}^{(k)}, \dots, Y_{i,m}^{(k)}\} \leftarrow \text{ShareAt}(y_i, \varepsilon_k, r_k)$, which results in a 3D table, as in Figure 8. (Note that the same r_k is used for every y_i .)
- (4) For every $i \in \{1, \dots, n\}$, \mathcal{V} sends the 2D table $\{\{Y_{i,1}^{(1)}, \dots, Y_{i,m}^{(1)}\}, \dots, \{Y_{i,1}^{(\tau)}, \dots, Y_{i,m}^{(\tau)}\}\} = \{Y_i^{(1)}, \dots, Y_i^{(\tau)}\}$ to $\Pi_{\text{OT}}^{1:n}$.
- (5) \mathcal{P} sends ℓ to $\Pi_{\text{OT}}^{1:n}$.
- (6) $\Pi_{\text{OT}}^{1:n}$ outputs $\{Y_\ell^{(1)}, \dots, Y_\ell^{(\tau)}\}$ to \mathcal{P} .
- (7) For every $k \in \{1, \dots, \tau\}$, \mathcal{P} :
 - (a) Computes $\mathbf{X}^{(k)} = \{X_1^{(k)}, \dots, X_m^{(k)}\}$ as a random additive share of x , i.e. $(x = \bigoplus_{j=1}^m X_j^{(k)})$
 - (b) Computes $\{view_{1,k}, \dots, view_{m,k}\} \leftarrow \Pi_{\text{MPCitH}}(\Pi_{\mathcal{F}}, (\mathbf{X}^{(k)}, Y_\ell^{(k)}))$.
- (8) For all $j \in \{1, \dots, m\}, k \in \{1, \dots, \tau\}$, \mathcal{P} sends $C_{view_{j,k}} \leftarrow \text{Com}(view_{j,k})$ to \mathcal{V}
- (9) \mathcal{V} de-commits s , which \mathcal{P} uses to derive $\{\varepsilon_1, \dots, \varepsilon_\tau\}$ and to reconstruct the 3D table as in above steps 2 - 3^b.
- (10) \mathcal{P} verifies the following properties hold for all $i, i' \in \{1, \dots, n\}, k \in \{1, \dots, \tau\}, j \in \{1, \dots, m\} \setminus \varepsilon_k$:

$$\bigoplus_{j=1}^m Y_{i,j}^{(k)} = y_i. \qquad Y_{i,j}^{(k)} = Y_{i',j}^{(k)}$$

If these properties do not hold, \mathcal{P} aborts.

\mathcal{P} decommits each $\{view_{j,k}\}_{k \in \{1, \dots, \tau\}, j \in \{1, \dots, m\} \setminus \varepsilon_k}$

- (11) \mathcal{V} checks that the decommitted views are consistent with honest executions of $\Pi_{\mathcal{F}}$, include an output of 1, and that $view_{j,k}$ has a Y input equal to $Y_{1,j}^{(k)}$.

^aThe input length $|x|$ can be either fixed or arbitrary.

^bAll messages sent by the receiver during $\Pi_{\text{OT}}^{1:n}$ are computed deterministically from the seed s and the common input $\{y_1, \dots, y_n\}$.

Figure 7: OR proof using MPC-in-the-Head

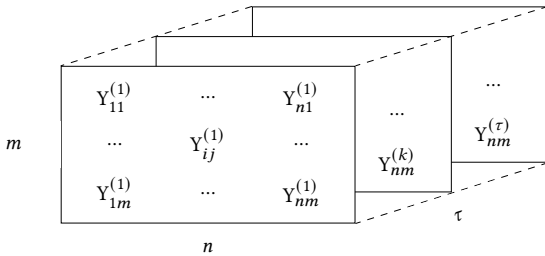


Figure 8: Notation for protocol $\Pi_f^{\text{MPCitH-OR}}$.

Consider the view of \mathcal{V} after step 9. The messages \mathcal{V} has received are encrypted queries of \mathcal{P} 's private input ℓ (step 1 of $\Pi_{\text{OT}}^{1:n}$), a transcript of a zero-knowledge proof (step 2 of $\Pi_{\text{OT}}^{1:n}$), and commitments to MPCitH views (step 8 of $\Pi_f^{\text{MPCitH-OR}}$).

Suppose \mathcal{V} does not follow the protocol honestly. In the honest protocol, \mathcal{V} 's messages are generated deterministically based on s , the common input, and \mathcal{P} 's messages. Hence after \mathcal{V} opens s in step 9, the prover can compare each message that \mathcal{V} sent against the expected message in the honest verifier protocol. By our assumption one of these messages is inconsistent, hence \mathcal{P} 's next action is to abort the protocol.

We construct a simulator \mathcal{S} for \mathcal{V} 's view after step 9 (shown in figure 9) for the case where \mathcal{V} deviates from the honest protocol and argue that the view when interacting with the simulator is indistinguishable from \mathcal{V} 's view when running

the security of the protocol in the presence of a semi-honest verifier.

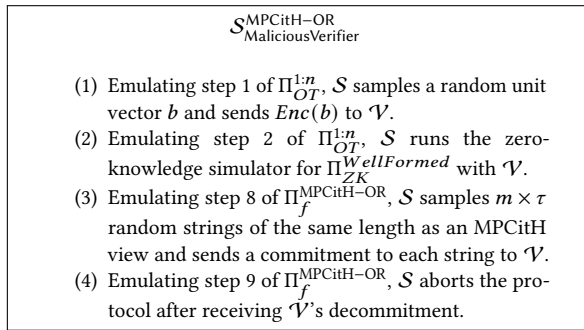


Figure 9: Malicious-verifier simulator for $\Pi_f^{\text{MPCitH-OR}}$

$\Pi_f^{\text{MPCitH-OR}}$. The encrypted query's indistinguishability follows from the semantic security of the encryption scheme. The zero-knowledge transcript's indistinguishability follows from the zero-knowledge property of $\Pi_{\text{ZK}}^{\text{WellFormed}}$. The indistinguishability of the committed views follows from the hiding property of Com. Finally, as discussed above a verifier that deviates from the honest protocol will always cause the protocol to abort after step 9.

Next, we claim that, when given access to an ideal functionality for $\mathcal{F}_{\text{OT}}^{1:n}$, our protocol is zero-knowledge against a verifier that does not deviate from the semi-honest protocol. Since $\Pi_{\text{OT}}^{1:n}$ implements $\mathcal{F}_{\text{OT}}^{1:n}$ with semi-honest sender security and we are assuming the verifier behaves semi-honestly, we may freely replace $\Pi_{\text{OT}}^{1:n}$ in our protocol with $\mathcal{F}_{\text{OT}}^{1:n}$. The simulator for $\Pi_f^{\text{MPCitH-OR}}$ then proceeds as follows.

- (1) Accept the verifier's inputs to the OT functionality, recovering all input encodings, $\{Y_i^{(1)}, \dots, Y_i^{(\tau)}\}$ for all i , as well as $(\epsilon_1, \dots, \epsilon_\tau)$. If any of these values are badly formed, the simulation sets an abort flag: $\text{abort} = 1$. Otherwise, for each $k \in \{1, \dots, \tau\}$, the simulator chooses random input shares, $\{X_1^{(k)}, \dots, X_m^{(k)}\}$, then discards $X_{\epsilon_k}^{(k)}$ and $Y_{i, \epsilon_k}^{(k)}$, for arbitrary i . (Note that, excluding $Y_{i, \epsilon_k}^{(k)}$ and $Y_{j, \epsilon_k}^{(k)}$, the remaining $n - 1$ shares of y_i and y_j are identical).
- (2) Let \mathcal{S} be the $(n - 1)$ -privacy simulator defined in Definition 2. The zero-knowledge simulator runs \mathcal{S} using the remaining $n - 1$ shares of x , and y_j . It commits to the resulting $n - 1$ views and sends them to the verifier.
- (3) The simulator runs steps 9-11 honestly: it accepts the decommitment to the verifier's randomness, performs the described correctness checks, and decommits to the simulated views if everything passes, and $\text{abort} \neq 1$.

By definition, the output of \mathcal{S} is indistinguishable from the decommitted views of the prover in the hybrid-world proof. The reader can verify that the remainder of the simulation is perfect.

Soundness: Suppose the Prover does not know a valid input x_i for any y_i . By the hiding property of Com, the prover learns nothing about the verifier's state other than the queried row from $\Pi_{\text{OT}}^{1:n}$ before it sends the commitments to its MPCitH views. And by the binding property of Com the prover's

MPCitH views must be fixed before the prover receives the rest of the verifier's state. Since the proof is accepted or rejected based on these views, we can reduce the soundness of the protocol to the security properties of the MPCitH protocol.

We consider 3 cases, depending on how many simulated MPCitH parties perform malicious behavior (as measured by comparing inconsistencies in the parties' views).

In the first case, all MPCitH parties act honestly. By the completeness of $\Pi_{\mathcal{F}}$, either all parties output 0 with all but negligible probability (in which case the proof is rejected) or the input (x_i, y_i) is a valid witness for f . By our previous assumption, if $f(x_i, y_i) = 1$, then y_i is not a member of the common input set, hence the input encoding $Y_{i,j}$ must be different from the encodings created by the verifier in at least one position. If any opened view's input encoding of y_i does not match the verifier's encodings the proof will be rejected, thus in order to create valid proof, \mathcal{P} must modify only the share of y_i that is not opened by \mathcal{V} . By the hiding property of Com, \mathcal{P} has no information about ϵ_j , hence the prover has at best $\frac{1}{m}$ chance of guessing correctly for a single iteration. Amplified over τ iterations the cheating prover's probability of success is $\leq \frac{1}{m^\tau}$.

The second case we consider is where exactly one simulated party performs malicious behavior. In this case, the malicious party's view must be inconsistent with at least one other view. Since $(n - 1)$ views are opened, the verifier will see this inconsistency unless the malicious view, or the receiver of the malicious message, is the one left unopened. Therefore, the cheating prover's probability of correctly guessing ϵ_j for every iteration is $(\frac{2}{m})^\tau$. As an aside, it should be straightforward to see that some smaller success probability is achieved if the prover mixes-and-matches the strategies of the first and second cases between iterations.

The final case is that there are two or more malicious parties. In this case it is guaranteed that there will be an inconsistent pair of views in any $(n - 1)$ -subset, giving the cheating prover a success probability of 0 in this case. \square

5 DISJUNCTIVE PROOFS FOR MIXED STATEMENTS

5.1 Disjunctive Proof for Mixed Statements

In the proof of assets problem, the Prover and Verifier hold as common input a list of hashed public keys, $L = \{y_1, \dots, y_n\}$, where $y_i = H(x_i)$, for some hash function H . The Prover wishes to prove that it knows secret key z such that (x, z) is a legitimate output of some cryptographic key generation algorithm, and, for some $y_i \in L$, $H(x) = y_i$.

More generally, we consider mixed statements of the form $f(x, y_i) = 1 \wedge g(x, z) = 1$, where $y_i \in L$, f is a Boolean (or "non-algebraic") function – in our application, a hash function – and $g(x, z)$ is an algebraic function – in our application, one verifying that z is the secret key corresponding to x .

Chase et al. [18] consider this question without the disjunction. That is, they assume $|L| = 1$, and focus solely on the challenge of constructing an efficient proof for mixed statements. They do this by leveraging the specific proof system

for f , built from Garbled Circuits, in the following way. The prover begins by committing to input x with $\text{Com}(x)$. The Verifier then prepares a garbled circuit for the Boolean circuit that outputs both $f(x)$, as well as a one-time MAC on the Prover’s input: $t = ax + b$. The prover is allowed to decode t , and commits to this as well. Finally, the prover provides a proof, using an algebraic proof system, that it knows (x, t) such that $f(x, y) = 1$, x is consistent with $\text{Com}(x)$, t is consistent with $\text{Com}(t)$, and $t = ax + b$. In this way, the MAC on x that was derived while proving $f(x, y) = 1$ ensures that the same input x is used when proving that $g(x, z) = 1$.

Note that Chase et al. compute $t = ax + b$ inside a Boolean circuit, which requires several thousand Boolean gates. We improve on their solution by using the oblivious transfer to avoid performing integer multiplication and addition in a Boolean circuit. As in the previous Section, we use MPC-in-the-head, rather than garbled circuits. Specifically, let x_i be the i th bit of x , and for each $u \in \{1, \dots, |x|\}$, let $\mathbf{0}_u$ and $\mathbf{1}_u$ denote the input encodings generated by the verifier for that input bit. The verifier chooses a at random, and samples b by choosing random b_u for each $u \in \{1, \dots, |x|\}$ and setting $b = \sum_u b_u$. When obliviously sending the encoding of the i th input bit, the verifier sends $(\mathbf{0}_u, b_u)$ and $(\mathbf{1}_u, 2^{u-1}a + b_u)$ to the OT functionality. By summing the 2nd value received in each of the $|x|$ received ordered pairs, the Prover recovers $ax + b$, and no computation in the circuit is required. We note that this leads to significant improvement over Chase et al. even when $|L| = 1$. In Figure 10, we present the full protocol, highlighting the changes in our disjunctive proof from Figure 7, in order to support mixed statements.

THEOREM 4. *Let $m \geq 3$, let $f(Y, X)$ be the function defined in Figure 10, let Com be a binding and hiding commitment scheme, $\text{MAC}(x)$ an unforgeable one-time MAC and let $\Pi_{\mathcal{F}}$ implement f with correctness and $(m-1)$ -privacy. Then the protocol described in Figure 10 is a zero-knowledge proof protocol for the language $\{(y_1, \dots, y_n), z, x) : \exists \ell \in \{1, \dots, n\} \text{ such that } f(x, y_\ell) = 1 \wedge g(x, z) = 1\}$ in the $(\text{Com}, \mathcal{F}_{\text{CO}T}^{1:n})$ -hybrid model.*

PROOF (SKETCH). *Zero Knowledge:* The simulator runs in the same fashion as in Theorem 3, and it inherits the same procedure for the ZK proof π .

Soundness: Our protocol inherits the soundness properties as in Theorem 3. Here the prover can also cheat by using an inconsistent witness x for functions f and g . However this is prevented from the unforgeability property of the one-time MAC, the binding property of the commitment scheme and the soundness property of the ZK proof π . \square

5.2 Proving the Value of Assets

When proving ownership of assets in a cryptocurrency such as Bitcoin, the exchange (i.e. the prover) needs to prove knowledge of a number of secret keys for the respective hashed public keys among those in the UTXO set. Additionally, they might wish to prove something about the values assigned to such keys, e.g. that their total value exceeds some minimum. For simplicity, we treat the UTXO set as a list of tuples, where

each tuple $(H_i || v_i)$ represents a hashed[†] public key and value pair, where $H_i = H(\text{pk}_i)$. Therefore, given a common input of a tuple list $L = \{(H_1 || v_1), (H_2 || v_2), \dots, (H_n || v_n)\}$, \mathcal{P} must prove knowledge of secret keys $\{\text{sk}_k\}_{k=1}^t$ corresponding to a set of public keys $S = \{\text{pk}_k\}_{k=1}^t$ such that $\forall k \in \{1, \dots, t\}$, $(\text{sk}_k, \text{pk}_k)$ is a valid output of the appropriate key generation algorithm, $(H(\text{pk}_k), v_k) \in L$, and, $\sum_{k=1}^t v_i \geq v$.

The protocol as discussed above, is not sufficient for the Proof of Assets application as it does not connect the provers’ keys to their corresponding “coin” value stored on the blockchain. In Figure 11 of the Appendix, we present an extension of our protocol that also supports values. The main change is to provide an additional MAC on the input values, in order to bind them with their respective hashed keys. Also for simplicity, we do not provide a k -out-of- n OR proof but rather a 1-out-of- n OR proof (i.e. we only assume that the exchange only controls one address in the UTXO set), however the protocol can be naturally extended to accommodate multiple keys.

6 IMPLEMENTATION

gOTzilla implementation is based on SealPIR [2] and MP-SPDZ [1][‡] libraries. As discussed in Section 3, we use SealPIR for our needed OT functionality, and we provide more implementation details below.

6.1 Evaluation

Our first set of benchmarks for protocol $\Pi_f^{\text{MPCitH-OR}}$ is performed locally, with the prover and verifier running on the same host. We run our benchmarks on a z1d.metal AWS instance using 48 threads (24 physical cores) and 384 MB of RAM. We performed our evaluations for a range of disjunctive elements between 2^{16} and 2^{24} . As we require $m^\tau \simeq 2^{40}$ soundness, we pick $m = 3$ MPCitH (minimum required) parties and $\tau = 25$ repetitions as this choice of parameters minimizes $(m-1) \cdot \tau$. Concretely, if we consider Limbo [22] as an efficient underlying MPCitH protocol, this implies about 50ms additional runtime cost on top of the rest of our protocol’s runtime, which does not depend on n , and is dwarfed by the overall runtime costs of our protocol (therefore we do not take it into account). Assuming $|f(x)| = 256$ bits, the size of each slice of the 3D table is $256 \cdot m \cdot (\tau - 1)$ (we only need to send $\tau - 1$ shares as the remaining one can be inferred) which implies a 1600 byte size per element. As shown in Table 3, for $n = 2^{20}$ the total communication between the prover and verifier is 6.18 MB, plus 3.45 MB for communicating the PIR Galois keys beforehand. However, both the prover and verifier need to generate a version of the 3D table in their local memory (or storage) based on the seed s (steps 9 and 3 of the protocol $\Pi_f^{\text{MPCitH-OR}}$ respectively). In our current implementation, we

[†]We also treat the double hashing RIPEMD160(SHA256(0)) of public keys in Bitcoin as a single hash function H .

[‡]MP-SPDZ provides the ZK proof of plaintext knowledge needed for $\Pi_{\text{ZK}}^{\text{WellFormed}}$. The proof implemented in this library is for the BGV cryptosystem [15], whereas to be compatible with SealPIR we need a proof in the BFV cryptosystem [23]. However, a BFV-compatible version of this proof has been designed [19], and claims theoretically cheaper cost than the BGV version [9], but currently has no publicly available implementation. For this reason we believe the existing BGV implementation provides a good estimate of the cost.

Setup. Let $f(x, y)$ be a Boolean function and $g(x, z)$ an algebraic function. Let \mathcal{F} be an m -party functionality that takes input (X_j, Y_j) from each party P_j and outputs $f(\oplus_{j=1}^m X_j, \oplus_{j=1}^m Y_j) \stackrel{?}{=} 1$ to all parties. Let $\Pi_{\mathcal{F}}$ be an m -party protocol that securely realizes \mathcal{F} with correctness and $(m-1)$ -privacy.

Common inputs. τ total number of repetitions, n values $\{y_1, \dots, y_n\} \in \{0, 1\}^\kappa$, and m , the number of parties in the MPC protocol run in the head of the Prover. m and τ are set such that $m^{-\tau} < 2^{-\lambda}$, where λ is a security parameter.

Prover's input. x, z, ℓ such that: $\ell \in \{1, \dots, n\}$, $f(x, y_\ell) = 1$ and $g(x, z) = 1$. We denote as $(x_1, \dots, x_{|x|})$ the bit representation of x (i.e. $x = \sum_{u=1}^{|x|} 2^{u-1} x_u$).

Verifier's input. $C_x = \text{Com}(x)$, $C_z = \text{Com}(z)$.

(1) Verifier \mathcal{V} generates its random tape s and sends $C_s \leftarrow \text{Com}(s)$ to the prover \mathcal{P} . Throughout the rest of the protocol, all randomness of the Verifier is generated by applying a PRG, $G(s)$. (We will, imprecisely, refer to these as "random" values.)

(2) \mathcal{V} uses the random seed s to sample the following values uniformly at random:

(a) $\varepsilon_k \xleftarrow{\$} \{1, \dots, m\}$ for $k \in \{1, \dots, \tau\}$.

(b) $r_k \xleftarrow{\$} \{0, 1\}^\kappa$ for $k \in \{1, \dots, \tau\}$.

(c) $w_{u,k} \xleftarrow{\$} \{0, 1\}^\kappa$ for $u \in \{1, \dots, |x|\}$, $k \in \{1, \dots, \tau\}$

(d) $a \xleftarrow{\$} \{0, 1\}^\lambda$, $b_u \xleftarrow{\$} \{0, 1\}^{|x|+\lambda}$ for $u \in \{1, \dots, |x|\}$. Define $b := \sum_{u=1}^{|x|} b_u$

(3) For $u \in \{1, \dots, |x|\}$, $k \in \{1, \dots, \tau\}$, \mathcal{V} computes $\{0_{u,1}^{(k)} \dots 0_{u,m}^{(k)}\} \leftarrow \text{ShareAt}(0, \varepsilon_k, w_{u,k})$ and $\{1_{u,1}^{(k)} \dots 1_{u,m}^{(k)}\} \leftarrow \text{ShareAt}(1, \varepsilon_k, w_{u,k})$

(4) For $i \in \{1, \dots, n\}$, $k \in \{1, \dots, \tau\}$, \mathcal{V} computes vector $Y_i^{(k)} := \{Y_{i,1}^{(k)}, \dots, Y_{i,m}^{(k)}\} \leftarrow \text{ShareAt}(y_i, \varepsilon_k, r_k)$

(5) **Exchange labels for inputs.**

For $i \in \{1, \dots, n\}$ denote 2D table $Y_i := \{Y_i^{(1)}, \dots, Y_i^{(\tau)}\}$, and for $u \in \{1, \dots, |x|\}$ denote 2D tables $\mathbf{0}_u := \{\{0_{u,1}^{(1)}, \dots, 0_{u,m}^{(1)}\}, \dots, \{0_{u,1}^{(\tau)}, \dots, 0_{u,m}^{(\tau)}\}\}$, $\mathbf{1}_u := \{\{1_{u,1}^{(1)}, \dots, 1_{u,m}^{(1)}\}, \dots, \{1_{u,1}^{(\tau)}, \dots, 1_{u,m}^{(\tau)}\}\}$

(a) \mathcal{V} sends $\{Y_1, \dots, Y_n\}$ to $\Pi_{\text{OT}}^{1:n}$.

(b) \mathcal{P} sends ℓ to $\Pi_{\text{OT}}^{1:n}$.

(c) $\Pi_{\text{OT}}^{1:n}$ outputs Y_ℓ to \mathcal{P} .

(d) For every $u \in \{1, \dots, |x|\}$

(i) \mathcal{V} sends $\{(\mathbf{0}_u, b_u), (\mathbf{1}_u, 2^{u-1}a + b_u)\}$ to $\Pi_{\text{OT}}^{1:2}$.

(ii) For every $u \in \{1, \dots, |x|\}$, \mathcal{P} sends x_u to $\Pi_{\text{OT}}^{1:2}$.

(iii) If $x_u = 0$ then $\Pi_{\text{OT}}^{1:2}$ outputs $(\mathbf{0}_u, b_u)$ to \mathcal{P} , otherwise it outputs $(\mathbf{1}_u, 2^{u-1}a + b_u)$. \mathcal{P} denotes whichever output it receives as $\{\{X_{u,1}^{(1)}, \dots, X_{u,m}^{(1)}\}, \dots, \{X_{u,1}^{(\tau)}, \dots, X_{u,m}^{(\tau)}\}, M_u\}$

(e) For $k \in \{1, \dots, \tau\}$ denote the 2D table $X^{(k)} := \{(X_{1,1}^{(k)} || \dots || X_{|x|,1}^{(k)}), \dots, (X_{1,m}^{(k)} || \dots || X_{|x|,m}^{(k)})\}$

(6) For every $k \in \{1, \dots, \tau\}$, \mathcal{P} computes $(\text{view}_{1,k}, \dots, \text{view}_{m,k}) \leftarrow \Pi_{\text{MPCitH}}(\Pi_{\mathcal{F}}, (X^{(k)}, Y_\ell^{(k)}))$.

(7) \mathcal{P} computes $\text{MAC}(x) = \sum_{u=1}^{|x|} (M_u) = a \cdot x + b$ and $C_{\text{MAC}(x)} \leftarrow \text{Com}(\text{MAC}(x))$.

(8) For all $j \in \{1, \dots, m\}$, $k \in \{1, \dots, \tau\}$, \mathcal{P} sends $C_{\text{MAC}(x)}$ and $C_{\text{view}_{j,k}} \leftarrow \text{Com}(\text{view}_{j,k})$ to \mathcal{V} .

(9) \mathcal{V} decommits s , which \mathcal{P} uses to reconstruct $\{\varepsilon_1, \dots, \varepsilon_\tau\}$, $\{\mathbf{0}_u, \mathbf{1}_u\}$, $\{Y_1, \dots, Y_n\}$

(10) \mathcal{P} verifies the following properties hold for all $i, i' \in \{1, \dots, n\}$, $u \in \{1, \dots, |x|\}$, $k \in \{1, \dots, \tau\}$, $j \in \{1, \dots, m\} \setminus \varepsilon_k$:

$$\bigoplus_{j=1}^m Y_{i,j}^{(k)} = y_i, \quad Y_{i,j}^{(k)} = Y_{i',j}^{(k)} \quad \bigoplus_{j=1}^m \mathbf{0}_{u,j}^{(k)} = 0 \quad \bigoplus_{j=1}^m \mathbf{1}_{u,j}^{(k)} = 1 \quad \mathbf{0}_{u,j}^{(k)} = \mathbf{1}_{u,j}^{(k)}$$

(11) \mathcal{P} decommits each $\{\text{view}_{j,k}\}_{k \in \{1, \dots, \tau\}, j \in \{1, \dots, m\} \setminus \varepsilon_k}$

(12) \mathcal{V} checks that the decommitted views are consistent with honest executions of $\Pi_{\mathcal{F}}$, and, if so, outputs 1.

(13) \mathcal{P} and \mathcal{V} execute the following ZK proof protocol: $\pi = \{(x, \text{MAC}(x), z) : C_x = \text{Com}(x) \wedge C_{\text{MAC}(x)} = \text{Com}(\text{MAC}(x)) \wedge \text{MAC}(x) = ax + b \wedge C_z = \text{Com}(z) \wedge g(x, z) = 1\}(\text{Com}(x), \text{Com}(z), \text{Com}(\text{MAC}(x)), a, b)$

(14) If π verifies, \mathcal{V} accepts, else \mathcal{V} rejects.

Figure 10: Disjunctive protocol via MPCitH for mixed statements. We denote by colored text the additional elements introduced compared to Fig.7

Table 2: Evaluation for protocol in Fig 10. PIR preprocessing: NTT transform. Note: a) MPCitH encoding is needed by both the \mathcal{P} and \mathcal{V} , therefore this column is counted twice in the total runtime b) PIR preprocessing is executed by both \mathcal{P} and \mathcal{V} in parallel.

Number of elements	MPCitH encoding (both \mathcal{P} and \mathcal{V})	PIR Preprocessing (\mathcal{P} and \mathcal{V})	PIR Query (\mathcal{P})	PIR Reply (\mathcal{V})	PIR Decode (\mathcal{P})	$\Pi_{ZK}^{WellFormed}$ (polynomial interpolation) (\mathcal{P})	$\Pi_{ZK}^{WellFormed}$ (bounded noise) (\mathcal{P})	Total Runtime	Run-time of [10]
2^{16}	31ms	392ms	<1ms	880ms	2ms	1ms	151ms	1.64s	31.87s
2^{18}	86ms	1.3s	2ms	2.2s	3ms	1ms	152ms	3.83s	37.5s
2^{20}	358ms	6.47s	2ms	7.07s	5ms	1ms	163ms	14.89s	60s
2^{22}	1.61s	9.7s	3ms	9.6s	10ms	2ms	165ms	22.7s	150s
2^{24}	7.4s	38.59s	4ms	32.6s	10ms	3ms	167ms	86.09s	510s

store the entire table in RAM (requiring about 18GB memory for 2^{20} such elements), however this table can be offloaded to disk and retrieved as needed at the cost of additional I/O operations, or alternatively, each slice of the cube can be generated on demand as needed.

Table 2 shows our local benchmarks in detail, where we provide a break down of the protocol’s total runtime as follows:

- (1) Verifier’s secret share phase (step 3 for ShareAt() of Fig. 7).
- (2) The 1-out-of- n OT phase (steps 4 - 6 of Fig. 7) which include:
 - Preprocessing, Query, Reply and Decode costs of SealPIR.
 - Polynomial interpolation (step. 3 of Fig. 3).
 - Proof of bounded noise (step. 1 of Fig. 3).

Note for brevity, we don’t include the costs of PRG generation, commitment and de-commitment costs, one-time MAC and ZK proof protocol as these are negligible compared to the overall runtime costs (which as shown in Table 2, are dominated by MPCitH encoding and PIR preprocessing and reply).

We also performed a second set of benchmarks over a network for $n = 2^{20}$, as shown on Table 4, where we take the additional network latency into account as well. In particular, the PIR Query - Reply would replace the 5th and 6th column together on Table 2, and similarly the rest of our network measurements. This shows that although our protocol has many rounds of interaction (most of them during $\Pi_{ZK}^{WellFormed}$), the overall impact to its total run-time is very small.

Comparison. We now make a concrete comparison of our protocol’s runtime and communication costs with Mac’n’cheese [10], which also aims for disjunctive Zero-knowledge proofs and has similar asymptotic costs as shown in Fig. 1. We observe that Mac’n’cheese is not explicitly tailored for disjunctive proofs comprised of circuits with the same structure, however as discussed in Section 1, in such a case its disjunctive statement $C(x) = y_1 \vee C(x) = y_2 \vee \dots \vee C(x) = y_n$ can be modified as $(C(x) = y) \wedge (y = y_1 \vee \dots \vee y = y_n)$ to avoid many circuit evaluations, resulting in a conjunctive statement which one part consists of a disjunctive statement of equality checks.

In addition, as discussed in Section 5, our protocol can be naturally extended to accommodate mixed statements, with only an overhead of $|x|$ 1-out-of-2 OTs, which only cost roughly

Table 3: Communication costs for Fig. 7 protocol (including Fig. 3 subroutine) for $n = 2^{20}$

PIR Query (Step 5)	64.14 KB
PIR Response (Step 4)	320.71 KB
$\Pi_{ZK}^{WellFormed}$ poly interp.	192.04 KB
$\Pi_{ZK}^{WellFormed}$ bounded noise	5.62 MB
Committed views (Step 8)	2.45 KB
Total	6.18 MB

25ms in total for $|x| = 256$ [4], a negligible cost compared to the main protocol’s benchmarks presented above.

Given those observations, we compare with Mac’n’cheese’s needed runtime for n equality checks plus proving $(C(x) = y)$ where C is a 250 million gate Boolean circuit, which is equivalent to converting from an arithmetic circuit representing the algebraic statement. Specifically when $n = 2^{20}$, the estimated reported cost of C for Mac’n’Cheese is 30 seconds, and the cost for 2^{20} equality checks (256 million gates), is another 30 seconds, running on a system with equivalent specifications[§], adding to a total runtime of 60 seconds, with a total communication cost of 63 MB. Given our measurements, we observe significant improvements even when comparing with the disjunctive equality checks part of Mac’n’cheese. In addition, Mac’n’cheese can handle only Boolean or arithmetic circuits, therefore as an example, a mixed statement in the form of $\text{SHA256}(g^x) = y$ would need around 250 million gates (as shown in Appendix C), while gOTzilla is compatible with techniques combining algebraic and non-algebraic statements similar to the work by Chase et al. [18] as discussed in Section 5, and therefore we don’t need to covert between circuit types.

7 CONCLUSION

We presented gOTzilla, a novel protocol for disjunctive Zero-Knowledge proofs, tailored for large disjunctions. While our protocol has equivalent asymptotic communication costs with

[§]These estimates were provided by Mac’n’cheese authors assuming cost per gate is 120ns.

Table 4: Latency costs for Fig. 7 protocol (including Fig. 3 subroutine) for $n = 2^{20}$

	PIR Query + Reply	$\Pi_{ZK}^{WellFormed}$ polynomial interpol.	$\Pi_{ZK}^{WellFormed}$ bounded noise	MPCitH views	Total
US East - East	7.405s	7ms	298ms	370ms	8.08s
US East - West	7.563s	126ms	892ms	381ms	8.962s
US East - Japan	7.738s	292ms	1.59s	421ms	10.04s

recent works, we show that gOTzilla offers a concrete improvement over the state-of-the-art, especially when the disjunctions include mixed (i.e. algebraic and non-algebraic) statements, since our protocol is more “mixed statement-friendly”. Finally, as the Bitcoin’s UTXO count is roughly 80 million at the time of writing [3], gOTzilla can serve as a basis for a Proof of Assets over Bitcoin’s blockchain, where an exchange can interactively prove its assets to an auditor in a few minutes.

REFERENCES

[1] [n. d.]. Multi-Protocol SPDZ. <https://github.com/data61/MP-SPDZ>

[2] [n. d.]. SealPIR: A computational PIR library that achieves low communication costs and high performance. <https://github.com/microsoft/SealPIR>

[3] 2022. Blockchain.com Unspent Transaction Outputs. <https://www.blockchain.com/charts/utxo-count>

[4] 2022. Efficient Multi-party computation toolkit. <https://github.com/emp-toolkit/emp-of#iknp-style-protocols>

[5] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. 2018. Non-Interactive Zero-Knowledge Proofs for Composite Statements. In *CRYPTO 2018, Part III (LNCS, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 643–673. https://doi.org/10.1007/978-3-319-96878-0_22

[6] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. 2017. Liger: Lightweight Sublinear Arguments Without a Trusted Setup. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2087–2104. <https://doi.org/10.1145/3133956.3134104>

[7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 962–979. <https://doi.org/10.1109/SP.2018.00062>

[8] Michael Backes, Lucjan Hanzlik, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. 2019. Efficient Non-Interactive Zero-Knowledge Proofs in Cross-Domains Without Trusted Setup. In *PKC 2019, Part I (LNCS, Vol. 11442)*, Dongdai Lin and Kazuo Sako (Eds.). Springer, Heidelberg, 286–313. https://doi.org/10.1007/978-3-030-17253-4_10

[9] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. 2019. Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ. In *SAC 2019 (LNCS, Vol. 11959)*, Kenneth G. Paterson and Douglas Stebila (Eds.). Springer, Heidelberg, 274–302. https://doi.org/10.1007/978-3-030-38471-5_12

[10] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. 2021. Mac’n’Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, 92–122. https://doi.org/10.1007/978-3-030-84259-8_4

[11] Mihir Bellare and Shafi Goldwasser. 1990. New Paradigms for Digital Signatures and Message Authentication Based on Non-Interactive Zero Knowledge Proofs. In *CRYPTO’89 (LNCS, Vol. 435)*, Gilles Brassard (Ed.). Springer, Heidelberg, 194–211. https://doi.org/10.1007/0-387-34805-0_19

[12] Eli Ben-Sasson, Iddo Bentov, Yitong Horeh, and Michael Riabzev. 2019. Scalable Zero Knowledge with No Trusted Setup. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, 701–732. https://doi.org/10.1007/978-3-030-26954-8_23

[13] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security*

and Privacy. IEEE Computer Society Press, 459–474. <https://doi.org/10.1109/SP.2014.36>

[14] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Heidelberg, 868–886. https://doi.org/10.1007/978-3-642-32009-5_50

[15] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 309–325. <https://doi.org/10.1145/2090236.2090262>

[16] Jan Camenisch and Anna Lysyanskaya. 2001. An Identity Escrow Scheme with Appointed Verifiers. In *CRYPTO 2001 (LNCS, Vol. 2139)*, Joe Kilian (Ed.). Springer, Heidelberg, 388–407. https://doi.org/10.1007/3-540-44647-8_23

[17] Jan Camenisch and Markus Stadler. 1997. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *CRYPTO’97 (LNCS, Vol. 1294)*, Burton S. Kaliski Jr. (Ed.). Springer, Heidelberg, 410–424. <https://doi.org/10.1007/BFb0052252>

[18] Melissa Chase, Chaya Ganesh, and Payman Mohassel. 2016. Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials. In *CRYPTO 2016, Part III (LNCS, Vol. 9816)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 499–530. https://doi.org/10.1007/978-3-662-53015-3_18

[19] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. 2020. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. Cryptology ePrint Archive, Report 2020/451. <https://ia.cr/2020/451>

[20] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO’94 (LNCS, Vol. 839)*, Yvo Desmedt (Ed.). Springer, Heidelberg, 174–187. https://doi.org/10.1007/3-540-48658-5_19

[21] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. 2015. Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 720–731. <https://doi.org/10.1145/2810103.2813674>

[22] Cyprien Delpach de Saint Guilhem, Emmanuelle Orsini, and Titouan Tanguy. 2021. Limbo: Efficient Zero-knowledge MPCitH-based Arguments. In *CCS ’21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 3022–3036. <https://doi.org/10.1145/3460120.3484595>

[23] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. <https://eprint.iacr.org/2012/144>

[24] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *USENIX Security 2016*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 1069–1083.

[25] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kapthuk. 2021. Stacking Sigmas: A Framework to Compose Σ -Protocols for Disjunctions. Cryptology ePrint Archive, Report 2021/422. <https://ia.cr/2021/422>

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design. In *CRYPTO’86 (LNCS, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, 171–185. https://doi.org/10.1007/3-540-47721-7_11

[27] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *17th ACM STOC*. ACM Press, 291–304. <https://doi.org/10.1145/22145.22178>

[28] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT 2016, Part II (LNCS, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Heidelberg, 305–326. https://doi.org/10.1007/978-3-662-49896-5_11

[29] Louis C. Guillou and Jean-Jacques Quisquater. 1988. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In *EUROCRYPT’88 (LNCS, Vol. 330)*, C. G. Günther (Ed.). Springer, Heidelberg, 123–128. https://doi.org/10.1007/3-540-45961-8_11

[30] David Heath and Vladimir Kolesnikov. 2020. Stacked Garbling for Disjunctive Zero-Knowledge Proofs. In *EUROCRYPT 2020, Part III (LNCS, Vol. 12107)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 569–598. https://doi.org/10.1007/978-3-030-45727-3_19

[31] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from secure multiparty computation. In *39th ACM STOC*, David S. Johnson and Uriel Feige (Eds.). ACM Press, 21–30. <https://doi.org/10.1145/1250790.1250794>

[32] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. 2013. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM CCS 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM Press, 955–966. <https://doi.org/10.1145/2508859.2516662>

- [33] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. 2018. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 525–537. <https://doi.org/10.1145/3243734.3243805>
- [34] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *38th FOCS*. IEEE Computer Society Press, 364–373. <https://doi.org/10.1109/SFCS.1997.646125>
- [35] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 238–252. <https://doi.org/10.1109/SP.2013.47>
- [36] Claus-Peter Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89 (LNCS, Vol. 435)*, Gilles Brassard (Ed.). Springer, Heidelberg, 239–252. https://doi.org/10.1007/0-387-34805-0_22
- [37] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 704–737. https://doi.org/10.1007/978-3-030-56877-1_25
- [38] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 1074–1091. <https://doi.org/10.1109/SP40001.2021.00056>

A DISJUNCTIVE PROOFS USING GARBLED CIRCUITS

A Garbled Circuit (GC) scheme is defined by the following algorithms [18]:

- $e, d, GC \leftarrow \text{Gb}(\text{pp}, \lambda, f, r)$ which on input of a boolean circuit f outputs a garbled circuit GC and encoding and decoding information e and d .
- $X \leftarrow \text{Enc}(e, x)$ which on input of encoding information e and an input x corresponding to f , outputs a garbled input X .
- $Y \leftarrow \text{Eval}(GC, X)$ which on input of garbled circuit GC and garbled input X outputs an encoded output Y .
- $y \leftarrow \text{Dec}(Y, d)$ which on input of an encoded output Y and decoding information d outputs y .
- $\{d, \emptyset\} \leftarrow \text{Ver}(GC, e, f)$ which on input of garbled circuit GC , encoding information e and boolean function f outputs either decoding information d or \emptyset .

DEFINITION 3. A GC scheme $(\text{Gb}, \text{Enc}, \text{Eval}, \text{Dec}, \text{Ver})$ must satisfy the following properties:

- **Correctness:** $\forall f, x, GC, e, d$: (a) For $Y \leftarrow \text{Eval}(GC, X)$, $f(x) \leftarrow \text{Dec}(Y, d)$ and (b) for $d \leftarrow \text{Ver}(GC, e, f)$ with $d \neq \emptyset$ and $Y \leftarrow \text{Eval}(GC, X)$, $\text{Dec}(Y, d) = f(x)$.
- **Authenticity:** $\forall f, x$ and PPT algorithm \mathcal{A} , there is a negligible function $\epsilon(\cdot)$ s.t. $\Pr[\exists y \neq f(x), y = \text{Dec}(d, d') : (e, d, GC) \leftarrow \text{Gb}(\text{pp}, \lambda, f, r), d' \leftarrow \mathcal{A}(GC, \text{Enc}(e, x))] \leq \epsilon(\lambda)$.
- **Privacy:** There exists a PPT simulator \mathcal{S} s.t. the distributions $(\text{Gb}, \text{Enc}, \text{Eval}, \text{Dec}, \text{Ver})$, $X \leftarrow \text{Enc}(e, x)$ and $\mathcal{S}(f, f(x))$ are indistinguishable.

As discussed in Section 5, Chase et al. [18] aims to provide ZK proof protocols for mixed statements in the form of $f(x, y) = 1 \wedge g(x, z) = 1$. Towards this goal, it constructs protocols in two different ways. The first (and more efficient) assumes the existence of a bit-wise commitment as part of a larger protocol, while the second requires a separate sub-circuit to compute a one-time MAC $t = a \cdot x + b$ which has $O(|x||a|)$ AND gates, where $|x|$ and $|a|$ is the bit length of

x and a respectively. For instance, if $|x| = |a| = 512$, then $|x||a| = 262144$, which is about 10 times the size of a SHA256 circuit.

We observe that this one-time MAC value can be computed during a COT step using a similar process to how we encode the input (shown in $\Pi_{\text{MAC},f}$ GC), where \mathcal{F}_{COT} is equivalent to \mathcal{F}_{OT} plus an opening phase to the receiver (refer to [18] for the ideal functionality). In this protocol, the prover and verifier compute $\text{MAC}(x) = a \cdot x + b$ as follows. The prover computes the bit decomposition of $x: x_1, \dots, x_{|x|}$. The verifier creates additive shares of $b: b_1, \dots, b_{|x|}$. For each bit x_u in x the two parties perform a 1-out-of-2 OT. If $x_u = 0$ the prover receives $M_u \leftarrow b_u$, otherwise the prover receives $M_u \leftarrow (2^{u-1}a + b_u)$. From $M_1, \dots, M_{|x|}$ the prover is able to compute $\text{MAC}(x)$ as $\sum_{u=1}^{|x|} M_u = \sum_{u=1}^{|x|} 2^{u-1}a \cdot x_u + b_u = a \cdot x + b = \text{MAC}(x)$.

However, even with the above optimization, the costs for a disjunctive proof remain linear in the size of the circuit.

In Figure 12 we show an optimized OR Proof using Garbled Circuits with MAC, while in Figure 13 we extend the previous protocol with value.

B OPTIMIZATION FOR PREPROCESSING

Let m be the number of parties used for MPCitH, and τ be the number of repetitions. For each input value y_i , the verifier generates the randomized shares of y_i as

$$(\text{ShareAt}(y_i, r_1), \dots, \text{ShareAt}(y_i, r_\tau))$$

Let ϵ_j be the position that will not be opened in the j^{th} run, then we have $\text{ShareAt}(y_i, r_j) = (r_{j,1}, \dots, r_{j,\epsilon_j-1}, r_{j,\epsilon_j} + y_i, r_{j,\epsilon_j+1}, \dots, r_{j,m})$ where $r_{j,1} + \dots + r_{j,m} = 0$. This can be rewritten as

$$\text{ShareAt}(y_i, r_j) = (r_{j,1}, \dots, r_{j,m}) + y_i \cdot s_j$$

where s_j is a vector of length m such that $s_{j,k} = 0$ for $k \neq \epsilon_j$ and $s_{j,\epsilon_j} = 1$. Abuse the notation a bit, we can write $\text{ShareAt}(y_i, r_j) = r_j + y_i \cdot s_j$.

Now, $(\text{ShareAt}(y_i, r_1), \dots, \text{ShareAt}(y_i, r_\tau)) = (r_1, \dots, r_\tau) + y_i \cdot (s_1, \dots, s_\tau) = r + y_i \cdot s$ where $(r + y_i \cdot s)$ is the i^{th} row of the database.

To compute the OT, the parties need to compute

$\sum_{i=1}^N C_i \cdot (r + y_{i+jN} \cdot s) = r \cdot (\sum_{i=1}^N C_i) + \sum_{i=1}^N C_i \cdot (y_{i+jN} \cdot s) = r \cdot (\sum_{i=1}^N C_i) + s \cdot (\sum_{i=1}^N C_i \cdot y_{i+jN})$ for $j \in [1; n/N]$, which is the dominant computation cost of the protocol and accounts for more than 95% the total run-time.

To compute this, we will first compute $\sum_{i=1}^N C_i$ and $\sum_{i=1}^N C_i \cdot y_{i+jN}$, followed by $r \cdot (\sum_{i=1}^N C_i)$ and $s \cdot (\sum_{i=1}^N C_i \cdot y_{i+jN})$.

Optimize the preprocessing. The preprocessing involves the computation of the number theoretic transform (NTT) of C_i and y_i . Note that due to the selected parameters, the plaintext modulus used in our Ring-LWE instance is only 12 bits. As y_i has 256 bits and does not fit into one coefficient, y_i is represented as a low degree polynomial (in \mathcal{R}_t). Specifically, we represent y_i as coefficients of a polynomial of degree 21, of which coefficients have 12 bits. Without any optimizations, we need to execute n NTT operations to compute $\text{NTT}(y_i)$. However, we can precompute the NTT for $u_j \in \mathcal{R}_t$ such that

Setup. $f(x, y)$ is a Boolean function and $g(x, z)$ is an algebraic function. $\Pi_{\mathcal{F}}$ is a semi-honest m -party protocol implementing the functionality \mathcal{F} which takes as input (X_j, Y_j) from each party P_j and outputs to all parties $f(\bigoplus_{j=1}^m X_j, \bigoplus_{j=1}^m Y_j) \stackrel{?}{=} 1$ with correctness and $(m-1)$ -privacy.

Common inputs. τ total number of repetitions, n public key and value pairs $(y_1, v_1), \dots, (y_n, v_n)$ and a minimum asset value v_0 . $y_i \in \{0, 1\}^\kappa$ and $m^{-\tau} < 2^{-\lambda}$, λ is a security parameter.

Prover's input. x, z, ℓ such that: $f(x, y_\ell) = 1$, $v_\ell \geq v_0$, and $g(x, z) = 1$. We denote as $(x_1, \dots, x_{|x|})$ the bit representation of x (i.e. $x = \sum_{u=1}^{|x|} 2^{u-1} x_u$).

Verifier's input. $C_x = \text{Com}(x)$, $C_z = \text{Com}(z)$, $C_v = \text{Com}(v_\ell)$.

- (1) Verifier \mathcal{V} generates its random tape s and sends $C_s \leftarrow \text{Com}(s)$ to the prover \mathcal{P} . Throughout the rest of the protocol, all randomness of the Verifier is generated by applying a PRG, $G(s)$. (We will, imprecisely, refer to these as "random" values.)
- (2) \mathcal{V} uses the random seed s to sample the following values uniformly at random:
 - (a) $\varepsilon_k \xleftarrow{\$} \{1, \dots, m\}$ for $k \in \{1, \dots, \tau\}$.
 - (b) $r_k \xleftarrow{\$} \{0, 1\}^\kappa$ for $k \in \{1, \dots, \tau\}$.
 - (c) $w_{u,k} \xleftarrow{\$} \{0, 1\}^\kappa$ for $u \in \{1, \dots, |x|\}$, $k \in \{1, \dots, \tau\}$
 - (d) $a \xleftarrow{\$} \{0, 1\}^\lambda$, $b_u \xleftarrow{\$} \{0, 1\}^{|x|+\lambda}$ for $u \in \{1, \dots, |x|\}$. Define $b := \sum_{u=1}^{|x|} b_u$
 - (e) $c \xleftarrow{\$} \{0, 1\}^\lambda$, $d \xleftarrow{\$} \{0, 1\}^{|v|+\lambda}$
- (3) For $u \in \{1, \dots, |x|\}$, $k \in \{1, \dots, \tau\}$, \mathcal{V} computes $\{0_{u,1}^{(k)} \dots 0_{u,m}^{(k)}\} \leftarrow \text{ShareAt}(0, \varepsilon_k, w_{u,k})$ and $\{1_{u,1}^{(k)} \dots 1_{u,m}^{(k)}\} \leftarrow \text{ShareAt}(1, \varepsilon_k, w_{u,k})$
- (4) For $i \in \{1, \dots, n\}$, $k \in \{1, \dots, \tau\}$, \mathcal{V} computes vector $\{Y_{i,1}^{(k)}, \dots, Y_{i,m}^{(k)}\} \leftarrow \text{ShareAt}(y_i, \varepsilon_k, r_k)$
- (5) **Exchange labels for inputs.**

For $i \in \{1, \dots, n\}$ denote 2D table $Y_i := \{Y_i^{(1)}, \dots, Y_i^{(\tau)}, cv_i + d\}$, and for $u \in \{1, \dots, |x|\}$ denote 2D tables $0_u := \{\{0_{u,1}^{(1)}, \dots, 0_{u,m}^{(1)}\}, \dots, \{0_{u,1}^{(\tau)}, \dots, 0_{u,m}^{(\tau)}\}\}$, $1_u := \{\{1_{u,1}^{(1)}, \dots, 1_{u,m}^{(1)}\}, \dots, \{1_{u,1}^{(\tau)}, \dots, 1_{u,m}^{(\tau)}\}\}$

 - (a) \mathcal{V} sends $\{(Y_1, c \cdot v_1 + d), \dots, (Y_n, c \cdot v_n + d)\}$ to $\Pi_{\text{OT}}^{1:n}$.
 - (b) \mathcal{P} sends ℓ to $\Pi_{\text{OT}}^{1:n}$.
 - (c) $\Pi_{\text{OT}}^{1:n}$ outputs $(Y_\ell, c \cdot v_\ell + d)$ to \mathcal{P} .
 - (d) For every $u \in \{1, \dots, |x|\}$
 - (i) \mathcal{V} sends $\{(0_u, b_u), (1_u, 2^{u-1}a + b_u)\}$ to $\Pi_{\text{OT}}^{1:2}$.
 - (ii) \mathcal{P} sends x_u to $\Pi_{\text{OT}}^{1:2}$.
 - (iii) If $x_u = 0$ then $\Pi_{\text{OT}}^{1:2}$ outputs $(0_u, b_u)$ to \mathcal{P} , otherwise it outputs $(1_u, 2^{u-1}a + b_u)$. \mathcal{P} denotes whichever output it receives as $\{\{X_{u,1}^{(1)}, \dots, X_{u,m}^{(1)}\}, \dots, \{X_{u,1}^{(\tau)}, \dots, X_{u,m}^{(\tau)}\}, M_u\}$
 - (e) For $k \in \{1, \dots, \tau\}$ denote the 2D table $X^{(k)} := \{(X_{1,1}^{(k)} \parallel \dots \parallel X_{|x|,1}^{(k)}), \dots, (X_{1,m}^{(k)} \parallel \dots \parallel X_{|x|,m}^{(k)})\}$
- (6) For every $k \in \{1, \dots, \tau\}$, \mathcal{P} computes $(view_{k,1}, \dots, view_{k,m}) \leftarrow \Pi_{\text{MPCitH}}(\Pi_{\mathcal{F}}, (X^{(k)}, Y_\ell^{(k)}))$.
- (7) \mathcal{P} computes $\text{MAC}(x) = \sum_{u=1}^{|x|} M_u$ and $C_{\text{MAC}(x)} \leftarrow \text{Com}(\text{MAC}(x))$. Similarly compute $\text{MAC}(v_\ell) = cv_\ell + d$ and $C_{\text{MAC}(v_\ell)} \leftarrow \text{Com}(\text{MAC}(v_\ell))$.
- (8) For all $j \in \{1, \dots, m\}$, $k \in \{1, \dots, \tau\}$, \mathcal{P} sends $C_{\text{MAC}(x)}$, $C_{\text{MAC}(v_\ell)}$, and $C_{view_{j,k}} \leftarrow \text{Com}(view_{j,k})$ to \mathcal{V} .
- (9) \mathcal{V} decommits s , which \mathcal{P} uses to reconstruct $\{\varepsilon_1, \dots, \varepsilon_\tau\}$, $\{0_u, 1_u\}$, and $\{Y_1, \dots, Y_n\}$
- (10) \mathcal{P} verifies the following properties hold for all $i, i' \in \{1, \dots, n\}$, $u \in \{1, \dots, |x|\}$, $k \in \{1, \dots, \tau\}$, $j \in \{1, \dots, m\} \setminus \varepsilon_k$:
$$\bigoplus_{j=1}^m Y_{i,j}^{(k)} = y_i, \quad Y_{i,j}^{(k)} = Y_{i',j}^{(k)} \quad \bigoplus_{j=1}^m 0_{u,j}^{(k)} = 0 \quad \bigoplus_{j=1}^m 1_{u,j}^{(k)} = 1 \quad 0_{u,j}^{(k)} = 1_{u,j}^{(k)}$$
- (11) \mathcal{P} decommits each $\{view_{j,k}\}_{k \in \{1, \dots, \tau\}, j \in \{1, \dots, m\} \setminus \varepsilon_k}$
- (12) \mathcal{V} checks that the decommitted views are consistent with honest executions of $\Pi_{\mathcal{F}}$ output 1.
- (13) \mathcal{P} and \mathcal{V} execute the following ZK proof protocols:
 - (a) $\pi_1 = \{(x, \text{MAC}(x), z) : C_x = \text{Com}(x) \wedge C_{\text{MAC}(x)} = \text{Com}(\text{MAC}(x)) \wedge \text{MAC}(x) = ax + b \wedge C_x = \text{Com}(x) \wedge C_z = \text{Com}(z) \wedge g(x, z) = 1\}(\text{Com}(x), \text{Com}(z), \text{Com}(\text{MAC}(x)), a, b)$
 - (b) $\pi_2 = \{(v_\ell, \text{MAC}(v_\ell)) : C_v = \text{Com}(v_\ell) \wedge C_{\text{MAC}(v_\ell)} = \text{Com}(\text{MAC}(v_\ell)) \wedge \text{MAC}(v_\ell) = cv_\ell + d \wedge v_\ell \geq v_0\}(C_v, C_{\text{MAC}(v_\ell)}, c, d, v_0)$
- (14) If any of π_1, π_2 do not verify, \mathcal{V} rejects, else accepts.

Figure 11: Disjunctive Composite protocol via MPCitH for Proving Assets in Bitcoin. We denote by **colored text** the additional elements introduced compared to Fig.10

Setup. Group \mathbb{G} where DDH assumption holds. $\text{Com}(\cdot)$ is a commitment scheme. Let $\mathcal{G} = (\text{Gb}, \text{En}, \text{De}, \text{Eval}, \text{Ve})$ be a garbling scheme.

Common input. y_1, \dots, y_n where $y_i \in \{0, 1\}^\kappa$.

Prover's input. $x \in \mathbb{G}$, where x is the witness to statement y_ℓ .

Verifier's input. $C_x = \text{Com}(x)$.

Protocol.

- (1) The verifier constructs a garbled circuit for F .

$$(GC, e, d) \leftarrow \text{Gb}(1^\kappa, F(x, y) = y \oplus f(x))$$

- (2) The prover sends (i, x_i) for all $i \in [n]$ to \mathcal{F}_{COT} .

- (3) The verifier sends $(i, (K_i^0, b_i), (K_i^1, 2^i a + b_i))$ for all $i \in [n]$ to \mathcal{F}_{COT} where a and b_i has length of λ and $|x| + \lambda$ bits respectively.

- (4) \mathcal{F}_{COT} outputs $(K_i^{x_i}, 2^i a x_i + b_i)$ for all $i \in [n]$ to the prover.

- (5) The prover sends ℓ to $\mathcal{F}_{\text{COT}}^{1:n}$.

- (6) The verifier sends (y_1, \dots, y_n) to $\mathcal{F}_{\text{COT}}^{1:n}$.

- (7) $\mathcal{F}_{\text{COT}}^{1:n}$ outputs y_ℓ to the prover.

- (8) The verifier sends the garbled circuit GC to the prover.

- (9) The prover evaluates the garbled circuit

$$Z \leftarrow \text{Eval}(GC, \{K_i^{x_i}\}_{i \in [n]}, y_\ell)$$

- (10) The prover computes $t = \sum_{i=0}^{n-1} (2^i a x_i + b_i) = ax + b$.

- (11) The prover commits to the garbled output Z and t by sending $\text{Com}(Z), \text{Com}(t)$ to the verifier and proves knowledge of opening.

- (12) The verifier sends open to \mathcal{F}_{COT} and $\mathcal{F}_{\text{COT}}^{1:n}$.

- (13) \mathcal{F}_{COT} sends (K_i^0, K_i^1) and $\mathcal{F}_{\text{COT}}^{1:n}$ sends (y_1, \dots, y_n) to the prover for all $i \in [n]$.

- (14) The prover verifies that the circuit was garbled correctly by running $\text{Ve}(GC, \{K_i^0, K_i^1\}_{i \in [n]}, F)$ and the garbled inputs for x, y_1, \dots, y_n are correct. If the check fail, the prover terminates. Else, it opens Z to the verifier.

- (15) The verifier checks that $\text{De}(d, Z) = 0$. Otherwise, it rejects and terminates.

- (16) The prover and the verifier execute a ZK proof to prove the following. $\pi = \{(x, t) : C_x = \text{Com}(x) \wedge C_t = \text{Com}(t) \wedge t = ax + b\}(C_x, C_t)$

- (17) If π does not verify, the verifier terminates.

Figure 12: OR Proof using Garbled Circuits with MAC. We denote by **colored text** the additional elements introduced compared to the protocol of Chase et al.

$u_j[j] = 1$ and $u_j[k] = 0$ for $j \neq k$ and $0 \leq j < 22$. Let $y_i = (y_{i,0}, \dots, y_{i,22})$, then $\text{NTT}(y_i) = \sum_{j=0}^{21} y_{i,j} \text{NTT}(u_j)$. We can further speed up the precomputation by using a lookup table that store $\text{NTT}(k \cdot u_j)$ for $k \in [0; 2^{12})$. The table will have at most $22 \times 4096 = 90112$ entries. After the precomputation step, each $\text{NTT}(y_i)$ can be computed by $21 \cdot N$ additions instead of $O(N \log N)$ multiplications. If $n = 2^{20}$, the preprocessing will cost $90112 \cdot N$ multiplications and $21 \cdot 2^{20} \cdot N$ additions with the lookup table. The naive approach will cost $O(2^{20} \cdot N \log N)$ multiplications ($\log N \geq 11$). If we use only 8 bits for each chunk of y_i , the lookup table can be precomputed with only $32 \times 256 \times N = 8192 \cdot N$ multiplications. However, we need to perform $31 \cdot N$ additions for each NTT operation instead of $21 \cdot N$.

Reducing the cost of 1-out-of- n committed OT. With a naive approach, the prover cannot verify the 1-out-of- n committed OT until the verifier decommits the random seed used in the computation. However, with our optimization, both the

prover and verifier can execute the heavy part of the 1-out-of- n committed OT at the same time. The dominant cost of the 1-out-of- n committed OT is the cost to compute $\text{NTT}(y_i)$ and $\sum_{i=1}^N C_i \cdot y_{i+jN}$ for $j \in [1; n/N]$. It is clear that these terms do not depend on the random seed that is used to generate r and s . These computations takes more than 95% of the total runtime of 1-out-of- n committed OT. Thus we save almost $2\times$ in terms of computation cost for the 1-out-of- n committed OT step.

C ELLIPTIC CURVE POINT MULTIPLICATION CIRCUIT

We estimate the size of the Boolean circuit that compute $x \cdot G$ where x is a scalar and G is a publicly known elliptic curve point. Let q be the bit length of the elliptic curve and $x = \sum_{i=0}^{q-1} x_i \cdot 2^i$, then $x \cdot G = \sum_{i=0}^{q-1} x_i \cdot (2^i \cdot G)$. Assume that the computation of G_i is cheap, the size of the circuit is dominated by the cost to add all these $q-1$ points together, each costs $14q$

Setup. Group \mathbb{G} where DDH assumption holds. $\text{Com}(\cdot)$ is a commitment scheme. Let $\mathcal{G} = (\text{Gb}, \text{En}, \text{De}, \text{Eval}, \text{Ve})$ be a garbling scheme.

Common input. $(y_1, v_1), \dots, (y_n, v_n)$ where $y_i \in \{0, 1\}^\kappa, v_i \in [0, L]$.

Prover's input. $x \in \mathbb{G}$, where x is the witness to statement y_ℓ such that $v_\ell \geq v_0$.

Verifier's input. $C_x = \text{Com}(x)$ and $C_{v_\ell} = \text{Com}(v_\ell)$.

Protocol.

- (1) The verifier constructs a garbled circuit for F .

$$(GC, e, d) \leftarrow \text{Gb}(1^\kappa, F(x, y) = y \oplus f(x))$$

- (2) The prover sends (i, x_i) for all $i \in [n]$ to \mathcal{F}_{COT} .

- (3) The verifier sends $(i, (K_i^0, b_i), (K_i^1, 2^i a + b_i))$ for all $i \in [n]$ to \mathcal{F}_{COT} where a and b_i has length of λ and $|x| + \lambda$ bits respectively.

- (4) \mathcal{F}_{COT} outputs $(K_i^{x_i}, 2^i a x_i + b_i)$ for all $i \in [n]$ to the prover.

- (5) The prover sends t to $\mathcal{F}_{\text{OT}}^{1:n}$.

- (6) The verifier sends $((y_1, a'v_1 + b'), \dots, (y_n, a'v_n + b'))$ to $\mathcal{F}_{\text{OT}}^{1:n}$ where a' and b' has length of λ and $|x| + \lambda$ bits respectively.

- (7) $\mathcal{F}_{\text{COT}}^{1:n}$ outputs $(y_\ell, a'v_\ell + b')$ to the prover.

- (8) The verifier sends the garbled circuit GC to the prover.

- (9) The prover evaluates the garbled circuit

$$Z \leftarrow \text{Eval}(GC, \{K_i^{x_i}\}_{i \in [n]}, y_\ell)$$

- (10) The prover computes $t = \sum_{i=0}^{n-1} (2^i a x_i + b_i) = ax + b$ and $v = a'v_\ell + b'$.

- (11) The prover commits to the garbled output Z and t by sending $\text{Com}(Z), \text{Com}(t), \text{Com}(v)$ to the verifier and proves knowledge of opening.

- (12) The verifier sends open to \mathcal{F}_{COT} and $\mathcal{F}_{\text{COT}}^{1:n}$.

- (13) \mathcal{F}_{COT} sends (K_i^0, K_i^1) and $\mathcal{F}_{\text{COT}}^{1:n}$ sends $((y_1, a'v_1 + b'), \dots, (y_n, a'v_n + b'))$ to the prover for all $i \in [n]$.

- (14) The prover verifies that the circuit was garbled correctly by running $\text{Ve}(GC, \{K_i^0, K_i^1\}_{i \in [n]}, F)$ and the garbled inputs for x, y_1, \dots, y_n are correct. If the check fail, the prover terminates. Else, it opens Z to the verifier.

- (15) The verifier checks that $\text{De}(d, Z) = 0$. Otherwise, it rejects and terminates.

- (16) The prover and the verifier execute the following ZK proofs: $\pi_1 = \{(x, t) : C_x = \text{Com}(x) \wedge C_t = \text{Com}(t) \wedge t = ax + b\}$
 $\pi_2 = \{(v_\ell, v) : C_{v_\ell} = \text{Com}(v_\ell) \wedge C_v = \text{Com}(v) \wedge v = a'v_\ell + b' \wedge v_\ell \geq v_0\}$

- (17) If any of π_1, π_2 do not verify, the verifier terminates.

Figure 13: OR Proof with MAC and value Protocol using Garbled Circuits. We denote by colored text the additional elements introduced compared to Fig. 12.

multiplications, or $14q^2(q-1)$ AND gates. Finally, we need to convert the projective coordinates to the regular coordinates by computing one field inverse and 2 multiplications which

cost around q^3 AND gates. In total, the size of the circuit is $15q^3 - 14q^2$ AND gates. For $q = 256$, the circuit will have around 250 million AND gates.

Table 5: Comparison of ZK proof systems for Proof of Assets for a *single* proof. $|x|$ is length of input, $|F|$ circuit size, λ security parameter. (for BTC UTXO $|x| = 512$ (BTC public key 256bits + 256 bits of padding for MD), probably $\lambda < x$, we can consider 128 or 256 sec bits. Circuit F' might be 10 times larger than F)

	No setup	NI	Prover	Verifier	Proof size	Notes
Chase et al. [18] (w/o MAC)	✓	✗	$O(x \text{ pub} + F \text{ sym})$	$O(x \text{ pub} + (F \text{ sym}))$	$O((F + x)\lambda)$	Σ -protocol + GC
Chase et al. [18] (w/ MAC)	✓	✗	$O(\lambda \text{ pub} + (F' + x \lambda) \text{ sym})$	$O(\lambda \text{ pub} + (F' + x \lambda) \text{ sym})$	$O((F' + x \lambda)\lambda)$	Σ -protocol + GC
Backes et al. [8]	✓	✓	$O(x + \lambda \text{ pub} + F \lambda \text{ sym})$	$O(x + \lambda \text{ pub} + F \lambda \text{ sym})$	$O((F \lambda + x)\lambda)$	Ped. Comm. + ZK-Boo
Figure 13	✓	✗	$O(\lambda \text{ pub} + (F \text{ sym}))$	$O(\lambda \text{ pub} + (F \text{ sym}))$	$O((F + x)\lambda)$	Σ -protocol + GC