Mac′n′Cheese: Zero-Knowledge Proofs for Arithmetic Circuits with Nested Disjunctions

Carsten Baum, Aarhus University
Alex J. Malozemoff, Galois, Inc.
Marc Rosen, Galois, Inc.
Peter Scholl, Aarhus University

Abstract

A zero-knowledge proof is a cryptographic primitive that is a versatile building block for both cryptographic protocols alongside a wide range of applications from cryptocurrencies to privacy-preserving auditing. Unfortunately, when the proof statements become very large, existing zero-knowledge proof systems easily reach their limits: either the computational overhead, the memory footprint, or the required bandwidth exceed levels that would be tolerable in practice.

We present an interactive zero-knowledge proof system for arithmetic circuits, called Mac′n′Cheese, with a focus on supporting large circuits while using low computational resources. Our work follows the commit-and-prove paradigm instantiated using information-theoretic MACs based on vector oblivious linear evaluation to achieve high efficiency. We additionally show how to optimize disjunctions, with a general OR transformation for proving the disjunction of m statements that has communication complexity proportional to the longest statement (plus an additive term logarithmic in m). These disjunctions can further be *nested*, allowing efficient proofs about complex statements with many levels of disjunctions. We also show how to make Mac′n′Cheese non-interactive (after a preprocessing phase) using the Fiat-Shamir transform, and with only a small degradation in soundness.

We have implemented the non-interactive variant of the online phase of Mac′n′Cheese and can achieve *2.5 μs per multiplication gate* while requiring a minimal amount of memory: for proving the knowledge of two 512-by-512 matrices that equal some fixed public matrix we require less than 36 MB of memory for both the prover and verifier. We achieve this through a streaming approach which is compatible with our disjunctions over sub-circuits.

1 Introduction

Zero knowledge (ZK) proofs are interactive protocols which allow a prover P to convince a verifier V that a certain statement x is true in such a way that V learns nothing beyond the validity of the statement. ZK proofs have a wide range of applications in cryptography, from signatures [BG90] to compiling other protocols from passive to active security [GMW87]. More recently, ZK proofs have seen widespread applications outside of classical cryptography, for example in the cryptocurrency space [BCG+14]. These constructions mostly focus on *succinctness* and *non-interactivity*; namely, the construction of "succinct" proofs that have a small verification runtime and that do not require interaction between P and V for validation.

However, for sufficiently large statements—on the order of billions of instructions—most existing proof systems fail due to either memory constraints or high prover running times. Systems such as SNARKs [BCG+13] or recent IOP-based constructions such as Ligero [AHIV17] or STARKs [BBHR19] suffer from exactly this drawback: they have an inherent asymptotic prover overhead, paying at least a multiplicative factor $\log(|x|)$ in computation when the statement has length $|x|$, and they need to keep the entire statement x in memory.

1.1 Our Approach: Mac′n′Cheese

In this work we introduce a novel ZK proof system called Mac′n′Cheese that is optimized for statements at scale. We use the *commit-and-prove* paradigm [CD97], where we "commit to" values using an information-theoretic message authentication code (MAC). For each committed value, P holds the MAC'ed value and

the tag, and V holds the MAC key. Such commitments can be generated very efficiently using vector oblivious linear evaluation (VOLE) [BCGI18] and they are linearly homomorphic. This homomorphism makes instantiating the commit-and-prove paradigm possible via known circuit randomization techniques [Bea92].

Naively, this commit-and-prove approach leads to a proof with bandwidth costs that scale linearly with the circuit size. To decrease this, in Mac′n′Cheese we support evaluating disjunctions in the spirit of "stacked garbling" [HK20].[1] Namely, we let the prover only communicate information that is required to evaluate the correct branch among the m disjunctions. Both parties still perform the computations necessary to evaluate each branch, but the verifier uses the messages for the correct branch for all m instances simultaneously. At a high level, our technique can be seen as a generalized OR composition of protocols, where the resulting proof only requires sending information corresponding to the 'true' protocol. We provide and analyze two different flavors of this idea which work over large fields, which we use in Mac′n′Cheese, as well as $\mathbb{F}_2$. Additionally, and unlike the stacked garbling approach [HK20], our disjunctions can be *recursively nested*, allowing proofs about more complex statements than just top-level disjunctions. We also generalize our approach to the threshold setting for showing that r out of m branches are correct, with communication that depends on r branches only.

We implemented the non-interactive variant of the online phase of Mac′n′Cheese in Rust. Currently, we do not have an implementation of VOLE, and thus all of our timings focus on memory usage and communication cost, with the assumption that VOLE will add a small amount of communication—1.3 bits per VOLE [WYKW20, Table 4]—and increase in runtime—130 ns per VOLE [WYKW20, Table 4].

Our benchmarks demonstrate that (the online phase of) Mac′n′Cheese requires approximately 2.5 μs per multiplication alongside using a minimal amount of memory. For example, when proving the existence of two 512-by-512 matrices that when multiplied together equals some public matrix, Mac′n′Cheese uses ≤ 36 MB of memory for both the prover and verifier. In addition, our benchmarks demonstrate the power of nested disjunctions: when proving a disjunction over sixteen 256-by-256 matrix multiplications, we can reduce the total communication from 6.1 GB to 385 MB, a 16× improvement.

1.2 Our Techniques

We present the Mac′n′Cheese approach in three steps: first, we describe the zero-knowledge protocol in a setting with idealized homomorphic commitments to single field elements. Next, we present an abstraction for such protocols which we call *Interactive Protocols with Linear Oracle Verification—IPs with LOVe* for short—and show how to construct such protocols for arithmetic circuit satisfiability, while also supporting nested disjunctions of general statements. Finally, we show how to use vector oblivious linear evaluation (VOLE) to efficiently instantiate IPs with LOVe. This gives rise to multiple specific optimizations that reduce the amount of interaction or simplify streaming.

Circuit satisfiability via idealized homomorphic commitments. Assume that the statement x, together with a witness w, is provided to P while V only obtains x. We consider x as a circuit C over a large finite field $\mathbb{F}$, such that $C(w) = 0$ iff $(x, w) \in \mathcal{R}$ and assume that w is a vector over $\mathbb{F}$.

Implementing the test that $C(w) = 0$ can be done using standard techniques with idealized homomorphic commitments [CD98], but we nevertheless sketch these now. First, P commits to (1) w, (2) triples of the form a, b, c such that $c = a \cdot b$, and (3) the outputs of all the gates of $C(w)$. P and V then engage in an interactive protocol to test that:

1. The commitments to gate outputs are consistent with C and w; and
2. The output of the output gate of C is zero.

Note that all of these checks reduce to testing that certain committed values are zero:

- This is clear for testing the output of the output gate.

[1] Our approach was inspired by stacked garbling, although the technique bears more resemblance to the earlier 'free if' technique for private function evaluation [Kol18].

- For each addition gate (or multiplications with public constants from $\mathbb{F}$) one can simply apply the respective linear operation to the commitments to the inputs of the gate, subtract the commitment of the output and test if the result is a commitment to zero.
- For each multiplication gate, we use Beaver's circuit randomization approach [Bea92, CD98, KOS16] to reduce multiplication to zero-testing a commitment to a linear combination of commitments to the gate inputs, outputs, and the random triples (a, b, c), alongside an additional random element sent by V. (In fact, this random element can be generated by the output of a random oracle on the protocol transcript using the Fiat-Shamir transform. We provide more details on this in § 4.1.)

When instantiating homomorphic commitments with vector oblivious linear evaluation (as we describe later), this basic protocol has an amortized communication complexity of 3 field elements per multiplication gate. This improves upon the arithmetic protocol of Weng et al. [WYKW20], which uses 4 field elements, although they also present a variant with 2 field elements per multiplication which has a higher computational cost due to polynomial operations.

Formalizing security using IPs with LOVe. While the aforementioned protocol is conceptually straightforward, proving security is not immediate, particularly if we want to additionally encompass optimizations such as our approach to disjunctive proofs. In § 3 we therefore introduce *IPs with LOVe*, which provide a different abstraction for ZK proofs tailored to our setting. In this abstraction, P begins by providing some proof string π to an oracle $\mathcal{O}$. The parties then exchange messages for a fixed number of rounds, at which point V sends multiple queries of the form (z_i, y_i) to $\mathcal{O}$. These queries are determined by V based on the messages that it received in the previous rounds. $\mathcal{O}$ truthfully tells V if $\langle \pi, z_i \rangle = y_i$ or not for each of these queries. Eventually, V outputs a bit to represent whether it accepts or not.

IPs with LOVe can be seen as a generalized form of linear interactive oracle proofs (IOPs) [BBC$^+$19], where on top of oracle queries, we allow the prover and verifier to exchange a number of messages. Linear IOPs were motivated [BBC$^+$19] as modelling natural proof systems based on homomorphic commitments, as well as methods for proving statements on secret-shared data. IPs with LOVe are also a natural fit for these types of proofs, while allowing us to go further by supporting optimizations for proofs with disjunctions.

Disjunctive proofs for IPs with LOVe. Our main technical contribution, described in § 3.2, can be seen as a general form of OR composition for IPs with LOVe. The communication complexity in the resulting OR proof is proportional to the *maximum* of that in the original proofs.

We limit ourselves to IPs with LOVe that are *public coin*, i.e., where V only sends messages that are random bits and where the queries to $\mathcal{O}$ can be determined deterministically based on the previous transcript. This is indeed the case for our commit-and-prove protocol. We then go on to show that if one has m such public coin IPs with LOVe $\Pi_1, \cdots, \Pi_m$ whose messages from P to V can be made "compatible", then one can construct a (public coin) IP with LOVe Π whose message complexity essentially only depends on the protocol Π_i which sends the most messages, plus an additional check that requires $O(m)$ communication but is independent of all m IPs of LOVe themselves. V accepts in Π if and only if at least one of the instances Π_i was accepting.

In order to run Π, P and V initially execute $\Pi_1, \ldots, \Pi_m$ in parallel. The key insight is that we only send those messages from P to V that belong to the one protocol Π_{i^*} where P has a witness w_{i^*} for x_{i^*}, padding with dummy messages such that the communication looks as if it could belong to any of the m branches. V uses the one message that it obtains per round for all $\Pi_1, \ldots, \Pi_m$ in parallel, not knowing to which of the m protocols it belongs. Finally, instead of performing the queries to $\mathcal{O}$ at the end of each Π_i, Π runs a standard (small) OR-proof à la Cramer et al. [CDS94] to show that the queries in at least *one* of the m branches are all valid. The trick here is that we can show that this OR-proof can itself be expressed as sending certain messages between P and V followed by queries to $\mathcal{O}$ from V, making Π a public coin IP with LOVe as desired.

Thresholds and recursion. The OR-proof of Cramer et al. [CDS94] can be generalized for any threshold r out of m, showing that indeed at least r instances of $\Pi_1, \ldots, \Pi_m$ were correct. We generalize our protocol Π to this setting, with communication r times that of the original Π, instead of m.

While both of these techniques only work for large fields $\mathbb{F}$, we also show a *recursive proof* that (1) also works over $\mathbb{F}_2$ and (2) has an additive message overhead, for m protocols and large fields, of $O(\log_2(m))$ instead of the $O(m)$ of Π. By using recursion, we are able to capture proofs about complex programs with arbitrary nested levels of disjunctions, with communication proportional to the longest path through the entire program. Both the threshold scheme and the recursive proof are described in more detail in § A.

From IPs with LOVe to IPs with VOLE—efficiently. Towards efficiently instantiating IPs with LOVe, in § 4 we describe a simple high-level syntax for expressing a large class of IPs with LOVe using an abstract homomorphic commitment notation. We refer to these as *commit-and-prove (*C&P*) IPs with LOVe*. This avoids the low-level details in the definition, simplifying the process of specifying and analyzing protocols. To illustrate this, we describe in § 4.2 and § 4.3 our protocols for circuit satisfiability and disjunctions in this framework.

Finally, we show that any C&P IP with LOVe can be combined with a VOLE protocol to obtain a ZK proof. This is described in § 4.4. We instantiate the oracle $\mathcal{O}$ that contains the string π using information-theoretic MACs of the form

$$\mathsf{MAC}_{(\alpha,\beta)}(x) := x\alpha + \beta,$$

where all values are in some field $\mathbb{F}$. We call α the "MAC key" and β the "MAC offset", and sometimes use the notation K to denote the tuple (α, β), held by the verifier, and τ to denote the MAC tag, held by the prover. These commitments are linearly homomorphic for keys that share the α component, so we can realize each oracle query as a zero-test on such commitments.

A batch of n MACs on random values is exactly equivalent to a VOLE of length n, since the MAC relation can be viewed as evaluating a linear function on the input x. This can be generated with high efficiency using recent (random) VOLE protocols based on arithmetic variants of the LPN assumption [BCGI18, BCG$^+$19a, WYKW20], with communication almost independent of n. VOLE on random inputs gives us a committed proof string of *random* elements; the prover can then take any of these random values and adjust them with a masked value to commit to an input of his choice.

Streaming and removing interaction. We wish to obtain a zero-knowledge proof that both has a *small memory footprint*, allowing streaming, and also *minimizes interaction*, so that ideally the proof is completely non-interactive after a one-time preprocessing phase (for generated the random VOLEs). We can achieve a small memory in our protocols by verifying each linear oracle query as it arises during the computation, rather than batching them together at the end. However, this introduces a high degree of interaction, since now the parties have to interact for every multiplication gate in the circuit.

The natural approach to avoiding interaction is to apply the Fiat-Shamir transform by obtaining the verifier's random challenges from a random oracle. However, the low-memory protocol to which we want to apply this has a very large round complexity, possibly even *linear in the circuit size*. Typically, the Fiat-Shamir transform is only applied to constant-round protocols; while there are general results for many-round protocols, in the worst case the soundness can degrade *exponentially* with the number of rounds [BCS16].

Despite this, in § 5 we show that for our circuit satisfiability protocol over a large field, Fiat-Shamir can be applied *without* any serious loss in soundness, beyond the usual scaling by the number of random oracle queries made by a cheating prover. We do this by giving two different Fiat-Shamir transforms that apply to C&P protocols, and analyzing the concrete soundness when applied to our protocols of interest.

1.3 Related Work

As far as we are aware, there are only two ZK proof systems that can successfully scale to large statements: the zero-knowledge garbled circuit (ZKGC) protocol of Jawurek, Kershbaum, and Orlandi [JKO13] and its associated optimizations [FNO15, ZRE15, HK20], and the concurrently developed Wolverine protocol of

Weng et al. [WYKW20]. Both of these approaches have provers that run linear in the proof statement alongside the ability to "stream"—namely, the prover and verifier are not required to store the entire proof statement in memory.

The ZKGC approach currently has a communication cost of 128 bits per AND gate. In addition, Heath and Kolesnikov [HK20] recently showed how to "stack" garbled circuits such that the communication cost of m disjunctions each containing t_i AND gates is reduced from $\sum t_i$ to $\max\{t_i\}$, further improving the performance of the ZKGC approach for certain proof statements. Our disjunction optimization is inspired by this approach [HK20], but is based on very different tools and additionally supports *nesting* disjunctions.

Concurrent to our work, Weng et al. [WYKW20] introduced Wolverine, a ZK protocol based on authenticated multiplication triples generated using vector oblivious linear evaluation, similar to Mac′n′Cheese. Wolverine supports both boolean and arithmetic circuits, while currently Mac′n′Cheese focuses on arithmetic circuits only. While Wolverine shows a better runtime per multiplication (1.6 μs versus 2.5 μs in Mac′n′Cheese), this discrepancy may be due to the multi-threaded nature of Wolverine's implementation, whereas Mac′n′Cheese is currently single-threaded. In addition, Wolverine does not support nesting disjunctions, one of the main contributions of Mac′n′Cheese.

2 Preliminaries

For relation $\mathcal{R} = \{(x, w) : x \in L, w \in W(x)\}$ for sets L and $W(x)$, let $L(\mathcal{R})$ denote the language L of the relation. For any vector r we denote by $r|_t$ the restriction to the first t elements. Let $[\mathsf{P} \leftrightarrow \mathsf{V}]$ denote the distribution of exchanged messages between P and V and let $[\mathsf{P} \leftrightarrow \mathsf{V}]_t$ denote the distribution of the transcript of the messages exchanged in the first t rounds. Denote by $\mathtt{View}_{\mathsf{V}}[\mathsf{P} \leftrightarrow \mathsf{V}]$ the view of V when interacting with P.

3 Interactive Proofs with Linear Oracle Verification

In this section we introduce our proof methodology: *interactive proofs with linear oracle verification* (IPs with LOVe) over some field $\mathbb{F}$. This formalization is a generalization of linear interactive oracle proofs [BBC+19], where in each round, the verifier chooses some linear function, and learns the evaluation of this on a proof string chosen by the prover. In comparison, we let the prover P first fix the proof string π. Then, both P and the verifier V exchange messages for a certain number of rounds. Finally, V issues a number of affine queries to π upon which it makes a decision on whether to accept or not. These queries can depend on the messages that were exchanged between both P and V throughout the protocol.

We let P fix π at the beginning of the protocol and allow V to access it via oracle queries only at the end of the protocol. That means that for $\pi \in \mathbb{F}^\ell$ we let V choose q queries $(z_1, y_1), \ldots, (z_q, y_q) \in \mathbb{F}^\ell \times \mathbb{F}$ which it sends to an oracle that stores π. This oracle checks that for each of the q queries the relation $\langle \pi, z_i \rangle = y_i$ holds. The query results are then (truthfully) reported to V by the oracle.

Note that in case $|\mathbb{F}|$ is superpolynomial in the security parameter, it is possible to always simply set $q = 1$ by randomly combining all queries into one. Since we want our definition to also capture instantiations over small fields including $\mathbb{F}_2$, we nevertheless allow for multiple queries.

Definition 1 (Interactive Protocol with Linear Oracle Verification)**.** *Let $\mathbb{F}$ be a field and $\ell, t, q \in \mathbb{N}$. Then a t-round q-query* interactive protocol with linear oracle verification $\Pi = (\mathsf{P}, \mathsf{V})$ *with oracle length ℓ, message lengths $r_1^{\mathsf{P}}, r_1^{\mathsf{V}}, \ldots, r_t^{\mathsf{P}}, r_t^{\mathsf{V}} \in \mathbb{N}$ and message complexity $\sum_{h=1}^{t}(r_h^{\mathsf{P}} + r_h^{\mathsf{V}})$ over $\mathbb{F}$ consists of the algorithm P and PPT algorithm V that interact as follows:*

1. *Initially, P obtains its respective input while V obtains the statement x. P then submits a string $\pi \in \mathbb{F}^\ell$ to the oracle, after which both parties receive a random, unique proof identifier $\mathsf{id}_\pi \in \{0,1\}^*$.*[2] *$\mathsf{P}$ then outputs a state s_0^{P} while V outputs a state s_0^{V}. We set an auxiliary variable $a_0 = \bot$.*

[2]We only use the random identifiers in our non-interactive Fiat-Shamir transformation in § 5, where we require that id_π should be unpredictable to P until it has chosen π.

2. *For round* $h \in [t]$, P *and* V *do the following:*

   (a) *First,* V *on input* s^{V}_{h-1} *and* a_{h-1} *outputs message* $e_h \in \mathbb{F}^{r^{\mathsf{V}}_h}$ *and state* s^{V}_h.

   (b) *Then,* P *on input* s^{P}_{h-1} *and* e_h *outputs message* $a_h \in \mathbb{F}^{r^{\mathsf{P}}_h}$ *and state* s^{P}_h.

3. *Finally,* V *on input* a_t *and state* s^{V}_t *makes* q *linear oracle queries to* π *and outputs a bit.*

We say that the protocol accepts *if* V *outputs 1 at the end of the protocol.*

Definition 2 (Honest-Verifier Zero-Knowledge Interactive Proof with Linear Oracle Verification)**.** *A* t*-round* q*-query interactive protocol with linear oracle verification* $\Pi = (\mathsf{P}, \mathsf{V})$ *over* $\mathbb{F}$ *is an* honest-verifier zero-knowledge interactive proof with linear oracle verification *(HVZK IP with LOVe) for a relation* $\mathcal{R}$ *with soundness error* ϵ *if it satisfies the following three properties:*

Completeness: *For all* $(x, w) \in \mathcal{R}$ *the interaction between* $\mathsf{P}(x, w)$ *and* $\mathsf{V}(x)$ *is accepting.*

Soundness: *For all* $x \notin L(\mathcal{R})$ *and for all (unbounded) algorithms* P^**, any interaction of* P^* *with* $\mathsf{V}(x)$ *is accepting with probability at most* ϵ.

Honest-Verifier Zero-Knowledge: *There exists a PPT algorithm* S *such that for any* $(x, w) \in R$ *the output of* $S(x)$ *is perfectly indistinguishable from* $\mathtt{View}_V[\mathsf{P}(x, w) \leftrightarrow \mathsf{V}(x)]$ *for any honest* V.

We use the notation $\mathsf{IP\text{-}LOVe}^{\mathcal{R},\mathbb{F}}_{t,q,\ell,\alpha,\epsilon}$ to denote a t-round, q-query HVZK IP with LOVe for relation $\mathcal{R}$ over field $\mathbb{F}$ with oracle length ℓ, message complexity α, and soundness error ϵ.

In this work, all the protocols we construct will additionally be proofs of knowledge and public coin, as in the following definitions.

Definition 3 (ZK Interactive Proof of Knowledge with LOVe)**.** *An* $\mathsf{IP\text{-}LOVe}^{\mathcal{R},\mathbb{F}}_{t,q,\ell,\alpha,\epsilon}$ *protocol* Π *is a* proof of knowledge *if for any accepting proof* π *for a statement* x *there exists a PPT extractor* E *that, on input* π*, outputs a witness* w *such that* $(x, w) \in \mathcal{R}$.

Definition 4 (Public Coin IP with LOVe)**.** *An* $\mathsf{IP\text{-}LOVe}^{\mathcal{R},\mathbb{F}}_{t,q,\ell,\alpha,\epsilon}$ *protocol* Π *is* public coin *if*

1. V *chooses each* $e_h \in \mathbb{F}^{r^{\mathsf{V}}_h}$ *for* $x \in L(\mathcal{R})$ *uniformly at random (and in particular,* independent *of* s^{V}_{h-1} *and* a_{h-1}*).*

2. *There exists a deterministic polytime algorithm* $\mathcal{Q}$*, which, on input* x *and* $\{e_h, a_h\}_{h\in[t]}$*, computes the* q *oracle queries* $(z_1, y_1), \ldots, (z_q, y_q)$ *of* V.

3. V *accepts iff all queries generated by* $\mathcal{Q}$ *are accepting.*

3.1 Stackable Public Coin IPs with LOVe

We now show that when both P and V agree on m relations $\mathcal{R}_1, \ldots, \mathcal{R}_m$ and instances $x_1, \ldots, x_m$ that can each be proven using (public coin HVZK) IPs with LOVe, then we can construct a communication-efficient protocol showing that at least one of the statements was true. Following the terminology of stacked garbling [HK20], we sometimes refer to this as a *stacked proof*.

More formally, the goal of P is to show that $(x_1, \ldots, x_m) \in L(\mathcal{R}_{\mathtt{OR}})$ where

$$(x_1, \ldots, x_m) \in L(\mathcal{R}_{\mathtt{OR}}) \iff x_1 \in L(\mathcal{R}_1) \vee \cdots \vee x_m \in L(\mathcal{R}_m).$$

Throughout this section, we will write $\hat{x}$ as a short-hand for $x_1, \ldots, x_m$ when the statements are clear from the context. Given IPs with LOVe for each instance x_i that each have message complexity α_i and query complexity q_i then it is possible to use a standard OR-proof such as that shown by Cramer et al. [CDS94] to construct an IP with LOVe with message complexity $\approx \sum_{i\in[m]} \alpha_i$ and query complexity $\approx \sum_{i\in[m]} q_i$. We

show how to instead obtain a message complexity $\sum_{i\in[m]} q_i + 2m + \max\{\alpha_i\}$ and query complexity m. We also give a variant where the message complexity scales with $O(\log m)$, instead of $2m$.

Towards this, we introduce the notion of equisimulatable IPs with LOVe. The idea is that we can compress the messages for the different proof branches sent by P in such a way that for the true branch, the correct message can be recovered by V. The distribution of the values for non-taken branches which V will obtain is indistinguishable from a real protocol execution.

For example, assume that in Π_1 P would in each round send 1 field element that appears uniformly random to V, while in Π_2 it sends 2 such elements with the same property. To achieve equisimulatability, if P takes the first branch it can always append a uniformly random element to the message it sends to V, whereas in the second case it just sends the actual message.

Definition 5 (Equisimulatable IPs with LOVe)**.** *Let $\Pi_1, \ldots, \Pi_m$ be protocols such that each Π_i is an IP with LOVe over the field $\mathbb{F}$ for the relation $\mathcal{R}_i$ with round complexity t_i. We say that $\Pi_1, \ldots, \Pi_m$ are* equisimulatable *if there exist two algorithms $\mathcal{CP}$ and $\mathtt{dec}$ such that:*

1. $\mathtt{dec}(\hat{x}, i, h, \mathcal{CP}(\hat{x}, i, h, a_h)) = a_h$ *if* $a_h \leftarrow \Pi_i(s^{\mathsf{P}}_{h-1}, e_h)$ *where Π_i's inputs are from an honest execution of Π_i; and*
2. $[\mathcal{CP}(\hat{x}, i, h, a_h) \mid a_h \leftarrow \Pi_i(s^{\mathsf{P}}_{h-1}, e_h)] = [\mathcal{CP}(\hat{x}, j, h, a_h) \mid a_h \leftarrow \Pi_j(s^{\mathsf{P}}_{h-1}, e_h)]$ *where both the inputs of Π_i and Π_j come from honest executions.*

We say that $\mathcal{CP}$ has message complexity α if the total number of $\mathbb{F}$-elements generated by $\mathcal{CP}$ for all $h \in [\max_{i\in[m]} t_i]$ is at most α.

If P in each IP with LOVe in each round only sends one element to V that always appears to be uniformly random from $\mathbb{F}$, then the message complexity $\alpha = \max_{i\in[m]} t_i$. This is the setting that we will mostly consider in this work.

3.2 Stacking with LOVe

Using the concept of *equisimulatability* of protocols we now show how to lower the message complexity when proving $\mathcal{R}_{\mathtt{OR}}$. The idea is inspired by the stacked garbling approach [HK20]. We assume that m IPs with LOVe Π_i exist for the individual $\mathcal{R}_i$ and construct an interactive protocol $\Pi_{\mathtt{OR}}$ which works as follows: P, having only w_{i^*} for one of the statements x_{i^*}, will generate the oracle string π by running Π_{i^*}'s first step to create π_{i^*}, which it then pads with extra random data. Then, P and V will *simultaneously* run all $\Pi_1, \ldots, \Pi_m$, with the following modification: P's message c_h to V in round h will be determined from $a_{h,i}$ using $\mathcal{CP}$, while V extracts the message $a_{h,i}$ for each of the instances from c_h using $\mathtt{dec}$. Due to equisimulatability, V can now execute all instances in parallel but cannot know which of these is the true one. Conversely, since all Π_i are public coin, V sends a randomness string that is long enough for any of the m instances in round h. The message complexity is now determined by $\mathcal{CP}$ and not the individual proofs.

Instead of performing the queries for each Π_i, which would reveal the index i^* of the true statement, we perform a small OR-proof that shows that the queries for at least one Π_i are all accepting. In order to perform this OR-proof, we use the additional random values in π. The complete protocol $\Pi_{\mathtt{OR}}$ can be found in Figure 1.

Theorem 1. *Let $|\mathbb{F}| \gg 2^\lambda$ and $\Pi_1, \ldots, \Pi_m$ be protocols such that each Π_i is a t_i-round q_i-query equisimulatable Public Coin IP with LOVe over $\mathbb{F}$ for relation $\mathcal{R}_i$ with oracle length ℓ_i and soundness error ϵ_i. Furthermore, assume that $\mathcal{CP}$ has overall message complexity α. Then the protocol $\Pi_{\mathtt{OR}}$ in Figure 1 is a Public Coin IP with LOVe for the relation $\mathcal{R}_{\mathtt{OR}}$ with*

1. *round complexity $3 + \max_{i\in[m]} t_i$;*
2. *oracle length $m + \max_{i\in[m]} \ell_i$;*
3. *query complexity m;*

Protocol $\Pi_{\mathtt{OR}}$

Let $\Pi_1, \ldots, \Pi_m$ be protocols such that each Π_i is t_i-round q_i-query equisimulatable public coin IP with LOVe over $\mathbb{F}$ for relation $\mathcal{R}_i$ with oracle length ℓ_i.

Both P and V have inputs $x_1, \ldots, x_m$ where $x_i \in L(\mathcal{R}_i)$. P additionally has input w_{i^*} for (at least) one $i^* \in [m]$ such that $(x_{i^*}, w_{i^*}) \in \mathcal{R}_{i^*}$. We define $q := \sum_{i\in[m]} q_i$, $\ell := \max_{i\in[m]} \ell_i$, and $t := \max_{i\in[m]} t_i$. Let $z_{k,i} = \overline{z}_{k,i} | \underbrace{0 \cdots 0}_{\ell - \ell_i \text{ times}}$.

1. P initially simulates Π_{i^*} on input (x_{i^*}, w_{i^*}) to obtain the string π_{i^*}. It then sets
$$\pi' = \pi_{i^*} | \underbrace{0 \cdots 0}_{\ell - \ell_{i^*} \text{ times}} \quad \text{and} \quad \pi = \pi' | r_1 \cdots r_m$$
where all r_i are chosen uniformly at random.
2. Define $s_0^{\mathsf{P}} := (x_{i^*}, w_{i^*})$. For $h \in [t]$, P and V do the following:
   (a) Let $r_{h,i}^{\mathsf{V}}$ be the length of the challenge that V would send for protocol Π_i in round h. V sets $r_h = \max_{i\in[m]} r_{h,i}^{\mathsf{V}}$, samples $e_h \leftarrow \mathbb{F}^{r_h}$ uniformly at random and then sends it to P.
   (b) P sets $(a_h, s_h^{\mathsf{P}}) \leftarrow \Pi_{i^*}(s_{h-1}^{\mathsf{P}}, e_h)$ where P only uses the first r_{h,i^*}^{P} elements of e_h as required by Π_{i^*}. It then computes $c_h \leftarrow \mathcal{CP}(\hat{x}, h, i^*, a_h)$ and sends c_h to V.
3. For V samples m strings $\rho_i \leftarrow \mathbb{F}^{q_i}$ each uniformly at random from $\mathbb{F}$ and sends these to P.
4. For $i \in [m]$, P samples $f_i \leftarrow \mathbb{F} \setminus \{i^*\}$ uniformly at random, computes $\{(\overline{z}_{k,i}, y_{k,i})\}_{k\in[q_i]} \leftarrow \mathcal{Q}(x_i, \{e_h, \mathtt{dec}(\hat{x}, h, i, c_h)\}_{h\in[t_i]})$ and sets $g_{k,i} := \langle \pi', z_{k,i} \rangle$. It then computes $g_i = \langle \rho_i, [g_{1,i} \cdots g_{q_i,i}] \rangle$, $y_i = \langle \rho_i, [y_{1,i} \cdots y_{q_i,i}] \rangle$ and $d_i := (g_i - y_i)/f_i + r_i$. P additionally sets $d_{i^*} := r_{i^*}$. Finally, P sends $\{d_i\}_{i\in[m]}$ to V.
5. V samples $f \leftarrow \mathbb{F}$ uniformly at random and sends it to P.
6. P sets $f_{i^*} := f - \sum_{i\in[m]\setminus\{i^*\}} f_i$ and sends $f_1, \ldots, f_m$ to V. V checks that $f =_? \sum_{i\in[m]} f_i$, aborting if not.
7. Define $\beta_i \in \mathbb{F}^q$ to be the vector that is f_i on the ith position and 0 everywhere else. For $i \in [m]$, V first generates $\{(\overline{z}_{k,i}, y_{k,i})\}_{k\in[q_i]}$ like P in Step 4 and sets $z_i = \langle \rho_i, [z_{1,i} \cdots z_{q_i,i}] \rangle$, $y_i = \langle \rho_i, [y_{1,i} \cdots y_{q_i,i}] \rangle$. Then for each $i \in [m]$ it sends the query $(z_i | \beta_i, y_i + f_i d_i)$ to the oracle. V accepts if all queries are true.

Figure 1: The protocol $\Pi_{\mathtt{OR}}$ for an OR-statement.

4. *message complexity* $2m + \sum_{i\in[m]} q_i + \alpha$*; and*
5. *soundness error* $\sum_{i\in[m]} \epsilon_i + (m+1)/|\mathbb{F}|$.

Proof. We now show that both Definition 2 and Definition 4 are fulfilled.

Completeness. Assume that, as stated in the protocol, $(x_{i^*}, w_{i^*}) \in \mathcal{R}_{i^*}$. This means that the protocol will indeed run for t rounds without abort, as P runs Π_{i^*} and a simulated continuation in Step 2 which are both well-defined and because V runs a t-round protocol that is agnostic of the messages received from P until this point. For the queries, we observe that if $\pi, f_i, d_i, z_i, y_i\beta_i, \rho_i$ are chosen as in the protocol, then for $i \neq i^*$ we have

$$\begin{aligned}
\langle \pi, z_i | \beta_i \rangle &= \langle \pi', z_i \rangle + \langle r_1 \cdots r_m, \beta_i \rangle = \langle \pi', \sum\nolimits_{k\in[q_i]} \rho_i[k] z_{k,i} \rangle + f_i r_i \\
&= \sum\nolimits_{k\in[q_i]} \rho_i[k] \langle \pi', z_{k,i} \rangle + f_i r_i = \sum\nolimits_{k\in[q_i]} \rho_i[k] g_{k,i} + f_i r_i \\
&= g_i + f_i r_i = y_{k,i} + f_i d_{k,i}
\end{aligned}$$

where the last step follows from the definition of d_i. For the i^*th instance we have

$$\begin{aligned}\langle \pi, z_{i^*} | \beta_{i^*} \rangle &= \langle \pi_{i^*}, z_{i^*} \rangle + f_{i^*} r_{i^*} = \langle \pi_{i^*}, \sum\nolimits_{k \in [q_{i^*}]} \rho_{i^*}[k] z_{k,i^*} \rangle + f_{i^*} r_{i^*} \\ &= \sum\nolimits_{k \in [q_{i^*}]} \rho_{i^*}[k] \langle \pi_{i^*}, z_{k,i^*} \rangle + f_{i^*} r_{i^*} = \sum\nolimits_{k \in [q_{i^*}]} \rho_{i^*}[k] y_{k,i^*} + f_{i^*} r_{i^*} \\ &= y_{i^*} + f_{i^*} r_{i^*} = y_{i^*} + f_{i^*} d_{i^*}.\end{aligned}$$

The round, oracle, query and message complexity can be obtained from the definition of the protocol.

Soundness. For the sake of notation we separate the messages sent in $\Pi_{\mathtt{OR}}$ into two phases. The messages sent in Step 2 are considered as *Phase 1* while those exchanged in Step 3 or thereafter are considered *Phase 2*.

Consider a setting where we would perform the regular $\mathsf{IP\text{-}LOVe}^{\mathcal{R},\mathbb{F}}_{t,q,\ell,\alpha,\epsilon}$-queries for each of the m statements after Phase 1. This would mean that we ran $\Pi_{\mathtt{OR}}$ without the OR-proof in the end. If we look at all messages sent during Phase 1 then if none of the m statements is true, we have that for each $i \in [m]$ at least one verification query must fail, except with probability $\sum_{i \in [m]} \epsilon_i$. This can be seen as follows: in the worst case, an ϵ_1-fraction of all Phase 1 challenges by V will lead to all oracle queries related to the $\mathsf{IP\text{-}LOVe}^{\mathcal{R},\mathbb{F}}_{t,q,\ell,\alpha,\epsilon}$ for x_1 being true, as Π_1 has soundness error ϵ_1. The same can be said about all other $m-1$ proofs, and in the worst case the transcripts on which they successfully finish are distinct. Hence if all statements are false, we know that when entering Phase 2 with probability at least $1 - \sum_{i \in [m]} \epsilon_i$ at least one oracle query per $i \in [m]$ must be false, or, alternatively written, $\forall i \in [m] \exists k \in [q_i] : \langle \pi|_{\ell_i}, \overline{z}_{k,i} \rangle \neq y_{k,i}$.

In this case there then are two situations, in which the protocol would terminate with an accepting verifier:

1. Either for some i we have that $\sum_{k \in [q_i]} \rho_i[k] \langle \pi|_{\ell_i}, \overline{z}_{k,i} \rangle = \sum_{k \in [q_i]} \rho_i[k] y_{k,i}$, or
2. For at most one choice of f V will accept.

In the first case, we easily see that the statement is equivalent to some ρ_i being in the kernel of the map defined by the non-zero vector $[\langle \pi|_{\ell_i}, \overline{z}_{1,i} \rangle - y_{1,i}, \ldots, \langle \pi|_{\ell_i}, \overline{z}_{q_i,i} \rangle - y_{q_i,i}]$, which happens with probability at most $1/|\mathbb{F}|$ for each i and $m/|\mathbb{F}|$ over all m instances.

In the second case, assume that there exist two accepting transcripts of messages that keep all messages of Phase 1 and 2 fixed, except for f (and therefore also some f_i), and where $\forall i \in [m] : \langle \pi, z_i \rangle = y_i + \Delta_i$ for some non-zero Δ_i by assumption. The two accepting transcripts have $f, \hat{f}$ as messages by V. We furthermore know that there must exist also different $f_i, \hat{f}_i$ sent by P as the transcripts are accepting.

It must hold that $\langle \pi, z_i | \beta_i \rangle = y_i + f_i d_i$ and $\langle \pi, z_i | \hat{\beta}_i \rangle = y_i + \hat{f}_i d_i$. β_i is by definition 0 everywhere except at one point, where it is f_i (and similarly $\hat{\beta}_i$). So we can write the aforementioned equalities as $y_i + \Delta_i + x_i f_i = y_i + f_i d_i$ and $y_i + \Delta_i + x_i \hat{f}_i = y_i + \hat{f}_i d_i$ where x_i is the value at the index of π that both $\beta_i, \hat{\beta}_i$ select. Therefore $\Delta_i = f_i(d_i - x_i) = \hat{f}_i(d_i - x_i)$. Since $f_i \neq \hat{f}_i$ this can only be fulfilled if $d_i = x_i$, implying $\Delta_i = 0$ which contradicts the assumption.

A cheating P would, by the argument above, have to hit the $\sum \epsilon_i$-fraction on Phase 1-challenges that make the queries accept, or has to end up in the kernel of one of the ρ_i, or otherwise can only answer one choice of f. By a union bound, the cheating probability is at most $\sum_{i \in [m]} \epsilon_i + (m+1)/|\mathbb{F}|$.

Honest-Verifier Zero-Knowledge. We now construct a simulator $\mathcal{S}$ for the protocol $\Pi_{\mathtt{OR}}$ that will, on input $x_1, \ldots, x_m$, output a transcript τ that is perfectly indistinguishable from a real transcript generated by interacting with an honest V.

Pick a protocol instance Π_j at random, run its HVZK simulator $\mathcal{S}_i$ and generate all c_h using $\mathcal{CP}$ as well as π_j. This will be perfectly indistinguishable from any transcript of Π_i with $i \neq j$ by the HVZK property of Π_j and the definition of Equisimulatability. Choose uniformly random ρ_i as in the real protocol.

Now fix random values $f_1, \ldots, f_m$ and choose all d_i uniformly at random. Run $\mathcal{S}_i$ of Π_i for each $i \in [m]$ to generate the respective queries $(\overline{z}_{k,i}, y_{k,i})$ that V would perform in Π_i and recompute the values g_i, y_i as in $\Pi_{\mathtt{OR}}$. Set π to begin with π_i, padded to the right length ℓ with 0s, and add values r_i that fulfill $d_i = (g_i - y_i)/f_i + r_i$ as in $\Pi_{\mathtt{OR}}$. Then output $d_i, f_i, f = \sum_{i \in [m]} f_i$.

By definition, f is uniformly random and all query responses are accepting. All f_i are uniformly random, conditioned on summing up to f as in the protocol. All d_i are uniformly random, as in $\Pi_{\mathtt{OR}}$ where they are computed from the uniformly random r_i. The distribution of ρ_i is identical to that of $\Pi_{\mathtt{OR}}$.

Public Coin. All challenges sent by V are chosen as required in Definition 4: during the t rounds, they are chosen as random strings of sufficient length. The challenge f is chosen uniformly at random as well, and so are all the ρ_i. For each of the queries $(z_i|\beta_i, y_i + f_i d_i)$ the values z_i and y_i are chosen deterministically by using $\mathcal{Q}$ and applying a deterministic transformation. β_i are determined by $f_1, \ldots, f_m$ that are sent by P. □

Generalizing to threshold proofs. The OR-proof used implicitly in $\Pi_{\mathtt{OR}}$ is a version of the technique from Cramer et al. [CDS94]. The authors describe how to additionally construct proofs of partial knowledge for any threshold, i.e., how to show that r out of the m statements are true. Their technique, together with a modification of $\Pi_{\mathtt{OR}}$, can be used to construct a proof in our setting where we implicitly only communicate the transcript of r statements, and not all t of them. $\Pi_{\mathtt{OR}}$ can then be seen as a special case where $r = 1$. More details are found in § A.1.

3.3 Recursive Stacking

$\Pi_{\mathtt{OR}}$ from § 3.2 has the disadvantage that it cannot run over a field of small size, since all the values f_i that are chosen by P in advance must be invertible. While this is not a problem when $|\mathbb{F}|$ is exponential in the security parameter, it can leak information about which branch is actually true if, e.g., $|\mathbb{F}| = 2$. In that case, P has to choose $f_i = 1$. But if V sends a challenge that forces f_{i^*} to be zero (which happens with probability $1/2$) then we know that branch i^* is the true branch in the OR proof.

We will now show an alternative construction which does not suffer from this problem. It has the additional advantage of introducing an overhead that is *only logarithmic in* m for large enough choices of $|\mathbb{F}|$. The intuition behind this alternative protocol is the following observation:

1. Any $\mathsf{IP\text{-}LOVe}^{\mathcal{R},\mathbb{F}}_{t,q,\ell,\alpha,\epsilon}$ Π accepts iff all queries are accepting. This means that for all its q queries $(z_1, y_1), \ldots, (z_q, y_q)$ it must hold that $\langle \pi, z_i \rangle = y_i$ i.e. $\langle \pi, z_i \rangle - y_i = \mu_i = 0$.
2. Similar to $\Pi_{\mathtt{OR}}$ we let V choose q random field elements $\rho_1, \ldots, \rho_q$ in $\mathbb{F}$. Then a statement is therefore true except with probability $1/|\mathbb{F}|$ if $0 = \mu = \sum_{i \in [q]} \rho_i \mu_i$. Observe that we equivalently have that $\mu = \langle \pi, z \rangle - y$ for $z = \sum_{i \in [q]} \rho_i z_i$ and $y = \sum_{i \in [q]} \rho_i y_i$.
3. If we simulate the parallel evaluation of multiple instances as in $\Pi_{\mathtt{OR}}$ then it is only for the "true" branch i^* that all its μ_{k,i^*} are 0. Equivalently, we have that only for the true branch the value μ_{i^*}, which is a random linear combination, will always be 0. Standard amplification also ensures this for small fields $\mathbb{F}$.
4. If we now compute the product $\mu_1 \cdots \mu_m$ correctly and it is 0, then at least one μ_j was 0 to begin with.

Clearly this approach works if we perform $m - 1$ multiplications between the m implicit variables μ_i and it will also work over $\mathbb{F}_2$ using amplification techniques. A drawback, on the other hand, is that it does not give rise to a k-out-of-m proof.

To see why the overhead can be logarithmic in m, we use the fact that after combining two protocols Π_1, Π_2 in such a way *the outcome is again stackable*: if we consider all multiplications as a tree, then we only have to provide those values necessary to prove a correct multiplication that are on the path from μ_{i^*} to the root. The actual proof for this proceeds in the following steps:

1. First we show that if Π_1, Π_2 fulfill similar conditions as in $\Pi_{\mathtt{OR}}$ then we can combine them using the multiplication-based approach.
2. Next, we show that starting with $2m$ proofs $\Pi_1, \ldots, \Pi_{2m}$ with similar conditions as in $\Pi_{\mathtt{OR}}$, if we construct proofs Π'_i from Π_{2i-1}, Π_{2i} using the multiplication method, then $\Pi'_1, \ldots, \Pi'_m$ again fulfill the same conditions i.e. are stackable. Also, this can be done with an overhead that is only as big as one Π_i plus one multiplication (or more, depending on $\mathbb{F}$).

3. Finally, by recursing the previous step, we obtain the log-overhead OR-proof.

The full construction together with a proof can be found in § A.2.

4 Building Zero Knowledge Proofs from IP with LOVe and VOLE

In this section, we first define (§ 4.1) a high-level syntax for specifying a subclass of IPs with LOVe, which models *commit-and-prove* (C&P) protocols based on homomorphic commitments. We then describe a simple protocol for arithmetic circuit satisfiability (§ 4.2) over a large finite field, with communication cost of 3 field elements per multiplication gate. To illustrate the disjunctions construction from § 3.2, we then give an example of how to obtain a disjunctive proof of several C&P statements. Finally, we show that by combining any C&P IP with LOVe with vector oblivious linear evaluation (VOLE) we get a zero knowledge proof system (§ 4.4).

4.1 Defining C&P Protocols

A C&P IP with LOVe runs between a prover P and a verifier V, and works over a finite field $\mathbb{F}$ which may depend on the security parameter. We use the notation $\underline{x}$ to mean that some value $x \in \mathbb{F}$ known by P has been committed to. Note that at this point, we do not consider any concrete commitment scheme or properties of commitments; rather, $\underline{x}$ is just a public, abstract identifier for the commitment.

In the protocol, we model the linear verification oracle by a special instruction AssertZero, which looks at the value in a commitment and checks whether this is zero. Since the parties may also perform linear operations on commitments, this allows modeling any linear query.

A C&P protocol is specified as follows:

Input phase: P chooses its input $w_1, \ldots, w_n \in \mathbb{F}$. Both parties are given the commitment identifiers $\underline{w_1}, \ldots, \underline{w_n}$, and a random identifier $\mathsf{id}_\pi \in \{0,1\}^\lambda$.

Protocol phase: The parties run a sequence of instructions of the following types:

- Random(): Outputs $\underline{r}$ for random $r \leftarrow \mathbb{F}$.
- $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(x)$: Sends value $x \in \mathbb{F}$ from P to V.
- $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(x)$: Sends value $x \in \mathbb{F}$ from V to P.
- $+(\underline{x}, \underline{y})$: Given commitments $\underline{x}$ and $\underline{y}$, outputs $\underline{z}$ for $z = x + y$.
- $+(\underline{x}, y)$: Given commitment $\underline{x}$ and value $y \in \mathbb{F}$, outputs $\underline{z}$ for $z = x + y$.
- $\cdot(\underline{x}, c)$: Given commitment $\underline{x}$ and scalar c, outputs $\underline{z}$ for $z = x \cdot c$.
- AssertZero($\underline{x}$): Asserts to V that $\underline{x}$ is a commitment to $x = 0$, aborting if not.

Output phase: If none of the AssertZero instructions failed, the verifier outputs 1. Otherwise, it outputs 0.

We can see that any C&P protocol Π for proving some property $\mathcal{P}$ about the inputs defines an IP with LOVe, where $\mathcal{P}$ is a statement for the $\mathcal{NP}$ relation $\mathcal{R}$, with $(x, w) \in \mathcal{R}$ if and only if $\mathcal{P}(w) = 1$. This translation of the C&P syntax is as follows:

1. For every Random instruction in Π, P picks a random $r_i \leftarrow \mathbb{F}$.
2. On input $(w_1, \ldots, w_n)$, P inputs to the oracle the proof string $\pi = (r_1, \ldots, r_t, w_1, \ldots, w_n)$, where t is the number of Random instructions required in Π.
3. Both parties then learn the random identifier id_π.
4. In order, the parties run every Send instruction in Π.

5. For each $\mathsf{AssertZero}(\underline{x})$ instruction, V makes a linear oracle query as follows:
   - Since $\underline{x}$ was obtained only from addition and multiply-by-constant gates applied to commitments to the values in π, V can compute $z \in \mathbb{F}^\ell$ and $y \in \mathbb{F}$ such that $x = \langle \pi, z \rangle + y$.
   - Query (z, y) to the oracle.
6. If all oracle queries return zero, V outputs 1.

Notice that when analyzing soundness of a C&P IP with LOVe, it is convenient to observe that each linear oracle query succeeds if and only if the input to its corresponding AssertZero instruction was zero.

Remark 1. *When considering the soundness of a* C&P *IP with LOVe, we need to allow the malicious prover to pick (ahead of time) the randomness r_i used in each* Random *instruction. This is never a problem in our constructions, since we only rely on this randomness to mask the prover's inputs. This stronger requirement is also useful when we later instantiate the protocol with VOLE, where it may be possible for a malicious prover to bias the random VOLE outputs.*

Below, we show that any (public-coin) C&P protocol where the sender's messages are all uniformly distributed is guaranteed to be zero-knowledge, and also satisfies equisimulatability, allowing it to be efficiently compiled into proofs of disjunctive statements.

Theorem 2. *Let Π be a* C&P *IP with LOVe for proving a property $\mathcal{P}$ over the field $\mathbb{F}$. Then, if the set of values input to* Send *in an honest execution is (perfectly) indistinguishable from random, it holds that*

1. *Π is honest-verifier zero-knowledge; and*
2. *m copies of Π (potentially for proving different properties $\mathcal{P}$) are* equisimulatable *(Definition 5).*

Proof. We first prove honest-verifier zero-knowledge by constructing a simulator $\mathcal{S}$ as required. $\mathcal{S}$ on input the statement x simulates the interaction in Π by sending random field elements to V, for every message sent by P. Finally, for each oracle query (z, y) made by V, $\mathcal{S}$ outputs zero. It is easy to see that $\mathcal{S}$ is perfectly indistinguishable from $\mathtt{View}_V[\mathsf{P}(x, w) \leftrightarrow \mathsf{V}(x)]$ for an honest V given the assumption that the values input to Send are indistinguishable from random: because V is honest it only queries the oracle on inputs that equal zero, and hence $\mathcal{S}$ always outputting zero on its oracle queries is perfectly indistinguishable from a real execution.

We next argue that for m protocols $\Pi_1, \ldots, \Pi_m$, where each Π_i is proving some property $\mathcal{P}_i$, the collection $(\Pi_1, \ldots, \Pi_m)$ is equisimulatable. We first need to give an algorithm $\mathcal{CP}$, which on input an index i, round number h and prover message a_h from an execution of Π_i, outputs the 'combined prover' message a'; our algorithm simply extends a_h to the maximum length of the prover's round-h message in any $\{\Pi_j\}_j$, by padding with random field elements.

Since the prover's messages are all uniformly random, it is clear that for any two indices i, i', the output of $\mathcal{CP}$ on a message a_h from Π_i is identically distributed to its output given a_h from $\Pi_{i'}$, which implies the second property of Definition 5.

Next, we need to specify a decoding algorithm `dec` for the verifier, which allows it to extract the correct message a_h from the $\mathcal{CP}$ output, given the index i. Since the length of a_h in Π_i is fixed, `dec` simply truncates its input to the correct length, which clearly gives the correct result. □

4.2 C&P IP with LOVe for Arithmetic Circuit Satisfiability

In this section we show a C&P IP with LOVe for arithmetic circuit satisfiability over a large field that satisfies (1) completeness, (2) soundness, and (3) that all inputs to Send are indistinguishable from random. Thus, by Theorem 2 we conclude that our protocol is also zero knowledge and supports disjunctions. Let $C : \mathbb{F}^n \to \mathbb{F}$ be a circuit known to both parties consisting of Add and Mult gates. The prover P has input $\vec{w} \in \mathbb{F}^n$ and wants to prove that $C(\vec{w}) = 0$ to V.

We begin by defining two auxiliary instructions we use: (1) Fix, which allows P to fix a random commitment to a value of its choosing, and (2) Reveal, which opens a commitment to V and checks this was done properly using $\mathsf{AssertZero}$.

- $\mathsf{Fix}(x)$: On input x from P, output a commitment $\underline{x}$. This is implemented as:
  1. Run $\mathsf{Random}() \to \underline{r}$.
  2. Run $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(x - r) \to y$.
  3. Run $+(\underline{r}, y) \to \underline{x}$.
- $\mathsf{Reveal}(\underline{x}) \to x$: On input commitment $\underline{x}$, output x to V. This is implemented as:
  1. $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(x)$.
  2. $\mathsf{AssertZero}(\underline{x} - x)$.

Our protocol works as follows. The prover first provides $\vec{w}$ as its input, so the parties initially get commitments $\underline{w_1}, \ldots, \underline{w_n}$. The parties then execute the following steps to evaluate the circuit C, where we denote by C^* the set of multiplication gates in C.

1. For each gate in C, in topological order, proceed as follows:

   $\mathsf{Add}(\underline{x}, \underline{y})$: Output $+(\underline{x}, \underline{y})$.

   $\mathsf{Mult}(\underline{x}, \underline{y})$: Run $\mathsf{Random}() \to \underline{a}$, $\mathsf{Fix}(xy) \to \underline{z}$, and $\mathsf{Fix}(ay) \to \underline{c}$. Output $\underline{z}$, and store the commitments $\underline{a}$ and $\underline{c}$.
2. Run $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(e)$, where $e \in_R \mathbb{F}$.
3. For $i \in [|C^*|]$:
   - Let $\underline{x_i}$ and $\underline{y_i}$ denote the inputs and $\underline{z_i}$, $\underline{a_i}$ and $\underline{c_i}$ denote the outputs and stored values in the i-th call to Mult.
   - Run $\mathsf{Reveal}(\underline{\varepsilon_i})$, where $\underline{\varepsilon_i} = e\underline{x_i} - \underline{a_i}$.
   - Run $\mathsf{AssertZero}(e\underline{z_i} - \underline{c_i} - \varepsilon_i \underline{y_i})$.
4. Run $\mathsf{AssertZero}(\underline{z_{\mathsf{out}}})$, where $\underline{z_{\mathsf{out}}}$ is the commitment to the output of the circuit.

Theorem 3. *Let $\mathcal{R}$ be a relation that can be represented by an arithmetic circuit C over field $\mathbb{F}$ such that $\mathcal{R}(\vec{x}, \vec{w}) = 1 \Leftrightarrow C(\vec{w}) = 0$. Then the above protocol is a $\mathsf{C\&P}$ IP with LOVe with respect to field $\mathbb{F}$ and relation $\mathcal{R}$ such that (1) completeness holds, (2) soundness holds with soundness error $1/|\mathbb{F}|$, and (3) all inputs to Send are perfectly indistinguishable from random.*

Proof. Completeness is immediate. For soundness, consider a malicious prover P^*, who at the beginning of the protocol chooses all of the randomness used in Random instructions. In the remainder of the protocol, P^* is only able to cheat in the Mult operation (since Reveal guarantees that ε_i is sent correctly). Consider a multiplication gate where P^*, i.e., instead of sending $\delta = xy - r$ in the Fix operation (for the random value r), it sent $\delta = xy + \Delta - r$, for some $\Delta \neq 0$. Similarly, let $\delta' = ay + \Delta' - r'$ be the value sent during the second Fix operation for that gate. We then have $z = xy + \Delta$ and $c = ay + \Delta'$.

The check for this gate passes only if $ez - c - \varepsilon y = 0$, which gives

$$e(xy + \Delta) - (ay + \Delta') - (ex - a)y = 0$$

which only holds if $e\Delta = \Delta'$. This occurs with probability $1/|\mathbb{F}|$, over the random choice of e. Since an incorrect statement must have at least one multiplication gate where the prover cheated, we obtain an overall soundness error of $1/|\mathbb{F}|$.

Finally, we prove that all inputs to Send are perfectly indistinguishable from random. Clearly the Send call in Step 2 satisfies this requirement. Likewise, the Send calls in Fix and Step 3 satisfy this requirement as well since we are masking by a uniformly random value which is not re-used. □

4.3 Disjunctions with C&P IPs with LOVe

Since our basic construction is an equisimulatable IP with LOVe, we can directly apply the stacking transformation to prove that at least one of several circuits $C_1, \ldots, C_m$ is satisfied. Using the recursive technique from § 3.3, we can even take this further and perform complex proofs about programs containing an arbitrary number of nested disjunctions. In this section, to see how a stacked IP with LOVe looks at a more concrete level, we give a brief overview of how a disjunctive proof looks in the C&P syntax.

Suppose that we have m C&P protocols $\Pi_1, \ldots, \Pi_m$, for proving properties $\mathcal{P}_1, \ldots, \mathcal{P}_m$ about their inputs, and all these protocols have uniform transcripts. For simplicity, let's assume that each Π_i only contains a single AssertZero statement, which verifies correctness of the entire proof.[3]

In the disjunctive proof from § 3.2, the prover runs the protocol Π_{i^*} corresponding to the true statement $\mathcal{P}_{i^*}$, receiving random challenges from the verifier. To hide which branch is being executed, the prover pads its messages to the maximum length of the message for that round in any Π_i, while the verifier's challenges are padded similarly. After running this, the prover is left with a commitment $\underline{z_{i^*}}$ corresponding to the true branch, which should be zero; however, the prover can *also* compute values z_i, for $i \neq i^*$, by following the computations across all the other branches using the verifier's challenges. Of course, these z_i values will most likely be non-zero. Meanwhile, the verifier can compute all of the commitments $\underline{z_1}, \ldots, \underline{z_m}$, it just does not know which is the correct one.

To complete the proof, the parties run a standard OR proof [CDS94] where the prover shows that at least one of the z_i commitments contains zero. Note that the size of this OR proof is independent of the size of any of the protocols Π_i. For completeness, we present this proof in Figure 2 using the C&P syntax. The idea is that the prover generates $m-1$ random challenges, for indices $i \neq i^*$, and crafts the messages d_i so that the verifier's check will always pass for these, even when $d_i \neq 0$. The verifier's random choice of e ensures that there is at least one index where P did not know the challenge, which must contain a commitment to zero.

OR Proof Construction of [CDS94]

Let $\underline{x_1}, \ldots, \underline{x_m}$ be the input commitments, where the prover knows an index j such that $x_j = 0$.

1. Run Random() to get random commitments $\underline{r_1}, \ldots, \underline{r_m}$.
2. P samples $e_i \in_R \mathbb{F}$, for all $i \neq j$.
3. P defines $d_i = r_i - x_i/e_i$, for $i \neq j$, and $d_j = r_j$, and sends $d_1, \ldots, d_m$ to V.
4. Let $\underline{z_i} = \underline{r_i} - d_i$, for $i \in [m]$.
   (Note that $z_j = 0$, while $z_i = x_i/e_i$ for all $i \neq j$)
5. Run $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(e \leftarrow \mathbb{F})$.
6. P computes $e_j = e - \sum_{i \neq j} e_i$.
7. Run $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(e_i)$, for $i = 1, \ldots, m-1$.
8. Run $\mathsf{AssertZero}(\underline{x_i} - e_i\underline{z_i})$, for $i = 1, \ldots, m$.

Figure 2: OR proof for showing that one out of m commitments contains zero.

4.4 Zero Knowledge from IP with LOVe and VOLE

As the final step, we show that we can convert any C&P IP with LOVe into a zero-knowledge proof, using vector oblivious linear evaluation (VOLE). We use a relaxed form of random VOLE functionality, in Figure 3, which picks random samples (r_i, τ_i), (α, β_i) such that $\tau_i = r_i\alpha_i + \beta_i$, but allows corrupt parties to choose their own randomness. This models existing random VOLE protocols based on the LPN assumption [BCGI18,

[3] Over a large field, AssertZero's can anyway be compressed down to just one, by checking a random linear combination chosen by the verifier.

BCG+19b], which can generate a large, length n VOLE with communication that is almost independent of n.

Commitments with MACs. We replace the abstract commitment notation $\underline{x}$ with one realized using information-theoretic MACs from VOLE. We write $\underline{x}$ to denote that the prover holds $x, \tau_x \in \mathbb{F}$, while the verifier holds β_x and the fixed MAC key $\alpha \in \mathbb{F}$. To open a commitment to x, the prover sends x, τ_x and the verifier checks that $\tau_x = x\alpha + \beta_x$. It is easy to see that cheating in an opening requires guessing the random MAC key α, so happens with probability $1/|\mathbb{F}|$.

As before, we overload the $+$ and $\cdot$ operators to denote addition and multiplication-by-constant, respectively, which can be done with just local computation due to linearity of the commitments.

The transformation. Given the linearly homomorphic commitment scheme based on VOLE, obtaining a ZK proof is relatively straightforward. First, the prover commits to its inputs, by sending each input masked with a random VOLE commitment, and then the parties run a coin-tossing functionality, $\mathcal{F}_{\mathsf{coin}}$ (Figure 4), to sample the random identifier given to the two parties. The parties then run the C&P protocol as usual, with the difference that each AssertZero is performed by having the prover send the tag τ_x, and the verifier check that $\tau_x = \beta_x$ (and hence, $x = 0$).

Functionality $\mathcal{F}^n_{\mathsf{VOLE}}$

The functionality interacts with a sender S, receiver R, and adversary $\mathcal{A}$.

On input $\alpha \in \mathbb{F}$ from R, the functionality does the following:

- Sample $r_i, \beta_i \leftarrow \mathbb{F}$, for $i = 1, \ldots, n$, and set $\tau_i = r_i\alpha + \beta_i$.
  - *If S is corrupted:* receive r_i, τ_i from $\mathcal{A}$ and recompute $\beta_i = \tau_i - r_i\alpha$.
  - *If R is corrupted:* receive β_i from $\mathcal{A}$ and recompute $\tau_i = r_i\alpha + \beta_i$.
- Output $(r_1, \tau_1, \ldots, r_n, \tau_n)$ to S and $(\beta_1, \ldots, \beta_n)$ to R.

Figure 3: Ideal functionality for vector oblivious linear evaluation over $\mathbb{F}$.

Functionality $\mathcal{F}_{\mathsf{coin}}$

On input λ, the functionality samples $r \leftarrow \{0,1\}^\lambda$ and sends r to both parties.

Figure 4: Ideal functionality for coin tossing.

Theorem 4. *Suppose $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$ is a C&P IP with LOVe for property $\mathcal{P}$, satisfying completeness, soudness error ε and honest-verifier zero-knowledge. Then, $\Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}$ is a zero-knowledge proof for the NP relation defined by $\mathcal{P}$, with soundness error $\varepsilon + 1/|\mathbb{F}|$. Furthermore, if $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$ is a proof of knowledge, then so is $\Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}$.*

Proof. Completeness follows immediately from the completeness of $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$, and the linear properties of the VOLE-based commitments. Regarding soundness, consider a cheating prover P^* who manages to prove an incorrect statement in $\Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}$ with probability ε^*. We can turn P^* into an adversary for the soundness of $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$. The only difference in the two executions, is that in $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$ there is no possibility to cheat during an AssertZero, whereas in $\Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}$, P^* may force an incorrect assertion with probability $1/|\mathbb{F}|$. We therefore have,

$$|\Pr[\mathsf{V} \text{ outputs 1 in } \Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}] - \Pr[\mathsf{V} \text{ outputs 1 in } \Pi^{\mathsf{LOVe}}_{\mathsf{CP}}]| \leq 1/|\mathbb{F}|.$$

Since $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$ has soundness error ε, we conclude that $\Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}$ has soundness error at most $\varepsilon + 1/|\mathbb{F}|$.

Finally, for the proof of knowledge property, observe that in $\Pi^{\mathsf{VOLE}}_{\mathsf{ZK}}$, the extractor can simulate $\mathcal{F}_{\mathsf{VOLE}}$, and so extract all of the random values r_i. This corresponds to knowing the entire proof string π in the underlying IP with LOVe, so we can run the same extraction algorithm as for $\Pi^{\mathsf{LOVe}}_{\mathsf{CP}}$. □

Transformation $\Pi_{\mathsf{CP}}^{\mathsf{LOVe}} \to \Pi_{\mathsf{ZK}}^{\mathsf{VOLE}}$

Let t be the number of Random instructions in $\Pi_{\mathsf{CP}}^{\mathsf{LOVe}}$, the underlying C&P IP with LOVe.

Input phase: The prover has input the witness $w_1, \ldots, w_n$:

1. The parties call $\mathcal{F}_{\mathsf{VOLE}}^{t+n}$, to obtain the random commitments $\underline{r_1}, \ldots, \underline{r_{t+n}}$.
2. P sends $d_i = r_i - w_{i-t}$ to V, for $i = t+1, \ldots, t+n$.
3. Both parties compute the commitment $\underline{w_{i-t}} = \underline{r_i} - d_i$, for $i = t+1, \ldots, t+n$.
4. The parties call $\mathcal{F}_{\mathsf{coin}}$ to obtain the identifier $\mathsf{id}_\pi \in \{0,1\}^\lambda$.

Protocol: The parties execute the instructions in the protocol as follows:

1. Random(): For the i-th such instruction, output the commitment $\underline{r_i}$.
2. Send: The parties send the appropriate message according to the protocol.
3. $+, \cdot$: The parties run add or multiply-by-constant with corresponding commitments.
4. AssertZero($\underline{x}$): P sends τ_x to V, who checks that $\tau_x = \beta_x$.

Output phase: The verifier outputs 1 if all AssertZero instructions passed.

Figure 5: Zero-knowledge proof from VOLE and C&P IP with LOVe.

5 Streaming and Non-Interactive Proofs via Fiat-Shamir

We now show how to modify our previous constructions for arithmetic circuit satisfiability and disjunctions to support streaming, and also be non-interactive via a variant of the Fiat-Shamir transform [FS87].

5.1 Streaming Interactive Proofs

We use the term *streaming* to refer to a protocol where both the prover and verifier algorithms can be run using only a constant amount of memory, independent of the size of the statement and witness. For disjunctive proofs, we relax this to allow $O(m)$ memory, where m is the maximum number of branches in any disjunction. Note that when looking at a C&P IP with LOVe, as well as requiring small memory for P and V, we need that the linear oracle queries can be performed with small memory. It is enough to require that P can compute the result of each oracle query incrementally during the protocol, and with constant memory; when translating the IP with LOVe into a zero-knowledge proof based on VOLE (§ 4.4), this ensures that the resulting protocol also has constant memory, since each AssertZero can be checked on-the-fly.

Recall that in our protocol for circuit satisfiability (§4.2), the multiplication gates are all verified in a batch at the end of the computation. This requires storing all commitments created during each multiplication in memory, leading to a memory cost that is linear in the circuit size.

We can easily avoid this by checking multiplications on-the-fly using an independent random challenge from V for each multiplication. Below, we describe this variant of the multiplication subprotocol. To help with removing interaction, we have also expanded the description of the protocol, by replacing Fix and Reveal instructions with messages from the prover and calls to AssertZero. Note that this is purely a syntactic change, since this is what is done anyway when compiling the C&P protocol to an IP with LOVe as in §4.1.

Streaming Mult$(\underline{x_i}, \underline{y_i})$**:** To evaluate the i-th multiplication gate:

1. Run Random() to get $\underline{a_i}, \underline{r_i}, \underline{r'_i}$.
2. Run $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(\delta_i = r_i - x_i y_i)$ and $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(\delta'_i = r'_i - a_i y_i)$.
3. Let $\underline{z_i} = \underline{r_i} - \delta_i$ and $\underline{c_i} = \underline{r'_i} - \delta'_i$.
4. Run $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(e_i \leftarrow \mathbb{F})$.

5. Run $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(\varepsilon_i = e_i x_i - a_i)$.
6. Run $\mathsf{AssertZero}(e_i \underline{x_i} - \underline{a_i} - \varepsilon_i)$.
7. Run $\mathsf{AssertZero}(e_i \underline{z_i} - \underline{c_i} - \varepsilon_i \underline{y_i})$.

Note that the first AssertZero call simply verifies that ε_i was revealed correctly, while the second call verifies the result of the multiplication.

To analyze the soundness of this, consider a cheating prover who sends values

$$\delta_i = r_i - x_i y_i - \Delta_i, \qquad \delta_i' = r_i' - a_i y_i - \Delta_i' \tag{1}$$

(Note that we can ignore the case of a prover who sends $\varepsilon_i \neq e_i x_i - a_i$, since this will always be caught by AssertZero.) The second assertion only passes if

$$\begin{aligned} & e_i z_i - c_i - \varepsilon_i y_i = 0 \\ \Leftrightarrow\ & e_i(x_i y_i + \Delta_i) - (a_i y_i + \Delta_i') - (e_i x_i - a_i) y_i = 0 \\ & \Leftrightarrow e_i \Delta_i = \Delta_i'. \end{aligned}$$

Since Δ_i, Δ_i' are fixed before e_i is chosen, if $\Delta_i \neq 0$ then verification succeeds with probability at most $1/|\mathbb{F}|$.

5.2 Removing Interaction From the Verifier

We now show how to transform the above protocol to remove all interaction from the verifier (apart from calls to AssertZero), using a variant of the Fiat-Shamir transform. This transformation is similar to that used for interactive oracle proofs [BCS16], where bounds were also given on the soundness of the resulting non-interactive proofs. However, these bounds are not suitable for us, since in the worst case the soundness can degrade *exponentially* with the number of rounds. Therefore, we also give concrete, tight bounds on the soundness of the transformation when applied to the streaming protocol above (which has $O(|C|)$ rounds).

In Figure 6 we define the transformation we use, which can be applied to any IP with LOVe. We make use of the random identifier id_π as input to the first call to the random oracle; this ensures that P is committed to π before it can obtain random challenges, and is crucial for security.

Transformation $\mathsf{FS}(\Pi)$

Let x be the NP statement being proven, and Π the underlying IP with LOVe.
Let $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$ be a random oracle.

- The prover chooses the proof string π, as in Π.
- Both parties receive the random public identifier $\mathsf{id}_\pi \in \{0,1\}^\lambda$.
- P computes the first message a_1, and challenge $e_1 = \mathsf{H}(x \| \mathsf{id}_\pi \| a_1)$.
- For each round $i > 1$, P computes its message a_i and the challenge

$$e_i = \mathsf{H}(a_i \| e_{i-1})$$

- P sends the messages $(a_1, \ldots, a_t)$.
- V recomputes all the challenges e_i, makes the same queries to the linear verification oracle as in the original protocol Π, and accepts if they all accept.

Figure 6: The transformation to non-interactive proof with LOVe.

Theorem 5. *Let Π_{Circuit} be the protocol for arithmetic circuit satisfiability from § 5.1, with soundness error $1/|\mathbb{F}|$. Then, the compiled protocol $\mathsf{FS}(\Pi_{\mathsf{Circuit}})$ is a non-interactive IP with LOVe with soundness error at most*

$$\frac{3Q}{|\mathbb{F}|} + \frac{Q}{2^\lambda}$$

where Q is the number of random oracle queries made by a malicious prover.

See § B for the proof.

5.3 Removing Interaction From AssertZero

We have just shown how to remove interaction from the verifier in our streamable IP with LOVe for circuit satisfiability. However, when translating this into a zero knowledge proof using, e.g., VOLE as in § 4.4, we still have to deal with all the AssertZero statements in a streaming-friendly manner. Ideally, we would like to batch all AssertZero's into one check by taking random linear combinations, however, this requires storing all intermediate values in memory. We now show how to avoid this, by carrying out the AssertZero statements both non-interactively and with constant memory.

Let Π_{NI} be any non-interactive C&P protocol, namely, one which does not contain any messages sent by the verifier, but has an arbitrary number of AssertZero queries. In Figure 7, we show how to transform Π_{NI} to have just one AssertZero. The idea is that, instead of taking a combination of all AssertZero's at the end, we compute this combination in an incremental manner. At each AssertZero on input $\underline{\gamma}$, we take a random challenge e (from a random oracle) and add $e \cdot \underline{\gamma}$ to a running state $\underline{z}$. At the end of the computation, to verify that all the γ values were zero, we simply run AssertZero on $\underline{z}$.

Transformation $\mathsf{Stream}(\Pi_{\mathsf{NI}})$

Let $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$ be a random oracle.

1. P first commits to its inputs as in Π_{NI}, and both parties receive the random identifier $\mathsf{id}_\pi \in \{0,1\}^\lambda$.
2. For the i-th $\mathsf{AssertZero}(\underline{\gamma_i})$ instruction in Π_{NI}, if $i = 1$:
   - Let $e_1 = \mathsf{H}(\mathsf{id}_\pi \| a_1)$ (where a_1 contains all messages sent until now)
   - Compute $\underline{z} = e_1 \underline{\gamma_1}$.

   Otherwise, if $i > 1$:
   - Let a_{j_i} be the set of messages sent in $\mathsf{Send}_{\mathsf{P}\to\mathsf{V}}$ instructions since the previous AssertZero.
   - Let $e_i = \mathsf{H}(e_{i-1} \| a_{j_i})$.
   - Update $\underline{z} = \underline{z} + e_i \underline{\gamma_i}$.
3. All other instructions are kept the same.
4. At the end of the program, run $\mathsf{AssertZero}(\underline{z})$.

Figure 7: The transformation to a streamable non-interactive IP with LOVe.

Theorem 6. *Let Π_{NI} be a non-interactive C&P protocol that is complete and has soundness error ε. Then, the compiled protocol $\mathsf{Stream}(\Pi_{\mathsf{NI}})$ is a non-interactive IP with LOVe with soundness error*

$$\varepsilon + \frac{2Q+1}{|\mathbb{F}|} + \frac{Q}{2^\lambda}$$

where Q is the number of RO queries made by a malicious prover to H.

See § C for the proof.

5.4 Streaming with Disjunctions

Consider protocols $\Pi_1, \ldots, \Pi_m$ for relations $\mathcal{R}_1, \ldots, \mathcal{R}_m$, which have each been compiled to be streamable, non-interactive IPs with LOVe with one query each, via our previous transformations. We can apply the OR transformation (Theorem 1) to obtain a protocol $\Pi_{\mathtt{OR}}$ for proving that at least one of the $\mathcal{R}_i$ is satisfied.

Since the original protocols were non-interactive and perform only one query, we can actually simplify the transformation from Figure 1, since there is no need for the verifier to send the random ρ_i challenges, which are used to compress the number of queries. As a result, the final protocol Π_{OR} essentially consists of running Π_i for the true branch, P sending the d_i-values, followed by one random challenge f from the verifier, then a message containing $f_1, \ldots, f_m$ from the prover and m linear oracle queries.

We can now apply the Fiat-Shamir transformation in Figure 6 to this protocol, to squash it back to being non-interactive. Since there is only one challenge from the verifier, it is easy to see that in this case, the soundness error does not increase by more than an additive factor of $Q/|\mathbb{F}|$, for Q random oracle queries. Regarding memory usage, while each branch can be evaluated in constant memory, the prover has to keep track of $O(m)$ values to be checked at the end, giving $O(m)$ memory cost for m branches.

Nested disjunctions and the logarithmic proofs. When repeatedly nesting disjunctions to obtain more complex statements, or logarithmic sized OR proofs as in § 3.3, we can apply the same transformations to obtain non-interactive and streamable proofs. In this case, the exact soundness error depends on the structure of the program and the sequence of the transformations. We leave it as future work to obtain a general method for analyzing this type of construction.

6 Implementation and Evaluation

We implemented the non-interactive variant of the online portion of Mac'n'Cheese in the Rust programming language. Our implementation is agnostic to the specific choice of finite field (so long as it is large enough for the desired security level). Network communication between the prover and verifier is protected by TLS, and we use the BLAKE3 hash function to generate challenges for the non-interactive protocol. Currently, our implementation is single-threaded.

VOLE. Our implementation supports pluggable VOLE backends. The backend that we use, at present, is a "dummy" backend, which (insecurely) generates random MACs by using a pre-shared seed with a PRNG. This simulates the preprocessing scenario in which Random MACs have been pre-generated.

Streaming. To facilitate streaming, our implementation does not view its input as an explicit circuit graph. Instead, the proof statement is lazily built-up by a series of function invocations. As a result, we get reduced memory consumption (for free) as temporary values get automatically freed when they are no longer in-scope.

In order to stream a two-way disjunction, both branches of the disjunction must be interleaved (otherwise the prover would be forced to either buffer the entirety of a branch, or reveal which branch is 'true'). To achieve this interleaving, we leverage Rust's `async/await` functionality to (cheaply) concurrently execute both branches.

Optimizations. We have implemented constant-folding operations for MACs (akin to Skipgate [SRH+17]). In particular, we track (on both the prover and the verifier) whether each MAC corresponds to a public/constant value, and perform peephole optimizations on these values. For example, we lower the product of a public value with a private value into a constant multiply to avoid the communication and computation overhead which would be incurred if we ran the full multiplication protocol. We also compute multiplications on purely public values without any communication and lower a public zero multiplied by a private value into a public zero. These peephole optimizations enable users of the Mac'n'Cheese implementation to write efficiently-executable proof statements without needing a value abstraction other than the MAC.

6.1 Evaluation

We benchmarked our implementation for prime field $\mathbb{F}_p$ for two choices of p: $2^{61}-1$ and $2^{127}-1$. We include $p = 2^{61}-1$ to compare most accurately with Wolverine, despite the fact that in the non-interactive setting

n	VOLEs (millions)	Running time (seconds)	Max mem (MB)	Total comm	Running time (seconds)	Max mem (MB)	Total comm
		$p = 2^{61} - 1$			$p = 2^{127} - 1$		
64	.8	.6	14.1	6.1 MB	1.1	14.2	12.1 MB
96	2.7	1.8	14.1	20.4 MB	3.8	14.2	40.8 MB
128	6.3	4.7	14.1	48.1 MB	9.3	14.2	96.5 MB
192	21.3	14.4	14.1	162.6 MB	32.4	16.8	325.1 MB
256	50.4	41.2	16.2	385.0 MB	79.3	20.3	770.0 MB
384	170.1	126.5	24.8	1.3 GB	276.2	40.0	2.5 GB
512	403.2	343.7	35.3	3.0 GB	649.3	60.2	6.0 GB
768	1360.1	1171.7	78.7	10.1 GB	1837.4	122.2	20.3 GB
1024	3223.3	2745.6	120.2	24.0 GB	3608.4	198.2	48.0 GB

Table 1: Performance results for the matrix multiplication experiment. The Max mem columns correspond to the maximum memory used between the prover and verifier. The running time and total communication does *not* account for the communication required by VOLE, which is roughly 130 ns and 1.32 bits per VOLE for $p = 2^{61} - 1$, respectively [WYKW20, Table 4].

we only get 61-bits of security. However, we note that this attack must necessarily be done *online* and thus may be mitigated to a certain extent.

All benchmarks were run on a machine with 40 Intel Xeon Silver 4114 cores operating at 2.20 GHz. All experiments were run locally, and thus bandwidth was *not* the limiting factor. To account for this unrealistic deployment scenario, we provide the total communication required in our experiments, allowing one to compute a rough estimation of the performance cost when the bandwidth is restricted. Also, as noted above, these results *do not* include the cost of VOLE. We are in the process of integrating the VOLE protocol recently presented by Weng et al. [WYKW20], but do not believe this will have a large impact on the overall running time and communication cost given that a single VOLE for $p = 2^{61} - 1$ can be generated in 130 ns at a communication cost of 1.32 bits [WYKW20, Table 4].

Mac′n′Cheese achieves a per multiplication cost of 2.5 μs for $p = 2^{61} - 1$ and a per multiplication cost of 4 μs for $p = 2^{127} - 1$. This is comparable to the 1.6 μs per multiplication achieved by Wolverine (which uses $p = 2^{61} - 1$). We believe the discrepancy can be accounted for by the fact that the results presented for Wolverine were over five threads, whereas Mac′n′Cheese is currently single-threaded.

To further explore the performance of Mac′n′Cheese, we ran two experiments, one to demonstrate the overall performance of Mac′n′Cheese when disjunctions do not come into play, and one to demonstrate the performance improvement when disjunctions are introduced. We discuss each in turn below.

Experiment 1: Matrix multiplication. In this experiment, we consider the setting where the prover wants to convince the verifier that it knows two private n-by-n matrices that multiply to some public matrix using the naive $O(n^3)$ multiplication algorithm. While a contrived example with seemingly no real-world use case, this example has been explored in other works [WYKW20, ZXZS20, BCTV14] as a means for benchmarking arithmetic-focused ZK schemes.

We ran Mac′n′Cheese on this proof statement, varying n from 64 to 1024; see Table 1 for our results. Of particular note is the small amount of memory required—even for a large 1024-by-1024 matrix, both the prover and verifier require under 200 MB of memory. However, note again that our results do *not* include the memory cost of VOLE, which could potentially contribute another 160 MB of memory to our results.

Experiment 2: Disjunctions. In this experiment, we explore the effect our nested disjunction optimization has on the overall running time and communication cost. We do so by combining the above matrix multiplication proof statement with disjunctions by considering the (admittedly contrived) scenario of a prover wanting to prove that it knows two n-by-n matrices such that their multiplication equals one of t public matrices, where we vary t from two to sixteen.

n	t	VOLEs (millions)	Running time (seconds)	P mem (MB)	Total comm (MB)	Running time (seconds)	Max mem (MB)	Total comm (MB)
			$p = 2^{61} - 1$			$p = 2^{127} - 1$		
64	2	1.6	1.3	14.1	12.1	2.0	14.1	24.3
		.8	.8	14.1	6.1	1.7	14.1	12.1
	4	3.2	2.8	14.1	24.3	4.5	14.1	48.5
		7.9	1.4	14.1	6.1	3.8	14.1	12.1
	8	6.4	5.6	14.1	48.5	9.3	16.1	97.0
		7.9	3.1	14.1	6.1	7.7	14.1	12.1
	16	12.7	11.9	14.1	97.0	19.2	14.1	194.0
		7.9	6.6	14.5	6.1	15.5	16.6	12.1
128	2	12.6	11.4	14.1	96.5	18.9	14.1	193.0
		6.3	7.2	14.1	48.3	16.3	14.1	96.5
	4	25.3	23.9	14.1	193.0	39.2	14.1	386.0
		6.3	12.1	14.8	48.3	32.6	17.3	96.5
	8	50.6	49.2	16.1	386.0	80.5	14.1	772.0
		6.3	24.9	17.8	48.3	65.6	22.3	96.5
	16	101.2	103.6	14.1	772.0	168.6	14.1	1544.0
		6.3	53.1	24.4	48.3	133.7	34.4	96.5
256	2	100.9	102.4	16.3	770.0	166.4	21.3	1540.0
		50.5	64.0	24.8	385.0	142.1	27.4	770.0
	4	201.9	209.3	16.2	1540.0	342.1	20.3	3080.0
		50.5	108.9	29.3	385.0	282.5	37.4	770.0
	8	403.7	435.9	18.1	3080.0	705.0	24.7	6160.0
		50.5	223.9	39.3	385.0	568.0	57.3	770.0
	16	807.4	813.2	16.4	6160.0	1396.6	27.6	12320.0
		50.5	450.8	63.2	385.0	1137.3	101.3	770.0

Table 2: Performance results for the disjunction experiment. t denotes the number of matrix multiplications we are OR-ing over. For any given disjunction number, the top row shows results for the naive approach and the bottom row shows results when using our nested disjunction optimization. As before, the running time and total communication does *not* account for the cost of VOLE.

Table 2 presents our results. We can immediately see the benefit of nested disjunctions both in terms of running time and overall communication. For example, for 256-by-256 matrices with sixteen disjunctions, we get the expected 16× communications improvement alongside a 1.2–1.8× running time improvement. Note that the reason the running time does not scale with communication is because the prover and verifier still need to compute all the branches and hence still need to pay the computation cost required there. However, recall that our experiments are run locally and hence are not communication-bound; in bandwidth-constrained environments the savings in communication will result in a potentially much larger running time improvement. We also note that concurrent disjunctions *increase* the memory usage of both the prover and verifier, at least for larger matrix sizes. This is due to the fact that both the prover and verifier are processing t matrix multiplications at once.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA). Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS 2017*. ACM Press, October / November 2017.

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In *CRYPTO 2019, Part III*, LNCS. Springer, Heidelberg, August 2019.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO 2019, Part III*, LNCS. Springer, Heidelberg, August 2019.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013, Part II*, LNCS. Springer, Heidelberg, August 2013.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*. ACM Press, November 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, LNCS. Springer, Heidelberg, August 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM CCS 2018*. ACM Press, October 2018.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC 2016-B, Part II*, LNCS. Springer, Heidelberg, October / November 2016.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succint non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security Symposium*. USENIX, 2014.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO'91*, LNCS. Springer, Heidelberg, August 1992.

[BG90] Mihir Bellare and Shafi Goldwasser. New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. In *CRYPTO'89*, LNCS. Springer, Heidelberg, August 1990.

[CD97] Ronald Cramer and Ivan Damgård. Linear zero-knowledge - a note on efficient zero-knowledge proofs and arguments. In *29th ACM STOC*. ACM Press, May 1997.

[CD98] Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In *CRYPTO'98*, LNCS. Springer, Heidelberg, August 1998.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO'94*, LNCS. Springer, Heidelberg, August 1994.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT 2015, Part II*, LNCS. Springer, Heidelberg, April 2015.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, LNCS. Springer, Heidelberg, August 1987.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*. ACM Press, May 1987.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *EUROCRYPT 2020, Part III*, LNCS. Springer, Heidelberg, May 2020.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM CCS 2013*. ACM Press, November 2013.

[Kol18] Vladimir Kolesnikov. Free `IF`: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In *ASIACRYPT 2018, Part III*, LNCS. Springer, Heidelberg, December 2018.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS 2016*. ACM Press, October 2016.

[SRH+17] Ebrahim M. Songhori, M. Sadegh Riazi, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. ARM2GC: Simple and efficient garbled circuit framework by skipping. Cryptology ePrint Archive, Report 2017/1157, 2017. `https://eprint.iacr.org/2017/1157`.

[WYKW20] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925, 2020. `https://eprint.iacr.org/2020/925`.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT 2015, Part II*, LNCS. Springer, Heidelberg, April 2015.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020.

A More Details on Stacking

In this section we describe how to construct *threshold stacking* and give a full construction and proof for the *recursive stacking* approach.

A.1 Threshold Stacking

We now describe how to construct a protocol for a higher threshold of true statements. Let $S \subset [m]$ be the subset of branches that P has a witness for and where $|S| = r$. We make the following modifications to $\Pi_{\mathtt{OR}}$:

1. In the original protocol, P in Step 1 creates one string π that is getting sent to the oracle for queries by V. This does not work in case of a general threshold r. One solution would be to concatenate m strings $\pi_1, \ldots, \pi_m$ and use the respective substring for the respective oracle queries, but this will increase the oracle length by a factor of m. As the oracle length might have an impact on efficiency, we want to avoid this and will instead use a solution that only increases it by a factor r.

   We can write each π_i as having length $\ell = \max_{i\in[m]} \ell_i$ by padding it with 0s. The honest P will only have $\pi_i, i \in S$ available. We can, for each of the ℓ elements of each π_i, define a polynomial $\Pi_{j\in[\ell]}$ of degree $r-1$ such that $\Pi_j(i) = \pi_i[j]$ for $i \in S$. Then, we can place the r coefficients of each Π_j in π, which therefore becomes $\pi = \Pi_1 | \ldots | \Pi_\ell | r_1, \ldots, r_m$.

   This requires us to modify how queries to the oracle will be made, but the change is simple. For each query $\langle \pi_i, \overline{z}_i \rangle =_? y_i$ we instead create the query $\langle \pi, \tilde{z}_i \rangle =_? y_i$ where $\tilde{z}_i \in \mathbb{F}^{r\ell}$. We will let the first r elements of $\tilde{z}_i$ consist of $[\overline{z}_i[1], i \cdot \overline{z}_i[1], \ldots, i^{r-1}\overline{z}_i[1]]$ i.e. of the first element of $\overline{z}_i$ multiplied with powers of i. Then, the next r elements will consist of $\overline{z}_i[2]$ multiplied with the same pattern of powers of i etc. The reason for this is that the powers of i will later implicitly compute $\Pi_j(i)$ and therefore, for the true branches, will reconstruct $\pi_i[j]$, such that the same inner product as in a real protocol will be computed.

2. In Step 2, we need to communicate a different message $a_{h,i}$ for each $i \in [m]$. This is because we want to execute multiple branches honestly but do not want to reveal which it is that we use. Our solution is as follows: For each round $h \in [t]$ where P sends messages we let $A_h(X)$ be the polynomial of degree $r-1$ such that $\forall s \in S: \ A_h(s) = a_h^s$. That means that we create a polynomial that agrees with the correct messages for all valid branches. V will later use the interpolation of each $A_h(X)$ at all $i \in [m]$ to determine the queries that it will issue[4].

3. One can see Step 4 of $\Pi_{\mathtt{OR}}$ as P choosing $m-1$ secret shares, where the choice of f later determines the secret and thus the last share. Here, knowing these shares in advance allows P to "lie" in $m-1$ proof instances. For the threshold setting, we can instead let P choose $m-r$ shares $f_i, i \notin S$ in Step 4 and compute the d_i-values accordingly (use e.g. Shamir's Secret Sharing). Later in Step 5 V sends r challenges $f_1, \ldots, f_r$. P will compute the remaining r shares of $f_i, i \in S$ such that the f_i form a packed secret sharing where $f_1, \ldots, f_r$ are indeed the secrets. It then sends all these shares to V who indeed checks that they form a correct secret-sharing of the demanded secret with the correct threshold $m-r$.

We leave a full discussion of this protocol for the full version of this paper.

A.2 Formalizing Recursive Stacking

Constructing the basic protocol. We construct the base protocol $\Pi_{\mathtt{A-OR}}$ that shows the relation $\mathcal{R}_{\mathtt{OR}}$ for two branches in Figure 8. The protocol $\Pi_{\mathtt{A-OR}}$, on a high level, works as follows:

1. First, the prover adds random multiplication triples to the string π. This will later allow it to prove that certain multiplicative relations hold between elements that are in the span of π. We do not care

[4]For this to work the Equisimulatability property of the individual protocols is not enough. It can be made to work though as messages from P have a pseudorandom distribution over $\mathbb{F}$.

Protocol $\Pi_{\mathtt{A-OR}}$

Let Π_1, Π_2 be protocols such that each Π_i is t_i-round q_i-query equisimulatable public coin IP with LOVe over $\mathbb{F}$ for relation $\mathcal{R}_i$ with oracle length ℓ_i.

Both P and V have inputs x_1, x_2 where $x_i \in L(\mathcal{R}_i)$. P additionally has input w_{i^*} for (at least) one $i^* \in [2]$ such that $(x_{i^*}, w_{i^*}) \in \mathcal{R}_{i^*}$. We define $q := q_1 + q_2$, $\ell := \max\{\ell_1, \ell_2\}$, and $t := \max\{t_1, t_2\}$. Let $z_{k,i} = \overline{z}_{k,i} | \underbrace{0 \cdots 0}_{\ell - \ell_i \text{ times}}$.

1. P initially simulates Π_{i^*} on input (x_{i^*}, w_{i^*}) to obtain the string π_{i^*} and the state s_0^{P}. It then sets
$$\pi = \pi_{i^*} | \underbrace{0 \cdots 0}_{\ell - \ell_{i^*} \text{ times}} | a_1 \dots a_{\aleph} b_1 \dots b_{\aleph} c_1 \dots c_{\aleph}$$
where a_v, b_v are chosen uniformly at random and $c_v = a_v \cdot b_v$.
2. Define $s_0^{\mathsf{P}} := (x_{i^*}, w_{i^*}), s_{t_{i^*}}^{\mathsf{P}} := \perp$. For $h \in [t]$, P and V do the following:
   (a) Let $r_h^{\mathsf{V},i}$ be the length of the challenge that V would send for protocol Π_i in round h. V sets $r_h = \max\{r_{h,1}^{\mathsf{V}}, r_{h,2}^{\mathsf{V}}\}$, samples $e_h \leftarrow \mathbb{F}^{r_h}$ uniformly at random and then sends it to P.
   (b) P sets $(a_h, s_h^{\mathsf{P}}) \leftarrow \Pi_{i^*}(s_{h-1}^{\mathsf{P}}, e_h)$ where P only uses the first r_{h,i^*}^{P} elements of e_h as required by Π_{i^*}. It computes $c_h \leftarrow \mathcal{CP}(\hat{x}, h, i^*, a_h)$ and sends c_h to V.
3. V samples $2\aleph$ strings $\{\rho_{v,1}, \rho_{v,2}\}_{v \in [\aleph]}$ of length q_1, q_2 each uniformly at random from $\mathbb{F}$ and sends these to P.
4. P for $i \in [2]$ computes
$$\{(\overline{z}_{k,i}, y_{k,i})\}_{k \in [q_i]} \leftarrow \mathcal{Q}(x_i, \{e_h, \mathtt{dec}(\hat{x}, h, i, c_h)\}_{h \in [t_i]}).$$
Additionally, for $v \in [\aleph]$ it sets
$$\hat{z}_{v,i} = \sum\nolimits_{k \in [q_i]} \rho_{v,i}[k] z_{k,i} \quad \text{and} \quad \hat{y}_{v,i} = \sum\nolimits_{k \in [q_i]} \rho_{v,i}[k] y_{k,i}.$$
Then for $v \in [\aleph]$ P computes $\mu_{v,1} = \langle \pi, \hat{z}_{v,1} | 0 \dots 0 \rangle - \hat{y}_{v,i}$, $\mu_{v,2} = \langle \pi, \hat{z}_{v,i} | 0 \dots 0 \rangle - \hat{y}_{v,2}$, $d_{v,1} = \mu_1 - a_v$, $d_{v,2} = \mu_2 - b_v$ and sends $d_{v,1}, d_{v,2}$ to V.
5. Define $\overline{\delta}_i \in \mathbb{F}^{3\aleph}$ to be the vector that is 1 on the ith position and 0 everywhere else. Also, let γ_v be the vector that is $-d_{v,2}$ in the vth position, $-d_{v,1}$ in the $v + \aleph$th position, -1 in the $v + 2\aleph$th position and 0 everywhere else. Then V does the following:
   (a) Compute $\hat{z}_{v,1}, \hat{z}_{v,2}, \hat{y}_{v,1}, \hat{y}_{v,2}$ like P did in Step 4.
   (b) For each $v \in [\aleph]$ send the queries $(0 \dots 0 | \gamma_v, d_{v,1} \cdot d_{v,2}), (\hat{z}_{v,1} | -\overline{\delta}_v, d_{v,1} + \hat{y}_{v,1}), (\hat{z}_{v,2} | -\overline{\delta}_{v+\aleph}, d_{v,2} + \hat{y}_{v,2})$ to the oracle.

   V accepts if all queries are true, otherwise it rejects.

Figure 8: The protocol $\Pi_{\mathtt{A-OR}}$ for an OR-statement.

here how to show that P is actually doing this honestly, as this is covered in a different part of this work.

2. Then, the evaluation of both branches is done in parallel as in $\Pi_{\mathtt{OR}}$.
3. Now instead of performing the oracle queries separately, we let V choose random linear combinations of the oracle queries similar to the intuition outlined in Section 3.3.
4. Finally P sends to V material that allows V to test if for at least one of the two branches all the queries are correct by checking that the product of the μ_i is 0. V then performs these queries.

All queries that V will do are deterministic given the transcript. Furthermore, the additional messages that

are sent by both P and V appear uniformly random. This will later allow us to *recursively construct a proof for more than* 2 *branches.*

Theorem 7. *Let Π_1, Π_2 be protocols such that each Π_i is a t_i-round q_i-query equisimulatable Public Coin IP with LOVe over $\mathbb{F}$ for relation $\mathcal{R}_i$ with oracle length ℓ_i, message complexity α_i and soundness error ϵ_i. Furthermore, let $\aleph \in \mathbb{N}^+$ and assume that $a_v \cdot b_v = c_v$ holds for all $v \in [\aleph]$.*

Then the protocol $\Pi_{\mathtt{A-OR}}$ in Figure 8 is a Public Coin IP with LOVe for the relation $\mathcal{R}_{\mathtt{OR}}$ with

1. *round complexity $2 + \max\{t_1, t_2\}$;*
2. *oracle length $3\aleph + \max\{\ell_1, \ell_2\}$;*
3. *query complexity $3\aleph$;*
4. *message complexity $(2 + q_1 + q_2)\aleph + \max\{\alpha_i\}$; and*
5. *soundness error $\epsilon_1 + \epsilon_2 + \left(1 - \left(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}\right)^2\right)^{\aleph}$.*

Here, $(q_1 + q_2) \cdot \aleph$ field elements of the message complexity are just to send the pseudorandom vectors $\rho_{v,1}, \rho_{v,2}$ from V to P. In practice, these can either be sent as a random seed that will be chosen by V and expanded by P or can be obtained using the Fiat-Shamir Transform, thus reducing the message complexity to just an $2\aleph$ additive overhead over proving the longer branch. When $|\mathbb{F}|$ is exponential in the security parameter, then $\aleph$ can be set to 1, meaning that the overhead consists of 2 field elements.

Proof. Completeness. If $(x_{i^*}, w_{i^*}) \in \mathcal{R}_{i^*}$ and P has w_{i^*} then Steps 1-4 will run by the same argument as in the completeness proof for $\Pi_{\mathtt{OR}}$. All that remains to show is that the $3\aleph$ queries will all be accepting.

For the first query, we have for each $v \in [\aleph]$ that

$$\begin{aligned}
\langle \pi, 0 \dots 0 | \gamma_v \rangle &= -d_{v,2}a_v - d_{v,1}b_v - c_v \\
&= -b_v \cdot \mu_{v,2} + a_v b_v - a_v \cdot \mu_{v,2} + a_v b_v - c_v \\
&= -b_v \cdot \mu_{v,1} - a_v \cdot \mu_{v,2} + a_v b_v \\
&= \mu_{v,1} \cdot \mu_{v,2} - b_v \cdot \mu_{v,1} - a_v \mu_{v,2} + a_v b_v \\
&= (\mu_{v,1} - a_v) \cdot (\mu_{v,2} - b_v) \\
&= d_{v,1} d_{v,2}
\end{aligned}$$

which holds because $a_v \cdot b_v = c_v$ and because $\langle \pi_{i^*}, \overline{z}_{k,i^*} \rangle = y_{k,i^*}$ for the true branch. The latter implies that also $\langle \pi, \sum_{k \in [q_{i^*}]} \rho_{v,i^*}[k] \overline{z}_{k,i^*} \rangle = \sum_{k \in [q_{i^*}]} \rho_{v,i^*}[k] y_{k,i^*}$ and therefore $\mu_{v,1} \cdot \mu_{v,2} = 0$.

For the remaining two queries, we observe for the first one that

$$\begin{aligned}
\langle \pi, \hat{z}_{v,1} | - \overline{\delta}_v \rangle &= \langle \pi_{i^*} | 0 \dots 0, \hat{z}_{v,1} \rangle - a_v \\
&= (\langle \pi_{i^*} | 0 \dots 0, \hat{z}_{v,1} - \hat{y}_{v,1}) - a_v + \hat{y}_{v,1} \\
&= d_{v,1} + \hat{y}_{v,1}
\end{aligned}$$

and the other query follows along the same lines.

Round complexity, oracle length, query complexity and message complexity follow from an inspection of the actual protocol definition.

Soundness. Consider that for a successful transcript we must have that all queries pass, i.e. in particular the last two for each $v \in [\aleph]$. There we have from the definition that

$$\langle \pi, \hat{z}_{v,1} | - \overline{\delta}_v \rangle = \langle \pi_j | 0 \dots 0, \hat{z}_{v,1} \rangle - a_v = d_{v,1} + \hat{y}_{v,1}$$

and

$$\langle \pi, \hat{z}_{v,2} | - \overline{\delta}_{v+\aleph} \rangle = \langle \pi_j | 0 \dots 0, \hat{z}_{v,2} \rangle - b_v = d_{v,2} + \hat{y}_{v,2}$$

Here, a_v, b_v are placeholders for the respective values that $\overline{\delta}_v$ and $\overline{\delta}_{v+\aleph}$ select from π. By subtracting $\hat{y}_{v,i}$ from each side we can then define $\mu_{v,i}$ as in the protocol.

As the first query for each $v \in [\aleph]$ must also hold (and because $a_v \cdot b_v = c_v$) it must be that $\mu_{v,1} \cdot \mu_{v,2} = 0$, since otherwise

$$\begin{aligned} d_{v,1} \cdot d_{v,2} &= \mu_{v,1} \cdot \mu_{v,2} - b_v \cdot \mu_{v,1} - a_v \mu_{v,2} + a_v b_v \\ &\neq -b_v \cdot \mu_{v,1} - a_v \mu_{v,2} + a_v b_v \\ &= -d_{v,2} a_v - d_{v,1} b_v - c_v. \end{aligned}$$

The probability for a single $v \in [\aleph]$ to have either of $\mu_{v,i}$ being 0 and thus $\mu_{v,1} \cdot \mu_{v,2} = 0$, can be computed as follows: The events that either are 0 or not are independent. We have $\Pr[\mu_{v,1} = 0] = \Pr[\mu_{v,2} = 0] = 1/|\mathbb{F}|$. Hence, the probability of either of them not being 0 is $\Pr[\mu_{v,1} \neq 0] = \Pr[\mu_{v,2} \neq 0] = (|\mathbb{F}| - 1)/|\mathbb{F}|$. As they are independent, the probability of both of them being non-zero simultaneously must be $\Pr[\mu_{v,1} = 0 \wedge \mu_{v,2} = 0] = (|\mathbb{F}| - 1)^2/|\mathbb{F}|^2$.

A dishonest prover can, when neither having w_1 or w_2, by assumption only make all oracle queries from $\{(\overline{z}_{k,1}, y_{k,1})\}_{k \in [q_1]}$ or $\{(\overline{z}_{k,2}, y_{k,2})\}_{k \in [q_2]}$ be correct with probability $\leq \epsilon_1 + \epsilon_2$ by a union bound. Thus, assume that both these from Π_1 and Π_2 don't go through. Those queries actually made by V in $\Pi_{\mathtt{A-OR}}$ go through for one $v \in [\aleph]$ with probability $1 - \left(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}\right)^2$ by the reasoning shown above and the independent choice of $\rho_{v,1}, \rho_{v,2}$. Repeating the experiment $\aleph$ times as in the protocol yields the promised soundness error.

Honest-Verifier Zero-Knowledge.

We now construct a simulator $\mathcal{S}$ for the protocol $\Pi_{\mathtt{A-OR}}$ that will, on input x_1, x_2, output a transcript τ that is perfectly indistinguishable from a transcript generated by interaction with an honest V.

Pick a protocol instance Π_i at random, run its HVZK simulator $\mathcal{S}_i$ and generate all c_h using $\mathcal{CP}$. This will be perfectly indistinguishable from any transcript of Π_j with $i \neq j$ by the HVZK property of Π_i and the definition of Equisimulatability. Choose uniformly random $\rho_{v,i}$ as in the real protocol. Additionally choose uniformly random values $d_{v,1}, d_{v,2}$ for all $v \in [\aleph]$. This is indistinguishable from the $d_{v,i}$ from $\Pi_{\mathtt{A-OR}}$ where they are computed by subtracting a uniformly random a_v or b_v from a secret value.

Public Coin. The argument is the same as for $\Pi_{\mathtt{OR}}$. □

Recursing. We now show how to recursively apply the construction of $\Pi_{\mathtt{A-OR}}$, which is possible if all the initial protocols are equisimulatable.

Lemma 1. *Let $\Pi_1, \ldots, \Pi_{2m}$ be Public Coin IPs with LOVe with the same properties as in Theorem 7. Furthermore, assume that* all *$2m$ are equisimulatable. Denote with Π'_i the protocol obtained by applying $\Pi_{\mathtt{A-OR}}$ to Π_{2i-1}, Π_{2i}. Then all $\Pi'_1, \ldots, \Pi'_m$ are also equisimulatable.*

Proof. By assumption, there already exists an algorithm $\mathcal{CP}$ which takes care of simulating all the messages that each $\Pi_{\mathtt{A-OR}}$ would perform in Step 2. All that remains to show is that also the messages sent in Step 4 of each instance of $\Pi_{\mathtt{A-OR}}$ can be packed into one vector c. This follows trivially: by the proof of the zero-knowledge property of $\Pi_{\mathtt{A-OR}}$ we know that all $d_{v,i}$ for each $\Pi_{\mathtt{A-OR}}$ are uniformly random. Let each Π'_i use the same repetition parameter $\aleph$, then $\mathcal{CP}$ on input $d_{v,1}, d_{v,2}$ will simply output $c = [d_{1,1}, \ldots, d_{\aleph,1}, d_{1,2}, \ldots, d_{\aleph,2}]$. Decode will reconstruct $d_{v,1}, d_{v,2}$ from such c accordingly. □

This now allows us to construct a protocol for $\mathcal{R}_{\mathtt{OR}}$ by recursively applying Theorem 7 and Lemma 1. For simplicity we assume that m is a power of 2.

Corollary 1. *Let $\Pi_1, \ldots, \Pi_m$ be protocols such that each Π_i is a t_i-round q_i-query equisimulatable Public Coin IP with LOVe over $\mathbb{F}$ for relation $\mathcal{R}_i$ with oracle length ℓ_i, message complexity α_i and soundness error ϵ_i. Furthermore, let $\aleph \in \mathbb{N}^+$ and assume that $a_{v,w} \cdot b_{v,w} = c_{v,w}$ holds for all $v \in [\aleph]$, $w \in [\log_2(m)]$.*

Then there exists a protocol $\Pi_{\mathtt{R-OR}}$ that is a Public Coin IP with LOVe for the relation $\mathcal{R}_{\mathtt{OR}}$ with

1. *round complexity $2 \cdot \log_2(m) + \max_{i \in [q]}\{t_i\}$;*

2. *oracle length* $3 \cdot \log_2(m) \cdot \aleph + \max_{i\in[m]}\{\ell_i\}$;

3. *query complexity* $3 \cdot \aleph$;

4. *message complexity* $(2\log_2(m) + 2 \cdot \max_{i\in[m]} q_i + 6(\log_2(m) - 1) \cdot \aleph) \cdot \aleph + \max\{\alpha_i\}$; *and*

5. *soundness error* $\sum_{i\in[m]} \epsilon_i + (m-1) \cdot \left(1 - \left(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}\right)^2\right)^{\aleph}$.

Proof. The idea for $\Pi_{\mathtt{R-OR}}$ is straightforward - first apply Lemma 1 to go from m protocols to $m/2$ instances. Throughout we will only use one oracle string π. Since P has $(x_{i^*}, w_{i^*}) \in \mathcal{R}_{i^*}$ it will first compute π_{i^*} as in Π_{i^*}, pad it with 0s as in $\Pi_{\mathtt{OR}}$ and then add $\log_2(m) \cdot \aleph$ triples $a_{v,w}, b_{v,w}, c_{v,w}$ where $a_{v,w}, b_{v,w}$ are uniformly random and $c_{v,w} = a_{v,w} \cdot b_{v,w}$. This is $\log_2(m)$ as many triples as in $\Pi_{\mathtt{A-OR}}$ and we need these for the recursion.

As before, we simulate the messages using the true branch and in the end compute the queries that would occur in any of the m branches from the messages that V obtains based on each Π_i. We then use the triples for $w = 1$ to compute the d_v-messages based on the instance of $\Pi_{\mathtt{A-OR}}$ where the true branch Π_{i^*} is a part of. Then, for each of the overall $m/2$ instances of $\Pi_{\mathtt{A-OR}}$ we compute the queries which V would make. Observe that we do not actually make these now. In each branch of $\Pi_{\mathtt{A-OR}}$, the adversary could have lied only with probability $\epsilon_{2i-1} + \epsilon_{2i} + \left(1 - \left(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}\right)^2\right)^{\aleph}$ by Theorem 7.

Next, as the remaining $m/2$ protocols are equisimulatable by Lemma 1, applying $\Pi_{\mathtt{A-OR}}$ to combine these will lead to $m/4$ equisimulatable protocols. We again combine the $m/2$ equisimulatable public coin IPs with LOVe using Theorem 7, now using the multiplication triples in π for $w = 2$. As we applied $\Pi_{\mathtt{A-OR}}$ in the last step and know that each of the first layer of $\Pi_{\mathtt{A-OR}}$ instances makes $3\aleph$ queries, V chooses challenges ρ that combine these appropriately as before. P computes the $d_{v,i}$-values necessary for the multiplication proofs done in the second layer of $\Pi_{\mathtt{A-OR}}$ instances on the inner products of the respective queries that would have been made in the end of the first layer of $\Pi_{\mathtt{A-OR}}$ instances as before. The values $d_{v,i}$ that we send in the second layer are computed based on the one of the $m/4$ instances of $\Pi_{\mathtt{A-OR}}$ that the true statement Π_{i^*} is a part of. The soundness error is by the definition of $\Pi_{\mathtt{A-OR}}$ the sum of the soundness error on either branch of the protocol plus an additional error term $\left(1 - \left(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}\right)^2\right)^{\aleph}$.

Repeating the aforementioned process $\log_2(m)$ times then leads to the claimed properties. □

Again, we have that $(2 \cdot \max_{i\in[m]} q_i + 6(\log_2(m) - 1) \cdot \aleph) \cdot \aleph$ field elements are chosen uniformly at random by V. In terms of practical complexity, our proof loses in soundness as m increases, which is expected. The additional loss term $(m-1) \cdot \left(1 - \left(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}\right)^2\right)^{\aleph}$ does, on the other hand, not have such a big impact: if $\mathbb{F} = 2$ then we'd have to choose $\aleph = 193$ for $m = 2$ to bring this error term down to $\approx 2^{-80}$. If we set $m = 1024$ then $\aleph = 217$ is sufficient. On the other hand, if $|\mathbb{F}|$ is exponential in the security parameter then we can always let $\aleph = 1$ and the additional term in $\Pi_{\mathtt{R-OR}}$ would still be negligible.

B Proof of Theorem 5

Proof. When transforming the interactive protocol Π_{Circuit}, for each $i > 1$, the i-th challenge is computed as:

$$e_i = \mathsf{H}(a_i \| e_{i-1})$$

where a_i is the prover's messages which were sent in the last round, that is, $a_i = (\delta_i \| \delta_i' \| \varepsilon_{i-1})$.

We begin by defining two events, concerning a malicious prover $\mathcal{A}$ in the protocol $\mathsf{FS}(\Pi_{\mathsf{Circuit}})$ for some arithmetic circuit C (corresponding to a statement x). We define the value Δ_i based on the deviation by $\mathcal{A}$ in the i-th multiplication gate, as in (1).

We define the events:

1. E_1 to be the event that $\mathcal{A}$ outputs a proof string π and a corresponding accepting transcript $(a_1, \ldots, a_t)$.
2. E_2 to be the event that both of the following hold: (1) If $(q_1, \ldots, q_t) := (x \| \mathsf{id}_\pi \| a_1, a_2 \| e_1, \ldots, a_t \| e_{t-1})$, and $\mathcal{A}$ queries both q_i and q_j for some $i < j$, then q_i was queried first; and (2) For any i where $\Delta_i \neq 0$, $\mathcal{A}$ queries $a_i \| e_{i-1}$.

Lemma 2. *Let x be a statement defining an arithmetic circuit C, and $\mathcal{A}$ be a malicious prover who makes Q queries to the random oracle H. Then, $\Pr[E_1 \wedge \neg E_2] \leq \frac{Q}{|\mathbb{F}|}$.*

Proof. Suppose that event E_1 occurs, and E_2 does not. Then, at least one of the following holds: (1) $\mathcal{A}$ queried $q_j = a_j \| e_{j-1}$, and then later q_i, for some $i < j$, (2) There exists an i where $\Delta_i \neq 0$, and $\mathcal{A}$ did not query $a_i \| e_{i-1}$.

Suppose (1) is true, and let i, j be a pair of indices where $i < j$ and $j - i$ is smallest, such that $\mathcal{A}$ queried q_j before q_i. We can therefore assume that $\mathcal{A}$ did not query the oracle on q_{j-1} before querying $q_j = a_j \| e_{j-1}$. Since $e_{j-1} = \mathsf{H}(q_{j-1})$, this means that the behaviour of $\mathcal{A}$ up until querying $a_j \| e_{j-1}$ is independent of e_{j-1}, and so this query happens with probability at most $Q/|\mathbb{F}|$.

In case (2), let i be such that $\Delta_i \neq 0$, but $\mathcal{A}$ did not query $a_i \| e_{i-1}$. Since the behaviour of $\mathcal{A}$ is independent of e_i, it can only produce a consistent transcript with probability $1/|\mathbb{F}|$, since it must firstly send $\varepsilon_i = e_i x_i - a_i$, and without knowing e_i this only holds with probability $1/|\mathbb{F}|$, unless $x_i = 0$. Even if $x_i = 0$, then $\mathcal{A}$ must still have passed the second $\mathsf{AssertZero}$, which requires that $\Delta_i e_i = \Delta'_i$, hence also requires guessing e_i. □

Lemma 3. *Let $\mathcal{A}$ be a malicious prover for some statement x that is not satisfiable, and suppose that $\mathcal{A}$ makes Q queries to the random oracle. Then, $\Pr[E_1 \wedge E_2] \leq 2Q/|\mathbb{F}| + 2^{-\lambda}$.*

Proof. Suppose that the proof accepts and event E_2 occurs. Since x is not satisfiable, there must exist a multiplication gate i, with inputs x_i, y_i, for which $\mathcal{A}$ used the incorrect product when committing to $x_i y_i$; that is, $\Delta_i \neq 0$. By assumption, $\mathcal{A}$ queried $a_i \| e_{i-1}$ to the random oracle to obtain e_i, for which $\Delta_i e_i = \Delta'_i$, since the proof accepts. We will now show that the value Δ_i must have been fixed at the time of $\mathcal{A}$ querying $a_i \| e_{i-1}$. Recall that Δ_i is the difference between the value $\delta'_i = r_i - x_i y_i$ an honest prover should send, and the actual δ_i sent by the adversary. For Δ_i to be fixed, it is enough to show that r_i, x_i, y_i have all been fixed at this point.

Claim 1. *If $\mathcal{A}$ made query $q_i = (a_i \| e_{i-1})$, then it also queried $q_1, \ldots, q_{i-1}$, and all these queries were made after submitting the proof π, except with probability at most $Q \cdot (|\mathbb{F}|^{-1} + 2^{-\lambda})$.*

Proof. Suppose at least one such query was not made, and let $j < i$ be an index where q_j was not queried, but $q_{j+1} = (a_{j+1} \| e_j)$ was. Since $e_j = \mathsf{H}(q_j)$, the probability that $\mathcal{A}$ queried q_{j+1} is at most $Q/|\mathbb{F}|$. Next, we argue that all queries $q_1, \ldots, q_i$ must have been made after π was chosen by $\mathcal{A}$. Note that by assumption (1) of event E_2, these queries were all made in order so we only need to consider q_1. If $q_1 = x \| \mathsf{id}_\pi \| a_1$ was queried before choosing π, then it was made independently of id_π, so this happens only with probability at most $Q/2^\lambda$. □

Assume now that all queries $q_1, \ldots, q_i$ were made after choosing the proof π. It follows that r_i was fixed prior to these queries being made. Since x_i, y_i are determined by some message sent in a previous Fix instruction, and also by π, these values were also fixed by these queries. This implies that Δ_i is already determined when $\mathcal{A}$ queries q_i, and a similar argument holds for Δ'_i. Hence, the probability that e_i satisfies $\Delta_i e_i = \Delta'_i$ is at most $Q/|\mathbb{F}|$.

Taking a union bound across all failure events, we get an overall probability of at most $2Q/|\mathbb{F}| + Q/2^\lambda$. □

Finally, considering a statement x which is not satisfiable, we can sum up the bounds from Lemmas 2 and 3, getting

$$\Pr[E_1] = \Pr[E_1 \wedge E_2] + \Pr[E_1 \wedge \neg E_2] \leq \frac{3Q}{|\mathbb{F}|} + \frac{Q}{2^\lambda}$$

□

C Proof of Theorem 6

Proof. Define the possible challenge queries $q_1 = \mathsf{id}_\pi \| a_1$ and $q_i = e_{i-1} \| a_{j_i}$, for $i > 1$.

Claim 2. *Suppose $\mathcal{A}$ queried q_i for some i. Then, $\mathcal{A}$ also queried $q_1, \ldots, q_{i-1}$, and made all these queries in order, after submitting π, except with probability at most $Q/|\mathbb{F}| + Q/2^\lambda$.*

Proof. This proceeds identically to the Claim in the proof of Theorem 5. □

Now consider an $\mathcal{A}$ in an execution of $\mathsf{Stream}(\Pi_{\mathsf{NI}})$, where there exists an i such that $\gamma_i \neq 0$ (and so in the original proof Π_{NI}, the proof accepts with probability at most ε). Let i be the largest such index, and define

$$z_i = \sum_{j=1}^{i} e_j \gamma_j$$

If the proof accepts (that is, if $\mathsf{AssertZero}(\underline{z})$ succeeds), then it must hold that $z_i = 0$.

By the Claim, if $\mathcal{A}$ queries q_i to the random oracle, then it previously queried each of $q_1, \ldots, q_{i-1}$ in turn, and after receiving the identifier id_π (except with negligible probability). It follows that when querying q_i, all the γ_j, for $j \leq i$, had been fixed, because these are determined uniquely by the randomness in π and the messages sent prior to this instruction, which were all given as part of the queries $\{q_j\}_{j \leq i}$. Hence, in this case, we have $\Pr[z_i = 0] \leq Q/|\mathbb{F}|$, because it only holds if

$$e_i = -\gamma_i^{-1} \sum_{j=1}^{i-1} e_j \gamma_j \tag{2}$$

On the other hand, if $\mathcal{A}$ does not query q_i to the oracle, then the execution of $\mathcal{A}$ is independent of e_i. After completing the protocol, therefore, the probability that $e_i = \mathsf{H}(q_i)$ satisfies (2) is $1/|\mathbb{F}|$.

Combining the above failure events, the overall soundness is no more than

$$\varepsilon + \frac{2Q+1}{|\mathbb{F}|} + \frac{Q}{2^\lambda}$$

□