Mac'n'Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions

Carsten Baum[1], Alex J. Malozemoff[2], Marc B. Rosen[2], and Peter Scholl[1]

[1] Aarhus University
[2] Galois, Inc.

Abstract. Zero knowledge proofs are an important building block in many cryptographic applications. 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 boolean and arithmetic circuits, called Mac'n'Cheese, with a focus on supporting large circuits. 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 online phase of Mac'n'Cheese and achieve a runtime of 144 ns per AND gate and 1.5 μs per multiplication gate in $\mathbb{F}_{2^{61}-1}$ when run over a network with a 95 ms latency and a bandwidth of 31.5 Mbps. In addition, we show that the disjunction optimization improves communication as expected: when proving a boolean circuit with eight branches and each branch containing roughly 1 billion multiplications, Mac'n'Cheese requires only 75 more bytes to communicate than in the single branch case.

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 family of novel ZK proof protocols called Mac'n'Cheese, which are 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] in a preprocessing phase, which can generate many such random commitments with only a small amount of interaction and computation [WYKW20].

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 efficiently evaluating *disjunctive statements*, namely, to prove that one out of m statements is true, the prover only needs to communicate the information needed to evaluate the true branch among all 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. The idea of optimizing disjunctive statements in this way was first considered in recent work on stacked garbling [HK20b], with proofs of disjunctions based on garbled circuits. They observed that disjunctive statements can arise in many natural applications, such as when proving in zero-knowledge the existence of a bug in a program, so optimizing these is well-motivated.

At a high level, our technique can be seen as a generalized OR composition for m protocols, where the resulting OR proof has communication complexity proportional to $\max\{C_i\}$, where C_i is the complexity of the i-th protocol. Contrasted with the classic OR proof approach [CDS94], which requires $\sum C_i$ communication, our techniques for stacking save a multiplicative factor of m. On the other hand, compared with stacked garbling [HK20b], our underlying protocols have around $20\times$ less communication, and are also more flexible, since we can support both boolean and arithmetic circuits.

Efficiency comparison and related work. Table 1 shows the efficiency of our protocols alongside other VOLE- or garbled-circuit-based protocols, where we fo-

Protocol	Boolean			Arithmetic			Disjunctions
	Comm.	Rounds	Mmps	Comm.	Rounds	Mmps	
Stacked garbling [HK20b]	128	3	0.3[1]	—	—	—	✓
Wolverine [WYKW20]	7*	3	2.0[2]	4	3	0.2[2]	✗†
Line-Point ZK [DIO21]	—	—	—	1	3	—	✗
QuickSilver [YSWW21]	1	3	12.2[3]	1	3	1.4[3]	✗
Mac′n′Cheese (simple)	9	3	—	3	3	—	✓
Mac′n′Cheese (batched)	$1+\epsilon^*$	$O(\log b)$	6.9[4]	$1+\epsilon^*$	$O(\log b)$	0.6[4]	✓

* For large batches (e.g., $b \geq 1$ million).
† While we believe Wolverine can be combined with our approach described in § 3.1, the performance implications of this combination are unclear.
[1] With a 100 ms latency and a 100 Mbps bandwidth.
[2] With a 0.1 ms latency and a 50 Mbps bandwidth.
[3] With a 0.1 ms latency and a 30 Mbps bandwidth for boolean and a 100 Mbps bandwidth for arithmetic.
[4] With a 93 ms latency and a 31.5 Mbps bandwidth.

Table 1. Comparison of different GC and VOLE-based ZK protocols (costs exclude OT/VOLE setup). "Comm." denotes the number of field elements communicated per multiplication. "Rounds" denotes the total number of rounds required, where we count rounds as the number of message flows, so one round is a single message from the prover to the verifier. "Mmps" denotes the number of multiplication gates per second in millions. *We caution against reading too much into these numbers due to differing experimental environments, and provide them mostly as a rough comparison guide.* "Disjunctions" denotes those protocols that support communication-optimized disjunctions. The variable b denotes the batch-size of multiplications, and ϵ denotes a value close to zero that depends on b. Concretely, for a batch size of $b = 1\,000\,000$ we require 17 rounds with $\epsilon = .008$ for the boolean case (using $\mathbb{F}_{2^{40}}$) and $\epsilon = .257$ for the arithmetic case (using $\mathbb{F}_p$ for $p = 2^{61} - 1$).

cus on communication cost per multiplication gate measured in field elements. As far as we are aware, there are only two ZK approaches that can successfully scale to large statements: the garbled circuit ZK approach [JKO13,FNO15,ZRE15,HK20b], and the more recent approach based on VOLE, namely the concurrent works Wolverine [WYKW20] and Line-Point ZK [DIO21], plus QuickSilver [YSWW21] (which builds on Line-Point ZK). All of these VOLE-based 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.

For Mac′n′Cheese, our first class of "simple" protocols reduces the communication complexity of Wolverine from 4 to 3 field elements for arithmetic circuits, and achieves a slightly higher cost (9 bits) for boolean circuits, while avoiding the need to amortize over many gates. Our second set of protocols has essentially the same practical cost as Line-Point ZK and QuickSilver, but is best run in large batches, so suited for bigger circuits. Importantly, all of our protocols

are compatible with our technique for efficient disjunctions—we currently do not know how to efficiently adapt this technique for QuickSilver or Line-Point ZK[3].

Finally, note that for zero knowledge from garbled circuits, the best approach currently has a communication cost of 128 bits per AND gate, which is around 18× higher than our approach that also supports disjunctive statements. We note that Heath and Kolesnikov [HK20b] were the first to consider "stacked" disjunctive proofs. Our approach was inspired by stacked garbling, but the technique is very different, and closer in spirit to the earlier 'free if' for private function evaluation [Kol18] (although neither technique follows from the other).

Implementation. We implemented the online phase of Mac′n′Cheese in the Rust programming language. Currently, we do not have an implementation of VOLE, which should add a small amount of communication—0.42 bits per VOLE—and slight increase in runtime—at most 85 ns per VOLE [WYKW20, Table 4].

When run on a real-world network (95 ms latency and a bandwidth of 31.5 Mbps), Mac′n′Cheese requires approximately 1.5 μs per multiplication gate (for $\mathbb{F}_{2^{61}-1}$), and 144 ns per AND gate (using $\mathbb{F}_{2^{40}}$). Run locally, Mac′n′Cheese requires approximately 276 ns per multiplication gate and 141 ns per AND gate. We also show that disjunctions have large communication savings: when run on a circuit containing eight branches each of which contains 1 billion multiplication gates, we see a communication increase of only *75 bytes* versus running a single 1-billion-gate branch.

1.2 Our Techniques

We present the Mac′n′Cheese approach in four 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 explain how IPs with LOVe naturally support nested disjunctions and can be compiled to ZK protocols using VOLE. We then provide efficient IPs with LOVe for general circuit satisfiability, which intuitively follow from such protocols for homomorphic commitments. Finally, we show that our protocols are compatible with streaming and that we can apply the Fiat-Shamir transform to reduce the round complexity with only a small loss in soundness.

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

[3] The challenge in applying our disjunction optimization to these protocols is that the verification check requires input from V, which allows a malicious V to try to guess the evaluated branch by supplying an invalid value for all other branches.

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

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

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

- This is clear for testing the output of the output gate.
- 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 VOLE (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 (§ 2). Proofs based on ideal homomorphic commitments can be modeled as a functionality where the prover initially commits to some secret values, and the verifier is then allowed to perform linear queries to the commitments, to check that certain relations hold. For instance, linear interactive oracle proofs (IOPs) [BBC+19] model exactly this. In § 2, we extend this paradigm with a new abstraction called *interactive proofs with linear oracle verification (IPs with LOVe)*. In this abstraction, P begins by committing some proof string $\boldsymbol{\pi}$ to an oracle $\mathcal{O}$. The parties then exchange messages for a fixed number of rounds, after which V sends multiple queries of the form $(\boldsymbol{z}_i, y_i)$ 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 $\langle \boldsymbol{\pi}, \boldsymbol{z}_i \rangle = y_i$ or not for each of these queries. Eventually, V outputs a bit to represent whether it accepts or not.

The key difference between IPs with LOVe and linear IOPs [BBC+19], is that on top of oracle queries, we allow the prover and verifier to exchange a number of messages. Therefore, IPs with LOVe naturally model homomorphic

commitments in the same way as linear IOPs, while also giving extra power from the exchange of messages, which is what we exploit in our protocols for efficient disjunctions.

From IPs with LOVe to IPs with VOLE. We show that any Public Coin IP with LOVe can be combined with a VOLE protocol to obtain a ZK proof. This is described in § 2.2. We instantiate the oracle $\mathcal{O}$ that contains the string $\boldsymbol{\pi}$ using information-theoretic commitments (or MACs) of the form

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

where x comes from a field $\mathbb{F}_p$ and all remaining value from an extension field $\mathbb{F}_{p^k}$ for $k \geq 1$. 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. Their binding guarantee follows from the size of $\mathbb{F}_{p^k}$.

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.

Disjunctive proofs for IPs with LOVe (§ 3). Our main technical contribution, described in § 3.1, 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. Note that our transformation is different to the stacked garbling approach [HK20b] (which does not fit the IP with LOVe paradigm), and we obtain much greater efficiency when using our IPs with LOVe instead.

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 derived deterministically from the protocol transcript. This is indeed the case for our general IP with LOVe protocols that we describe later. 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 $\boldsymbol{w}_{i^*}$ 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, logarithmic overhead, and recursive nesting. The OR-proof of Cramer et al. [CDS94] can be generalized for any threshold r out of m, showing that at least r instances of $\Pi_1, \ldots, \Pi_m$ were correct. We generalize our protocol to this setting, with communication r times that of Π_i, instead of m.

While the above techniques avoid the factor m blowup from [CDS94], they do still incur an *additive* $O(m)$ overhead in the number of statements. We present a different approach, which reduces this to *logarithmic* using recursion. The key idea is that we can build a 1-out-of-2 disjunctive proof, which itself satisfies the conditions required to be stacked. Applying recursion in a binary tree-like manner, we obtain a 1-out-of-m proof with $O(\log m)$ overhead. Note that the ability to recurse is also useful when capturing proofs about complex programs, which may contain arbitrary nested levels of disjunctions, with communication proportional to the longest path through the entire program. The original stacked garbling approach [HK20b] did not support nested disjunctions, however, a later update shows how to handle them [HK20a].

Efficient IPs with LOVe for circuit satisfiability (§ 4). Towards efficiently instantiating IPs with LOVe, in § 4.1 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 (*$\mathsf{C\&P}$*) 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 a simple protocol for circuit satisfiability.

Next, in § 4.3 we present an optimized circuit satisfiability protocol that batch-checks n multiplication gates simultaneously. To achieve this, we adapt the $\log(n)$-round inner product check of Boneh et al. [BBC$^+$19] to $\mathsf{C\&P}$ IPs with LOVe, which we then use for the batch check.

Due to the additive overhead generated from the batch check, it might not be the most communication-efficient approach for binary circuits when n is small. In § 4.4 we therefore present a batch check of multiplications for binary circuits that has an overhead of 9 bits, essentially independent of n. This check uses *reverse multiplication-friendly embeddings* [BMN18,CCXY18] which were previously mainly used for efficient multiplications in MPC protocols.

Streaming and removing interaction (§ 5). 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 generating the random VOLEs). We show how to achieve a small memory footprint 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*. The Fiat-Shamir transform is typically only applied to constant-round protocols, since in the worst-case, the soundness can degrade *exponentially* with the number of rounds [BCS16]. Several works, however, have defined extra conditions on the underlying protocol which suffice to avoid this degradation, for the cases of interactive oracle proofs [BCS16] and general interactive proofs [CCH$^+$19].

Following in this direction, we adapt the concept of *round-by-round soundness* [CCH$^+$19] of interactive proofs to IPs with LOVe. We then show that by applying a Fiat-Shamir transform, any IP with LOVe satisfying this modified notion can be transformed into a NIZK (with VOLE preprocessing) in the random oracle model, with negligible soundness degradation. Finally, we also show that our streamable protocols for circuit satisfiability do indeed have round-by-round soundness, so can safely be made non-interactive.

2 Interactive Proofs with Linear Oracle Verification

In this section we introduce our proof methodology, called *interactive proofs with linear oracle verification* (IPs with LOVe). In addition, we show how, using vector oblivious linear evaluation (VOLE), any public coin IP with LOVe can be turned into a zero-knowledge proof.

Notation. For any vector $\boldsymbol{r}$ we denote by $\boldsymbol{r}|_t$ the restriction to the first t elements and by $\boldsymbol{r}[i]$ the ith element of $\boldsymbol{r}$. Let $[\mathsf{P} \leftrightarrow \mathsf{V}]$ denote the distribution of exchanged messages between two parties 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. We defer the (standard) definition of zero-knowledge proofs to the full version.

2.1 Definitions

We now formalize IPs with LOVe over a finite field $\mathbb{F}_{p^k}$. 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 $\boldsymbol{\pi}$, which is a vector of field elements. Then, both P and the

verifier V exchange messages for a certain number of rounds. Finally, V issues a number of affine queries to $\boldsymbol{\pi}$, 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 $\boldsymbol{\pi} \in \mathbb{F}_{p^k}^{\ell}$ at the beginning of the protocol and allow V to access it via oracle queries only at the end of the protocol. In the oracle query stage, we let V choose q queries $(\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_q, y_q) \in \mathbb{F}_{p^k}^{\ell} \times \mathbb{F}_{p^k}$ which it sends to an oracle that stores $\boldsymbol{\pi}$. This oracle checks that for each of the q queries the relation $\langle \boldsymbol{\pi}, \boldsymbol{z}_i \rangle = y_i$ holds. The query results are then (truthfully) reported to V by the oracle.

Note that by default, both $\boldsymbol{\pi}$ and the queries lie over the extension field $\mathbb{F}_{p^k}$. In some cases, such as when we are proving statements over $\mathbb{F}_p$, some elements of $\boldsymbol{\pi}$ may only be in $\mathbb{F}_p$, which allows for improved efficiency when instantiating IPs with LOVe, as we will see later. In this case, during the query phase we view any $\mathbb{F}_p$ value as an element of $\mathbb{F}_{p^k}$ via some fixed embedding.

Definition 1 (Interactive Protocol with Linear Oracle Verification). *Let $\mathbb{F}_{p^k}$ 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 $\mathbb{F}_{p^k}$ 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 $\boldsymbol{\pi} \in \mathbb{F}_{p^k}^{\ell}$ to the oracle. P then outputs a state s_0^{P} while V outputs a state s_0^{V}. We set an auxiliary variable $a_0 = \perp$.*
2. *For round $h \in [t]$, P and V do the following:*
   (a) *First, V on input s_{h-1}^{V} and a_{h-1} outputs message $\boldsymbol{e}_h \in \mathbb{F}_p^{r_h^{\mathsf{V}}}$ and state s_h^{V}.*
   (b) *Then, P on input s_{h-1}^{P} and $\boldsymbol{e}_h$ outputs message $\boldsymbol{a}_h \in \mathbb{F}_p^{r_h^{\mathsf{P}}}$ and state s_h^{P}.*
3. *Finally, V on input $\boldsymbol{a}_t$ and state s_t^{V} makes q linear oracle queries to $\boldsymbol{\pi}$ over $\mathbb{F}_{p^k}$ and outputs a bit.*

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

Remark 1. Note that the prover and verifier's messages $(\boldsymbol{a}_h, \boldsymbol{e}_h)$ are specified as elements of the base field $\mathbb{F}_p$, and this is how we count message complexity. This is an arbitrary restriction, since these messages can easily be used to encode extension field elements or general bit strings.

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 $\mathbb{F}_{p^k}$ 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 $(x, \boldsymbol{w}) \in \mathcal{R}$ the interaction between $\mathsf{P}(x, \boldsymbol{w})$ 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 $(x, \boldsymbol{w}) \in R$ the output of $S(x)$ is perfectly indistinguishable from $\mathtt{View}_V[\mathsf{P}(x, \boldsymbol{w}) \leftrightarrow \mathsf{V}(x)]$ for any honest V.*

We use the notation IP-LOVe to denote a t-round, q-query HVZK IP with LOVe for relation $\mathcal{R}$ over field $\mathbb{F}_{p^k}$ with oracle length ℓ, message complexity α elements of $\mathbb{F}_p$, 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). *Let Π be an* IP-LOVe *protocol for a statement x using a proof string $\boldsymbol{\pi}$ such that V accepts with probability $> \epsilon$. Then Π is a* proof of knowledge *if there exists a PPT extractor E that, on input $x, \boldsymbol{\pi}$, outputs a witness $\boldsymbol{w}$ such that $(x, \boldsymbol{w}) \in \mathcal{R}$.*

Definition 4 (Public Coin IP with LOVe). *An* IP-LOVe *protocol Π is* public coin *if*

1. *V chooses each $\boldsymbol{e}_h \in \mathbb{F}_p^{r_h^{\mathsf{V}}}$ for $x \in L(\mathcal{R})$ uniformly at random (and in particular,* independent *of s_{h-1}^{V} and $\boldsymbol{a}_{h-1}$).*
2. *There exists a deterministic polytime algorithm $\mathcal{Q}$, which, on input x and $\{\boldsymbol{e}_h, \boldsymbol{a}_h\}_{h \in [t]}$, computes the q oracle queries $(\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_q, y_q)$ of V.*
3. *V accepts iff all queries generated by $\mathcal{Q}$ are accepting.*

From q-query to 1-query. Given a q-query IP with LOVe, we can always convert it to one with a single oracle query, with a small loss in soundness, by taking random linear combinations of all queries over a large enough extension field. This transformation, given below, is public-coin and adds just one extra round of communication, so when using IPs with LOVe, we will often assume they have only one query, to simplify our protocols.

Let Π be an IP-LOVe over $\mathbb{F}_p$ (p need not be prime), and let k be such that p^k is superpolynomial in a statistical security parameter. We construct an IP with LOVe over $\mathbb{F}_{p^k}$, by viewing the proof $\boldsymbol{\pi} \in \mathbb{F}_p^\ell$ from Π as a vector in $\mathbb{F}_{p^k}^\ell$, running the same protocol and then modifying the query phase as follows. Recall that the q queries $(\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_q, y_q)$ in Π accept if and only if $\langle \boldsymbol{\pi}, \boldsymbol{z}_i \rangle = y_i$ i.e. $\mu_i := \langle \boldsymbol{\pi}, \boldsymbol{z}_i \rangle - y_i = 0$ in $\mathbb{F}_p$. Now, we modify the protocol by having V send q random elements $\rho_1, \ldots, \rho_q \in \mathbb{F}_{p^k}$ to P.[4] Notice that if $\mu := \sum_{i \in [q]} \rho_i \mu_i = 0$, all queries are satisfied except with probability p^{-k}. We equivalently have $\mu = \langle \boldsymbol{\pi}, \boldsymbol{z} \rangle - y$ for $\boldsymbol{z} = \sum_{i \in [q]} \rho_i \boldsymbol{z}_i$ and $y = \sum_{i \in [q]} \rho_i y_i$. This shows we can reduce the q oracle queries down to just one query $(\boldsymbol{z}, y)$ over $\mathbb{F}_{p^k}$, at the cost of an extra q elements of $\mathbb{F}_{p^k}$ sent from V to P, and the soundness error increasing by p^{-k}.[5]

[4] If we did not want a public-coin protocol, we could skip this message from V to P.

[5] Alternatively, V could send a single random $\rho \in \mathbb{F}_{p^k}$, and define $\rho_i = \rho^i$. This reduces communication while increasing the error probability to $q \cdot p^{-k}$, by applying the Schwartz-Zippel Lemma.

2.2 Instantiating IPs with LOVe Using VOLE

We now show that any IP-LOVe can be transformed into a zero-knowledge proof, by using *vector oblivious linear evaluation (VOLE)* to instantiate the linear oracle queries. The functionality for random VOLE is given in Figure 1: it picks a vector of random samples $(\boldsymbol{r}, \boldsymbol{\tau})$, $(\alpha, \boldsymbol{\beta})$ such that $\boldsymbol{\tau} = \boldsymbol{r}\alpha + \boldsymbol{\beta}$, and outputs them to the respective parties. This can be seen as a secret-sharing of the products $\boldsymbol{r}[i]\alpha$, for $i = 1, \ldots, \ell$. Note that we relax security slightly by allowing corrupt parties to choose their own randomness. This models existing random VOLE protocols based on the LPN assumption [BCGI18,BCG+19b,WYKW20], which can generate a large, length ℓ VOLE with communication that is almost independent of ℓ.

Commitments with MACs. We can view each output of a VOLE as an information-theoretic MAC on the value $\boldsymbol{r}[i]$, which commits the prover to $\boldsymbol{r}[i]$. We write $[x]$ to denote that the prover holds $x, \tau_x \in \mathbb{F}_{p^k}$, while the verifier holds β_x and the fixed MAC key $\alpha \in \mathbb{F}_{p^k}$. 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 $1/p^k$.

Since α is the same for each commitment, these commitments are *linearly homomorphic*. Indeed, given two commitments $[x], [y]$, the parties can obtain $[x + y]$ by computing $x + y, \tau_x + \tau_y$ and $\beta_x + \beta_y$, respectively. Similarly, we can do multiplication by constant, and addition by constant c (here, the verifier adds αc to β_x, while the prover adds c to x). We overload the $+$ and $\cdot$ operators to denote these operations being performed on the commitments.

The transformation (Figure 2). Given the linearly homomorphic commitment scheme based on VOLE, obtaining a ZK proof is relatively straightforward. First, the prover commits to its proof string $\boldsymbol{\pi}$, by sending each component masked with a random VOLE commitment. The parties then run the IP-LOVe protocol as usual, until the query phase. Here, each linear query is computed by applying the linear function to the committed $\boldsymbol{\pi}$, followed by opening the result to check it gives the correct value. In the full version, we prove the following.

Theorem 1. *Suppose Π_{LOVe} is a public-coin* IP-LOVe *for relation $\mathcal{R}$, satisfying completeness, soudness error ϵ and honest-verifier zero-knowledge. Then, $\Pi_{\mathsf{ZK}}^{\mathsf{VOLE}}$ is an honest-verifier zero-knowledge proof for relation $\mathcal{R}$, with soundness error $\epsilon + p^{-k}$. Furthermore, if Π_{LOVe} is a proof of knowledge, then so is $\Pi_{\mathsf{ZK}}^{\mathsf{VOLE}}$.*

Optimizations: Random proof elements, and subfield VOLE. We describe two simple optimizations, which reduce communication in certain cases.

Firstly, in our protocols, the proof string $\boldsymbol{\pi}$ will often contain many random field elements; clearly, the values d_i in Step 2 of the Input Phase (cf. Figure 2) do not need to be sent in this case, since P can choose $\boldsymbol{\pi}[i] = \boldsymbol{r}[i]$.

Functionality $\mathcal{F}_{\mathsf{VOLE}}^{\ell,p,k}$

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

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

- Sample $\boldsymbol{r}, \boldsymbol{\beta} \leftarrow \mathbb{F}_{p^k}^{\ell}$, and set $\boldsymbol{\tau} = \boldsymbol{r}\alpha + \boldsymbol{\beta}$.
  - *If S is corrupted:* receive $\boldsymbol{r}, \boldsymbol{\tau}$ from $\mathcal{A}$ and recompute $\boldsymbol{\beta} = \boldsymbol{\tau} - \boldsymbol{r}\alpha$.
  - *If R is corrupted:* receive $\boldsymbol{\beta}$ from $\mathcal{A}$ and recompute $\boldsymbol{\tau} = \boldsymbol{r}\alpha + \boldsymbol{\beta}$.
- Output $(\boldsymbol{r}, \boldsymbol{\tau})$ to S and $\boldsymbol{\beta}$ to R.

Fig. 1. Ideal functionality for vector oblivious linear evaluation over $\mathbb{F}_{p^k}$.

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

Let ℓ be the length of the proof string in Π_{LOVe}, the underlying IP with LOVe over $\mathbb{F}_{p^k}$.

Input phase: The prover, on input the witness $\boldsymbol{w}$, chooses the proof string $\boldsymbol{\pi} \in \mathbb{F}_{p^k}^{\ell}$ according to Π_{LOVe}.

1. The parties call $\mathcal{F}_{\mathsf{VOLE}}^{\ell,p,k}$. View the outputs as random commitments $[\boldsymbol{r}[1]], \ldots, [\boldsymbol{r}[\ell]]$, where P learns $\boldsymbol{r}[i] \in \mathbb{F}_{p^k}$.
2. P sends $d_i = \boldsymbol{r}[i] - \boldsymbol{\pi}[i]$, for $i = 1, \ldots, \ell$.
3. Compute the commitments $[\boldsymbol{\pi}[i]] = [\boldsymbol{r}[i]] - d_i$.

Protocol: The parties exchange messages in the protocol, according to Π_{LOVe}.

Query phase: Let $(\boldsymbol{z}_j, y_j) \in \mathbb{F}_{p^k}^{\ell} \times \mathbb{F}_{p^k}$, for $j \in [q]$, be the oracle queries defined by Π_{LOVe} (known to both parties, since Π is public-coin). For each j:

1. Compute $[\mu_j] = \sum_{i=1}^{\ell} \boldsymbol{z}_j[i] \cdot [\boldsymbol{\pi}[i]] - y_j$
2. P sends τ_{μ_j} (the MAC on μ_j) to V, who checks that $\tau_{\mu_j} = \beta_{\mu_j}$.

The verifier outputs 1 if all checks pass.

Fig. 2. Zero-knowledge proof from VOLE and IP with LOVe.

Secondly, when working over an extension field $\mathbb{F}_{p^k}$, sometimes it is known that $\boldsymbol{\pi}$ will consist mainly of elements in the base field $\mathbb{F}_p$ (viewed as a subset of $\mathbb{F}_{p^k}$ by a fixed embedding). In this case, we can optimize communication by using *subfield* VOLE instead of VOLE over $\mathbb{F}_{p^k}$. In subfield VOLE [BCG+19b], $\boldsymbol{r}$ is sampled as a uniform vector over $\mathbb{F}_p$ instead of $\mathbb{F}_{p^k}$, while the MACs $\boldsymbol{\tau}$ and keys $\alpha, \boldsymbol{\beta}$ are still computed over $\mathbb{F}_{p^k}$. This allows P to commit to values from the subfield $\mathbb{F}_p$ (which may be small), by sending only elements from $\mathbb{F}_p$, while still achieving soundness error p^{-k}. Note that this still allows committing to an extension field element $x \in \mathbb{F}_{p^k}$ if needed, by decomposing x into a linear combination of $\mathbb{F}_p$ elements, and committing to each component separately (this works because the MACs are linear over $\mathbb{F}_{p^k}$).

When analyzing the complexity of our protocols, we assume that the above two optimizations have been applied.

3 Stackable Public Coin IPs with LOVe

In this section, we 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 over the same field $\mathbb{F}_p$, then we can construct a communication-efficient protocol showing that at least one of the statements was true. Following the terminology of stacked garbling [HK20b], we sometimes refer to this as a *stacked proof*. 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. Suppose we have IPs with LOVe over $\mathbb{F}_p$ for each instance x_i, with message complexity α_i. The classic OR-proof technique by Cramer et al. [CDS94] can be used to give an IP with LOVe with message complexity $\approx \sum_{i \in [m]} \alpha_i$. This would be done by running all m proofs in parallel (which means sending messages for all of them), and then showing that at least one finished with the expected output using [CDS94]. We show how to instead reduce the message complexity of such a proof to $2mk + \max\{\alpha_i\}$, where the soundness error grows by $\approx p^{-k}$. 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 the Π_1 branch, P sends one $\mathbb{F}_p$ element that appears uniformly random to V, while in the Π_2 branch it sends two such elements with the same property. To achieve equisimulatability, if P actually proves the first branch to be true then 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. In both cases, the distribution of the message sent by P is identical and V cannot identify which branch was taken by P.

Formally, for m statements with protocols $\Pi_1, \ldots, \Pi_m$, we use the following two algorithms, which should satisfy the definition below.

- The "combined prover" algorithm $\mathcal{CP}$ takes as input instances $x_1, \ldots x_m$, instance index i, round index h, and prover message $\boldsymbol{a}_h \in \mathbb{F}_p^{r_h^{\mathsf{P}}}$, and outputs a message $c \in \mathbb{F}_p^*$, which encodes $\boldsymbol{a}_h$ while disguising it to hide the index i.
- The "decode" algorithm $\mathtt{dec}$ takes as input instances $x_1, \ldots x_m$, index i, round index h, and combined prover message $c \in \mathbb{F}_p^*$, and recovers $\boldsymbol{a}'_h \in \mathbb{F}_p^{r_h^{\mathsf{P}}}$.

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 $\mathbb{F}_p$ 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. *If* $\boldsymbol{a}_h \leftarrow \Pi_i(s^{\mathsf{P}}_{h-1}, \boldsymbol{e}_h)$*, where* Π_i*'s inputs are from an honest execution of* Π_i*, then*
$$\mathsf{dec}(\hat{x}, i, h, \mathcal{CP}(\hat{x}, i, h, \boldsymbol{a}_h)) = \boldsymbol{a}_h.$$
2. *For any* i, j*, the distributions* $\{\mathcal{CP}(\hat{x}, i, h, \boldsymbol{a}_h) \mid \boldsymbol{a}_h \leftarrow \Pi_i(s^{\mathsf{P}}_{h-1}, \boldsymbol{e}_h)\}_{h\in[t_i]}$ *and* $\{\mathcal{CP}(\hat{x}, j, h, \boldsymbol{a}_h) \mid \boldsymbol{a}_h \leftarrow \Pi_j(s^{\mathsf{P}}_{h-1}, \boldsymbol{e}_h)\}_{h\in[t_j]}$*, where both the inputs of* Π_i *and* Π_j *come from honest executions, are perfectly indistinguishable.*

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

In our constructions of IPs with LOVe, all prover messages will appear uniformly random. We show below that this implies both the zero-knowledge property and equisimulatability, which gives us an easy criterion for proving that an IP-LOVe can be stacked. We prove the following lemma in the full version.

Lemma 1. *Let* Π *be an* IP-LOVe *for proving relation* $\mathcal{R}$*, satisfying completeness, where* V *accepts iff all queries are accepting. If the messages from* P *in an honest execution are (perfectly) indistinguishable from random, it holds that*

1. Π *is honest-verifier zero-knowledge; and*
2. m *such instances of* Π *(potentially for different relations* $\mathcal{R}' \neq \mathcal{R}$*) are equisimulatable (cf. Definition 5).*

3.1 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 protocol, given in Figure 3, is inspired by the stacked garbling approach [HK20b], although uses a very different technique.

We start with m equisimulatable IPs with LOVe Π_i, over $\mathbb{F}_p$, for proving individual relations $\mathcal{R}_i$. Note that p can be any prime power, with no restrictions on size. We construct a protocol $\Pi_{\mathtt{OR}}$, defined over $\mathbb{F}_{p^k}$, which works as follows. P, having only $\boldsymbol{w}_{i^*}$ for one of the statements x_{i^*}, will generate the oracle string $\boldsymbol{\pi}$ by running Π_{i^*}'s first step to create $\boldsymbol{\pi}_{i^*}$, which it then pads with extra random data, and embeds in to $\mathbb{F}_{p^k}$. 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 $\boldsymbol{a}_{h,i}$ using the combined prover algorithm $\mathcal{CP}$, while V extracts the message $\boldsymbol{a}_{h,i}$ for each of the instances from c_h using dec. Due to equisimulatability, V can now execute all instances in parallel but cannot tell 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.

The challenge now, is that V cannot simply perform the oracle queries for all Π_i, since this would reveal the index i^* of the true statement. Instead, we perform a [CDS94]-style OR-proof to shows that at least one of the query's for the Π_i is accepting. Recall, the basic idea behind [CDS94] is that given m Σ-protocols for proving relations, an OR proof can be done by having the *prover*

choose the random challenge f_i for $m-1$ of the instances, so it can simulate the correct messages to be sent in every false instance, without knowing a witness. Then, after receiving the m initial messages of each Σ-protocol, the verifier picks a challenge f, which defines the challenge $f_{i^*} = f - \sum_i f_i$ corresponding to the true instance i^* (while hiding i^* from V).

We instantiate the above, where each "instance" corresponds to a small protocol for verifying that the oracle query $\langle \boldsymbol{\pi}_i, \boldsymbol{z}_i \rangle = y_i$ for protocol Π_i succeeds, without actually performing the query. To carry out one such small protocol, P includes an extra random value r_i in the proof string $\boldsymbol{\pi} := (\boldsymbol{\pi}_i \| r_i)$ (one such r_i for each branch).

For the true Π_{i^*} the prover later sends $d_{i^*} := r_{i^*}$ to V. Using the random challenge f_{i^*}, the verifier then makes a query to test that $\langle \boldsymbol{\pi}, \boldsymbol{z}_{i^*} \| f_{i^*} \rangle =_? y_{i^*} + f_{i^*} d_{i^*}$. This clearly accepts if $d_{i^*} = r_{i^*}$ and the original query ($y_{i^*} =_? \langle \boldsymbol{\pi}_{i^*}, \boldsymbol{z}_{i^*} \rangle$) accepts.

Importantly, if P does *not* know a valid witness, but can pick f_i in advance, then P can cheat by setting $d_i = (\langle \boldsymbol{\pi}_i, \boldsymbol{z}_i \rangle - y_i)/f_i + r_i$, causing the aforementioned oracle query to succeed as well. This is the crux of Phase II of the protocol in Figure 3.

Note that the theorem below assumes that each protocol Π_i uses only one query. As discussed at the end of § 2.1, this can always be achieved by combining queries into one (at the cost of one additional round). The proof of the following theorem can be found in the full version.

Theorem 2. *Let $\Pi_1, \ldots, \Pi_m$ be protocols such that each Π_i is a t_i-round, 1-query, equisimulatable Public Coin IP with LOVe over $\mathbb{F}_p$ 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 3 is a Public Coin IP with LOVe over $\mathbb{F}_{p^k}$ 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;*
4. *message complexity $2mrk + \alpha$ elements of $\mathbb{F}_p$; and*
5. *soundness error $\sum_{i \in [m]} \epsilon_i + 1/p^k$.*

If $\Pi_1, \ldots, \Pi_m$ are all proofs of knowledge, then so is $\Pi_{\mathtt{OR}}$.

Generalizing to threshold proofs. In [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 m of them. $\Pi_{\mathtt{OR}}$ can then be seen as a special case where $r = 1$, where for general r we use Shamir secret-sharing, instead of additive shares of the verifier's challenge f. More details are found in the full version.

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

Let $\Pi_1, \ldots, \Pi_m$ be protocols such that each Π_i is t_i-round, 1-query equisimulatable public coin IP with LOVe over $\mathbb{F}_p$ 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 $\boldsymbol{w}_{i^*}$ for (at least) one $i^* \in [m]$ such that $(x_{i^*}, \boldsymbol{w}_{i^*}) \in \mathcal{R}_{i^*}$. We define $\ell := \max_{i\in[m]} \ell_i$, and $t := \max_{i\in[m]} t_i$. Let $\boldsymbol{z}_{k,i} = \overline{\boldsymbol{z}}_{k,i} \| \underbrace{0 \cdots 0}_{\ell - \ell_i \text{ times}}$.

1. P simulates Π_{i^*} on input $(x_{i^*}, \boldsymbol{w}_{i^*})$ to obtain the string $\boldsymbol{\pi}_{i^*}$. It then sets

$$\boldsymbol{\pi}' = \boldsymbol{\pi}_{i^*} \| \underbrace{0 \cdots 0}_{\ell - \ell_{i^*} \text{ times}} \quad \text{and} \quad \boldsymbol{\pi} = \boldsymbol{\pi}' \| r_1 \cdots r_m$$

   where all r_i are chosen uniformly at random in $\mathbb{F}_{p^k}$.

   (Phase I: Running the stacked proof)
2. Define $s_0^{\mathsf{P}} := (x_{i^*}, \boldsymbol{w}_{i^*})$. 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 $\boldsymbol{e}_h \leftarrow \mathbb{F}_p^{r_h}$ uniformly at random and then sends it to P.
   (b) P sets $(\boldsymbol{a}_h, s_h^{\mathsf{P}}) \leftarrow \Pi_{i^*}(s_{h-1}^{\mathsf{P}}, \boldsymbol{e}_h)$ where P only uses the first r_{h,i^*}^{P} elements of $\boldsymbol{e}_h$ as required by Π_{i^*}. It then computes $c_h \leftarrow \mathcal{CP}(\hat{x}, h, i^*, \boldsymbol{a}_h)$ and sends c_h to V.

   (Phase II: Running the small OR proof)
3. For $i \in [m] \setminus \{i^*\}$, P samples $f_i \leftarrow \mathbb{F}_{p^k}^*$ uniformly at random and computes $(\boldsymbol{z}_i, y_i) \leftarrow \mathcal{Q}(x_i, \{\boldsymbol{e}_h, \mathtt{dec}(\hat{x}, h, i, c_h)\}_{h\in[t_i]})$. It then computes $d_i := (\langle \boldsymbol{\pi}', \boldsymbol{z}_i \rangle - y_i)/f_i + r_i$, and defines $d_{i^*} := r_{i^*}$. Finally, P sends $(d_1, \ldots, d_m) \in \mathbb{F}_{p^k}^m$ to V.
4. V samples $f \leftarrow \mathbb{F}_{p^k}$ uniformly at random and sends it to P.
5. P sets $f_{i^*} := f - \sum_{i\in[m]\setminus\{i^*\}} f_i$ and sends $f_1, \ldots, f_{m-1}$ to V. V computes the last challenge $f_m = f - \sum_{i=1}^{m-1} f_i$.
6. Let $\boldsymbol{\beta}_i \in \mathbb{F}_{p^k}^m$ be the vector that is f_i in the ith position and 0 everywhere else. For $i \in [m]$, V first generates $(\boldsymbol{z}_i, y_i)$ like P in Step 3. Then, for each $i \in [m]$ it sends the query $(\boldsymbol{z}_i \| \boldsymbol{\beta}_i, y_i + f_i d_i)$ to the oracle. V accepts if all queries are true.

Fig. 3. The protocol $\Pi_{\mathtt{OR}}$ for an OR-statement.

3.2 Recursive Stacking

$\Pi_{\mathtt{OR}}$ from § 3.1 has the drawback that to verify one out of m statements, we still need $O(m)$ communication complexity. We now give an alternative construction that obtains an overhead only *logarithmic in* m.

The idea behind this alternative protocol is as follows:

1. Any IP-LOVe Π accepts iff all queries are accepting. Assuming (wlog) there is only one query, this means that for the query (z, y), we have $\langle \boldsymbol{\pi}, z \rangle = y$ i.e. $\mu := \langle \boldsymbol{\pi}, z \rangle - y = 0$.

2. If we simulate the parallel evaluation of m protocol instances as in $\Pi_{\mathtt{OR}}$, then if for any branch i^* it holds that $\mu_{i^*} = 0$, then i^* must correspond to a "true" branch.
3. If the prover can then compute the product $\mu_1 \cdots \mu_m$, and prove that this is 0, then at least one μ_j was 0 to begin with.

A naive instantiation of the above approach is to perform $m - 1$ multiplications between the m implicit variables μ_i, and open the result. However, this would still give $O(m)$ overhead. Instead, we carefully apply recursion to make this overhead logarithmic. Here, we use the fact that after combining two protocols Π_1, Π_2 with the multiplication method sketched above, we can obtain a protocol which *itself is again stackable*: considering all multiplications as a tree, 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.
3. Finally, by recursing the previous step, we obtain the log-overhead OR-proof.

The full construction, together with its proof, can be found in the full version. One drawback of this approach, though, is that unlike our previous OR-proof based method, it does not give rise to a t-out-of-m proof.

4 IPs with LOVe for Circuit Satisfiability

In this section, we present our protocols for proving circuit satisfiability of arithmetic and boolean circuits. First, in § 4.1, we define a high-level *commit-and-prove* (C&P) syntax for IPs with LOVe. This makes it simpler to specify protocols, and also aligns with the VOLE instantiation used in § 2.2. We then describe a simple protocol for arithmetic circuit satisfiability over a finite field $\mathbb{F}_p$ (§ 4.2), with communication cost of 3 field elements per multiplication gate for large p. We next show how we can utilize fully linear PCPs by Boneh et al. [BBC+19] to reduce the amortized multiplication cost to just over 1 $\mathbb{F}_p$ element per multiplication gate (§ 4.3), when the circuit size is large enough. The same approach also works over *binary* fields with the same cost (§ 4.4).

To highlight the power of our disjunctive proof from § 3.1, we point out that all of these protocols fulfil the criteria of our stacking approach, so lead to efficient proofs of disjunctions. Recall that from Lemma 1, it suffices that the sender's messages in the IP-LOVe are uniformly random, which we show for all protocols in this section.

4.1 Defining C&P Protocols

We now define a high-level, commit-and-prove (C&P) syntax for specifying a large class of IPs with LOVe over $\mathbb{F}_{p^k}$.

We require that the witness in the IP with LOVe is a vector $\boldsymbol{w} = (w_1, \ldots, w_n)$ of $\mathbb{F}_p$ elements, and that the prover chooses the proof string $\pi = (w_1, \ldots, w_n, r_1, \ldots, r_t)$, where each r_i is uniformly random. As remarked in § 2.2, we may sometimes wish to mix values in $\mathbb{F}_p$ and $\mathbb{F}_{p^k}$, so allow the possibility that some r_i's are sampled from $\mathbb{F}_{p^k}$ and others are in the base field.

Following the notation used for homomorphic commitments in § 2.2, we write $[x]$ to denote that some value x is committed to by the prover P. Initially, P is committed to every element w_i, r_i of the proof string π. Subsequently, we allow the parties to perform affine operations on these committed values, obtaining new commitments.

Finally, we model the linear verification oracle by a special instruction AssertZero, which checks whether its input is a commitment to 0. Since any commitment comes from an affine function of π, this exactly models linear queries to π. We then specify a C&P protocol over $\mathbb{F}_{p^k}$ as follows:

Input phase: P has input the witness $w_1, \ldots, w_n \in \mathbb{F}_p$, and samples random values $r_1, \ldots, r_t \leftarrow \mathbb{F}_{p^k}^t$ (optionally, some r_i's may be in $\mathbb{F}_p$).
P inputs the proof string $\pi = (w_1, \ldots, w_n, r_1, \ldots, r_t)$.

Protocol phase: The parties, given commitments $[w_1], \ldots, [w_n]$, run a sequence of instructions of the following types:

- $\mathsf{Random}(\mathbb{F})$ (for $\mathbb{F} \in \{\mathbb{F}_p, \mathbb{F}_{p^k}\}$): Retrieve $[r]$, where $r \in \mathbb{F}$ is the next suitable random value in π.
- $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(x)$: Sends value $x \in \mathbb{F}_p$ from P to V.
- $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(x)$: Sends value $x \in \mathbb{F}_p$ from V to P.
- $[z] = a[x] + b[y] + c$: Define the commitment $[z]$ for $z = ax + by + c$, given some public values a, b, c.
- $\mathsf{AssertZero}([x])$: Asserts to V that $[x]$ is a commitment to $x = 0$.

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

As described previously, by translating AssertZero calls into linear oracle queries, any C&P protocol specified in the above syntax defines a valid IP-LOVe.

4.2 C&P IP with LOVe for Arithmetic Circuits

We now show a C&P IP with LOVe for arithmetic circuit satisfiability that satisfies (1) completeness, (2) soundness, and (3) that all inputs to Send are indistinguishable from random. Thus, by Lemma 1 we conclude that our protocol is also zero knowledge and supports disjunctions. We prove circuit satisfiability over a field $\mathbb{F}_p$, but define a protocol over $\mathbb{F}_{p^k}$ for some $k \geq 1$, so that soundness can be boosted if necessary.

We begin by defining two auxiliary "instructions": (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 AssertZero.

– $\mathsf{Fix}(x) \to [x]$: On input $x \in \mathbb{F}$ (where $\mathbb{F} \in \{\mathbb{F}_p, \mathbb{F}_{p^k}\}$) from P, output a commitment $[x]$. This is implemented as:
  1. $\mathsf{Random}(\mathbb{F}) \to [r]$.
  2. $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(x - r) \to y$.
  3. $[r] + y \to [x]$.
– $\mathsf{Reveal}([x]) \to x$: On input commitment $[x]$, output x to V. This is implemented as:
  1. $\mathsf{Send}_{[\mathsf{P}\to\mathsf{V}]}(x)$.
  2. $\mathsf{AssertZero}([x] - x)$.

Our protocol works as follows. Let $C : \mathbb{F}_p^n \to \mathbb{F}_p$ be a circuit known to both parties consisting of Add and Mult gates, which we want to show evaluates to zero on some input. The prover provides the witness $\boldsymbol{w} \in \mathbb{F}_p^n$ as input, so the parties initially get commitments $[w_1], \ldots, [w_n]$. 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:
   $\mathsf{Add}([x], [y])$: Output $[x] + [y]$.
   $\mathsf{Mult}([x], [y])$: Run $\mathsf{Random}(\mathbb{F}_{p^k}) \to [a]$, $\mathsf{Fix}(xy) \to [z]$, and $\mathsf{Fix}(ay) \to [c]$. Output $[z]$, and store the commitments $[a]$ and $[c]$.
2. Run $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(e)$, where $e \in_R \mathbb{F}_{p^k}$.
3. For each $i \in [|C^*|]$ let $[x_i]$ and $[y_i]$ denote the inputs and $[z_i]$, $[a_i]$ and $[c_i]$ denote the outputs and stored values in the i-th call to Mult. Then run $\mathsf{AssertMult}([x_i], [y_i], [z_i], [a_i], [c_i], e)$.
4. Run $\mathsf{AssertZero}([z_{\mathsf{out}}])$, where $[z_{\mathsf{out}}]$ is the commitment to the output of C.

The subprotocol $\mathsf{AssertMult}$ used above works as follows:

$\mathsf{AssertMult}([x], [y], [z], [a], [c], e)$**:**
1. Run $\mathsf{Reveal}([\varepsilon])$, where $[\varepsilon] = e[x] - [a]$.
2. Run $\mathsf{AssertZero}(e[z] - [c] - \varepsilon[y])$.

Before proving security, observe that the communication complexity is 3 field elements per multiplication gate, of which one is over $\mathbb{F}_p$ (for fixing xy) and two over $\mathbb{F}_{p^k}$ (for fixing ay, and revealing ε).

Theorem 3. *Let $\mathcal{R}$ be a relation that can be represented by an arithmetic circuit C over $\mathbb{F}_p$ such that $\mathcal{R}(x, \boldsymbol{w}) = 1 \Leftrightarrow C(\boldsymbol{w}) = 0$. Then the above protocol is a* C&P *IP with LOVe over $\mathbb{F}_{p^k}$ for $\mathcal{R}$, such that (1) completeness holds, (2) soundness holds with soundness error p^{-k}, (3) all inputs to* Send *are perfectly indistinguishable from random, and (4) the protocol is a proof of knowledge.*

The proof can be found in the full version. At a high level, soundness holds by the security of the Mult operation, where a malicious prover essentially needs to align its invalid Fix values with the verifier's random e value, which happens with probability $1/|\mathbb{F}_{p^k}|$.

4.3 Improved C&P IP with LOVe for Arithmetic Circuits

The protocol from § 4.2 communicates 3 field elements per verified multiplication. We now present an alternative multiplication verification procedure, called AssertMultVec, that builds on a protocol from Boneh et al. [BBC+19][6]. AssertMultVec simultaneously proves n multiplication instances at the cost of communicating $n + O(\log(n))$ $\mathbb{F}$-elements. In particular, for $\mathbb{F}_p$ for $p = 2^{61} - 1$ we require around 64.3 bits of communication per multiplication.

Boneh et al. [BBC+19] introduce a logarithmic-sized proof for "parallel-sum" circuits. In a "parallel-sum" circuit, identical subcircuits C' are evaluated in parallel on possibly different inputs, with the sum of the output of each C' being the output of the overall circuit. The high-level idea then is to embed checks for different instances of C' within a single polynomial, allowing the verifier to verify n instances of C' in parallel. This protocol, when letting C' be a multiplication gate, can then be used to simultaneously verify the sum of n multiplications. We call this protocol AssertDotProduct.

In more detail, the AssertDotProduct protocol works as follows. Suppose P wants to prove that $[z] = \sum_{i\in[n]}[x_i][y_i]$. P begins by defining n polynomials $f_1, \ldots, f_{n/2}, g_1, \ldots, g_{n/2}$ such that $f_i(j) = x_{(j-1)n/2+i}$ and $g_i(j) = y_{(j-1)n/2+i}$, and then computing $h = \sum_{i\in[n/2]} f_i g_i$. P then commits to h by committing to its coefficients (denoted as $[h]$). V defines its own polynomials f'_i, g'_i over the *committed* values $[x_{(j-1)n/2+i}]$ and $[y_{(j-1)n/2+i}]$ to check that $\sum_{i\in[n/2]} f'_i g'_i = h$. By Schwartz-Zippel, this can be done by checking that

$$\sum_{i\in[n/2]} f'_i(r) g'_i(r) = h(r) \tag{1}$$

for a random r chosen by V. Here, observe that the evaluation of f'_i, g'_i, h in a public constant r boils down to multiplying the committed coefficients of each polynomial with appropriate powers of r and summing up the result, both of which are local operations. Then, verifying Equation 1 after fixing r is again a dot product check, although over vectors of length $n/2$, and we can recursively apply AssertDotProduct until $n = 1$. Note that only two $\mathbb{F}_{p^r}$-elements are communicated during one iteration of AssertDotProduct: when committing to h and sending r. See Figure 4 for a formal presentation of the protocol. There, for the base-case of AssertDotProduct, we use the multiplication checking procedure from § 4.2.

Given AssertDotProduct, we can batch-verify n multiplications as follows:

1. Assume that n tuples $[x_i], [y_i], [z_i]$ have been committed by P.
2. V chooses a randomization factor r that it sends to P.
3. P shows that $\langle r^i[x_i], [y_i]\rangle = \sum_{i\in[n]} r^i[z_i]$. Since r is public, computing $r^i[x_i]$ and $\sum_{i\in[n]} r^i[z_i]$ is local.

This protocol, called AssertMultVec, is presented in Figure 4.

[6] This approach was recently used in the context of MPC-in-the-head-based ZK protocols [dSGOT21].

$\boxed{\mathsf{AssertMultVec}([x_1],\ldots,[x_n],[y_1],\ldots,[y_2],[z_1],\ldots,[z_n]) \Rightarrow \forall i\ x_i y_i = z_i}$

1. $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(r)$ for $r \in_R \mathbb{F}_{p^k}\backslash\{0\}$.
2. $\mathsf{AssertDotProduct}(r^1[x_1],\ldots,r^n[x_n],[y_1],\ldots,[y_n],\sum_{i\in[n]} r^i[z_i])$.

$\boxed{\mathsf{AssertDotProduct}([x_1],\ldots,[x_n],[y_1],\ldots,[y_n],[z]) \Rightarrow z = \sum_i x_i y_i}$

If $n \leq 2$:

1. For $i \in [n]$: $\mathsf{Mult}([x_i],[y_i]) \to ([z_i],[a_i],[c_i])$.
2. Run $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(e)$, where $e \in_R \mathbb{F}_{p^k}$.
3. For $i \in [n]$: $\mathsf{AssertMult}([x_i],[y_i],[z_i],[a_i],[c_i],e)$.
4. $\mathsf{AssertZero}(\sum_{i\in[n]}[z_i] - [z])$.

Otherwise:

1. P defines polynomials of least degree $f_1,\ldots,f_{n/2},g_1,\ldots,g_{n/2} \in \mathbb{F}_p[X]$ such that for $j \in [2]$: $f_i(j) = x_{(j-1)n/2+i}$, $g_i(j) = y_{(j-1)n/2+i}$.
   P defines the polynomial $h = \sum_{i\in[n/2]} f_i g_i \in \mathbb{F}_p[X]$. Note that h has degree ≤ 2. Let c_0, c_1, c_2 denote the coefficients of h.
2. For $i \in \{0,1,2\}$: $\mathsf{Fix}(c_i) \to [c_i]$.
3. For $i \in [n/2]$: P and V compute (committed) polynomials of least degree $[f'_i]$ and $[g'_i]$ satisfying for $j \in [2]$: $f'_i(j) = [x_{(j-1)n/2+i}]$, $g'_i(j) = [y_{(j-1)n/2+i}]$.
4. Let $[h']$ be the (committed) polynomial defined by the $[c_i]$ values.
5. $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(r)$ where $r \in_R \mathbb{F}_{p^k}\backslash\{0,1\}$.
6. $\mathsf{AssertZero}(\sum_{i\in[2]}[h'](i) - [z])$.
7. $\mathsf{AssertDotProduct}([f'_1](r),\ldots,[f'_{n/2}](r),[g'_1](r),\ldots,[g'_{n/2}](r),[h'](r))$.

Fig. 4. Protocols for efficient multiplications. See text for necessary notation.

It is clear that both AssertDotProduct and AssertMultVec are complete and zero-knowledge. The follow theorem, proven in the full version, shows they are also sound.

Theorem 4. *If the protocol* AssertMultVec *passes, then the input commitments have the required relation except with probability* $\frac{n+4\log n+1}{p^k-2}$

An alternative version of AssertMultVec with a soundness error that is only logarithmic in n can be achieved as follows:

$\mathsf{AssertMultVec}'([x_1],\ldots,[x_n],[y_1],\ldots,[y_2],[z_1],\ldots,[z_n])$**:**

1. $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(r_1,\ldots,r_n)$ for $r_1,\ldots,r_n \in_R \mathbb{F}_{p^k}$.
2. $\mathsf{AssertDotProduct}(r_1[x_1],\ldots,r_n[x_n],[y_1],\ldots,[y_n],\sum_{i\in[n]} r_i[z_i])$.

One can easily show that $\mathsf{AssertMultVec}'$ has the desired soundness, although at the expense of communicating more random elements from V to P. In practice, one can optimize this by having V choose a random PRG seed that it sends to P, with $r_1,\ldots,r_n$ derived deterministically from the seed.

4.4 C&P IP with LOVe for Binary Circuits

The protocol from § 4.3 is agnostic to the underlying field, so we can use $p = 2$ (and large enough k for soundness) to otain a proof for binary circuits. For $\mathbb{F}_{2^{40}}$ and a batch size of 1 000 000 this requires approximately 1.008 bits per verified AND-gate.

One of the downsides to the batching approach is that it is most efficient for large batches of multiplications. When evaluating a disjunctive branch, however, the size of the batch may be limited by the number of multiplications in the branch. This is because we need to "complete" a batch of multiplications before we can apply the OR-proof. Unfortunately, this smaller batch size increases the per-bit communication cost: as an example, a batch size of 100 requires approximately 10 bits per verified AND-gate.

We now present an alternative approach that can achieve a fixed per-bit communication cost of 9 bits per verified AND-gate. This approach uses *reverse multiplication friendly embeddings* [BMN18,CCXY18] (RFMEs), defined as follows.

Definition 6. *A $(k, m)_p$-*RFME *is a pair (ϕ, ψ) of linear maps $\phi : \mathbb{F}_p^k \to \mathbb{F}_{p^m}$ and $\psi : \mathbb{F}_{p^m} \to \mathbb{F}_p^k$ such that $\boldsymbol{x} * \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}))$, where $*$ denotes pairwise multiplication.*

Cascudo et al. [CCXY18] showed that for $p = 2$ and $r < 33$, there exist $(3r, 10r - 5)_2$-RFMEs. Noting that for efficiency we would like as small a field as possible, alongside the requirement of have a statistical security parameter of at least 40, we use $(15, 45)_2$-RFMEs, and thus work over $\mathbb{F}_{2^{45}}$. Thus, we can verify the multiplication of 15-element binary vectors $[\boldsymbol{x}]$ and $[\boldsymbol{y}]$ at the cost of a single multiplication in $\mathbb{F}_{2^{45}}$ as follows. The parties locally compute $[a] \leftarrow \phi([\boldsymbol{x}])$ and $[b] \leftarrow \phi([\boldsymbol{y}])$, compute $[c] \leftarrow [a] \cdot [b]$ using the multiplication verification protocol over $\mathbb{F}_{2^{45}}$, and finally locally compute $[\boldsymbol{z}] \leftarrow \psi([c])$. This has a per-multiplication cost of 10 bits per multiplication.

We can do slightly better by having the prover provide the verifier an *advice* vector to help compute $[c]$. Let $\boldsymbol{d}$ be a binary vector for the linear bijection $f : \mathbb{F}_2^{15} \times \mathbb{F}_2^{30} \to \mathbb{F}_2^{45}$ such that $f(\boldsymbol{z}, \boldsymbol{d}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$. If $[\boldsymbol{z}]$ is provided by the prover, the verifier can *locally* compute $[c]$ by computing $[\boldsymbol{c}'] \leftarrow f([\boldsymbol{z}], [\boldsymbol{d}])$ and then mapping $[\boldsymbol{c}']$ to its associated element in $\mathbb{F}_{2^{45}}$. The parties can then check that $[c] = [a] \cdot [b]$ as before. Overall this gives a per-bit communication cost of 9 bits. See the full version for more details.

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]. We first show how to stream our IPs with LOVe, at the cost of increased round complexity. Then, we show how IPs with LOVe can be transformed into NIZKs (with VOLE preprocessing)

with the Fiat-Shamir transform. To analyze the soundness of this approach, we define a form of *round-by-round soundness* for IPs with LOVe, similar to Canetti et al. [CCH+19], and show that this is satisfied by our constructions.

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, in addition to requiring a small memory footprint for P and V, we also 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 (§ 2.2), this ensures that the resulting protocol also has constant memory, since each AssertZero can be checked on-the-fly.

Recall that in our protocols for circuit satisfiability, 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.

For the more efficient amortized protocol, this drawback seems inherent, however, we can easily avoid it for the simpler protocol from § 4.2, by checking multiplications on-the-fly using an independent random challenge from V for each multiplication. The change is very simple, and for completeness, shown in the modified multiplication sub-protocol below.

Streaming Mult$([x_i], [y_i])$**:** To evaluate the i-th multiplication gate:

1. Run $\mathsf{Random}(\mathbb{F}_p)$ to get $[r_i]$ and $\mathsf{Random}(\mathbb{F}_{p^k})$ to get $[r'_i], [a_i]$.
2. Run $\mathsf{Fix}(x_i y_i) \to [z_i]$, and $\mathsf{Fix}(a_i y_i) \to [c_i]$.
3. Run $\mathsf{Send}_{[\mathsf{V}\to\mathsf{P}]}(e_i \leftarrow \mathbb{F}_{p^k})$.
4. Run $\mathsf{Reveal}([\varepsilon])$, where $[\varepsilon] = e_i[x_i] - [a_i]$.
5. Run $\mathsf{AssertZero}(e_i[z_i] - [c_i] - \varepsilon_i[y_i])$.

For the soundness of this protocol, following the exact same analysis as in § 4.2, we get a soundness error of p^{-k}, due to the random choice of each challenge e_i. In § 5.3, we show that this protocol also satisfies *round-by-round soundness*, implying that it can be made non-interactive using Fiat-Shamir.

5.2 Batching AssertZero with Constant Memory

Recall from § 2 that often, it is useful to combine all the AssertZero statements (that is, linear oracle queries) of an IP with LOVe into just one check, by batching them together at the end. However, just as with our original circuit evaluation protocol, this is not amenable to constant memory for streaming algorithms.

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

1. P first commits to its inputs as in Π_{LOVe}.
2. Initialize a dummy commitment $[z] := 0$.
3. For the i-th $\mathsf{AssertZero}([\gamma_i])$ instruction in Π_{LOVe}
   - V sends a random challenge $e_i \in \mathbb{F}_{p^k}$.
   - Update $[z] = [z] + e_i[\gamma_i]$.
4. All other instructions are kept the same.
5. At the end of the program, run $\mathsf{AssertZero}([z])$.

Fig. 5. The transformation to batch AssertZero in a streamable manner.

Instead, in Figure 5 we give an alternative transformation, which transforms any C&P IP with LOVe Π_{LOVe} to have just one AssertZero, *without* storing all intermediate values.

The idea is that, instead of taking a combination of all AssertZero's at the end, we can compute this combination in an incremental manner. At each AssertZero on input $[\gamma]$, we take a random challenge e and add $e \cdot [\gamma]$ to a running state $[z]$. At the end of the computation, to verify that all the γ values were zero, we simply run AssertZero on $[z]$. Since the challenge e is only sampled *after* the value being checked for zero was committed, it holds from the argument for the batching method from § 2 that cheating in this check requires guessing a random challenge, so this transformation only increases the soundness error by p^{-k}.

5.3 Round-by-round soundness for IPs with LOVe

The intuition behind the definition of *Round-by-round soundness* is that in any given round of a protocol Π, one can define a function State, which, given the current transcript of Π, outputs a bit indicating whether Π's execution will fail. We require that, if State predicts failure in some round i, then State also predicts failure in round $i+1$, with high probability over the verifier's random challenge.

Previously, round-by-round soundness has been defined for standard interactive proofs [CCH+19], without any form of oracle queries. Below is our modified definition, tailored to IPs with LOVe. Note that unlike the Zero Knowledge setting, where State's inputs are publicly known, here we give the State function also the oracle string $\boldsymbol{\pi}$ as input; without this, there would be no easy way for the State algorithm to simulate whether a given oracle query to $\boldsymbol{\pi}$ would succeed or not. Note also that since we assume Π is public-coin, the oracle query inputs $(\boldsymbol{z}, y)$ are all computable given the complete transcript, so they are also known to State.

Definition 7. *Let Π be a t-round, public-coin* IP-LOVe *for a relation $\mathcal{R}$. We say that Π has* round-by-round soundness error ϵ *if there exists a deterministic (not necessarily efficient) function* State *that takes input an instance x, proof $\boldsymbol{\pi}$ and partial transcript $\mathcal{T}$, and outputs* accept *or* reject*, such that the following properties hold:*

1. *If* $x \notin L$, *then* $\mathsf{State}(x, \boldsymbol{\pi}, \emptyset) = \mathsf{reject}$.
2. *If* $\mathsf{State}(x, \boldsymbol{\pi}, \mathcal{T}) = \mathsf{reject}$ *for a partial transcript* $\mathcal{T}$ *up to round* $h \in [t]$, *then for every potential prover message* $\boldsymbol{a}_h$,

$$\Pr_{\boldsymbol{e}_{h+1} \leftarrow \mathbb{F}^{r^{\mathsf{V}}_{h+1}}} [\mathsf{State}(x, \boldsymbol{\pi}, \mathcal{T} \| \boldsymbol{a}_h \| \boldsymbol{e}_{h+1}) = \mathsf{accept}] \leq \epsilon$$

3. *For any halting transcript* $\mathcal{T}$, *if* $\mathsf{State}(x, \boldsymbol{\pi}, \mathcal{T}) = \mathsf{reject}$ *then* V *rejects.*

Round-by-round soundness of our protocols for circuit satisfiability. In the full version, we show that our IPs with LOVe for circuit satisfiability, including the streamable protocol from § 5.1–5.2, and the efficient batched multiplication protocol from § 4.3 satisfy round-by-round soundness. We also show that the same holds for our stacking protocol from § 3.

Roughly speaking, for our circuit satisfiability protocol, the State algorithm takes as input the proof string π, so can immediately extract the witness and try to verify whether the statement is true. In later rounds, State also checks whether the prover's messages are inconsistent with π and the verifier's challenges, and changes to reject if so. A similar strategy works in all our protocols to show that round-by-round soundness holds.

Soundness of Fiat-Shamir for IPs with LOVe. We now show that the Fiat-Shamir transformation, when applied to a zero-knowledge proof built from VOLE and an IP with LOVe, is sound if the underlying IP with LOVe satisfies round-by-round soundness.

For this, we follow the VOLE-based protocol from § 2.2, while replacing the verifier's random challenges with outputs of a random oracle. We use a slightly augmented VOLE functionality, denoted $\mathcal{F}_{\mathsf{VOLE}+\mathsf{id}}$, which additionally samples a random identifier $\mathsf{id} \in \{0,1\}^\lambda$, and gives this to both parties after receiving their input. We feed this into the random oracle, which binds the statement and proof to this instance. The result we obtain is similar to the FS transform for interactive oracle proofs [BCS16], with the differences that (1) we start from IPs with LOVe using VOLE preprocessing, and (2) we assume round-by-round soundness, which is a stronger property than state-restoration soundness from [BCS16], but we find it simpler to work with. We prove the following theorem in the full version.

Theorem 5. *Let* Π_{LOVe} *be a* t*-round, 1-query, public-coin* $\mathsf{IP}\text{-}\mathsf{LOVe}$ *for relation* $\mathcal{R}$ *with round-by-round soundness* ϵ, *which is also complete and zero-knowledge. Then, the compiled protocol* $\Pi^{\mathsf{VOLE}}_{\mathsf{NIZK}}$ *in Figure 6 is a non-interactive zero-knowledge proof in the* $\mathcal{F}_{\mathsf{VOLE}+\mathsf{id}}$*-hybrid model, with soundness error at most*

$$p^{-k} + \epsilon t + Q(\epsilon + 2/|\mathcal{C}| + 2^{-\lambda})$$

where Q *is the number of random oracle queries made by a malicious prover, and* $|\mathcal{C}|$ *is the size of the smallest challenge set in any given round of* Π.

Furthermore, if Π_{LOVe} *is a proof of knowledge, then so is* $\Pi^{\mathsf{VOLE}}_{\mathsf{NIZK}}$.

Transformation $\Pi_{\mathsf{LOVe}} \to \Pi^{\mathsf{VOLE}}_{\mathsf{NIZK}}$

Let ℓ be the length of the proof string in Π_{LOVe}, the underlying IP with LOVe over $\mathbb{F}_{p^k}$.

1. The parties call $\mathcal{F}^{\ell,p,k}_{\mathsf{VOLE}+\mathsf{id}}$, obtaining random commitments $[r_1], \ldots, [r_\ell]$. Both parties also receive a random identifier $\mathsf{id} \in \{0,1\}^\lambda$.
2. The prover chooses the proof string $\boldsymbol{\pi} \in \mathbb{F}^\ell_{p^k}$ according to Π_{LOVe}, and computes $d_i = r_i - \boldsymbol{\pi}[i]$.
3. Compute the commitments $[\boldsymbol{\pi}[i]] = [r_i] - d_i$. Let $\boldsymbol{\tau}$ be the prover's MACs on $\boldsymbol{\pi}$, and $(\alpha, \boldsymbol{\beta})$ the verifier's keys.
4. P defines the dummy first message $\boldsymbol{a}_0 = \perp$, and challenge $\boldsymbol{e}_1 = \mathsf{H}(x \| \mathsf{id} \| d \| \boldsymbol{a}_0)$.
5. For each round $i = 1, \ldots, t$, P computes its message $\boldsymbol{a}_i$ and the next challenge
$$\boldsymbol{e}_{i+1} = \mathsf{H}(\boldsymbol{a}_i \| \boldsymbol{e}_i)$$
6. For the oracle query $(\boldsymbol{z}, y) \in \mathbb{F}^\ell_{p^k} \times \mathbb{F}_{p^k}$, P computes the MAC
$$\tau^Q = \langle \boldsymbol{\tau}, \boldsymbol{z} \rangle$$
7. P sends the proof $(d_1, \ldots, d_\ell, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_t, \tau^Q)$.
8. V recomputes all the challenges $\boldsymbol{e}_i$, then computes its query verification key
$$\beta^Q = \langle \boldsymbol{\beta}, \boldsymbol{z} \rangle$$
and checks that $\tau^Q = \beta^Q + \alpha \cdot y$. If the check passes, V accepts.

Fig. 6. NIZK from VOLE and IP with LOVe.

6 Implementation and Evaluation

We have implemented the online protocol of Mac'n'Cheese with the batched multiplication approach of § 4.3 in the Rust programming language. Our implementation achieves a computational security of 128 bits and a statistical security of ≥ 40 bits. 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.

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 stackful coroutines to (cheaply) concurrently execute both branches.

Concurrency. Despite running a multi-round interactive protocol (without extensive use of the Fiat-Shamir transform), we are still able to achieve high-performance even over a high-latency, throughput-limited network link. We reach

this result by running one thread which exclusively sends data from the prover to the verifier to Fix MACs to prover-private values. Once this thread has Fixed a batch of MACs, it submits them to one of many background threads to verify the assertions on these MACs. Each background thread has its own unique connection between the prover and the verifier, so can independently wait for a response from the other party. As a result, we can provision a large number of background threads (which will spend most of their time waiting for the network, rather than running computation) to mitigate the latency effects of running our multi-round protocol over a network. In addition, this design allows us to leverage multiple cores independent of the circuit structure.

6.1 Evaluation

We benchmarked our implementation for $\mathbb{F}_{2^{40}}$ (for boolean circuits) and $\mathbb{F}_{2^{61}-1}$ (for arithmetic circuits). Unless otherwise specified, all benchmarks were run between two machines: a laptop (2018 MacBook Pro with 8 logical cores and 16 GB of RAM) on the east coast of the U.S., and a server (40 Intel Xeon Silver 4114 cores operating at 2.20 GHz) on the west coast of the U.S. The network had an average latency of 95 ms and an average bandwidth of 31.5 Mbps. All numbers are the average of at least four runs of the given experiment.

As noted above, these results *do not* include the cost of VOLE. We are in the process of integrating the VOLE protocol of 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 $\mathbb{F}_{2^{61}-1}$ can be generated in 85 ns at a communication cost of 0.42 bits [WYKW20, Table 4], and can be largely precomputed.

As mentioned above, our implementation is multi-threaded, and in order to reduce communication latency we pipeline processing as much as possible. We use 50 threads for all of our experiments, although we note that the CPU utilization on both the prover and verifier never exceeds 226% (where the maximum possible utilization is the number of cores times 100%). We reiterate that our verifier was run on a commodity laptop, and while we use a large number of threads, this does *not* equate to extremely high CPU utilization.

Microbenchmarks. Using a multiplication batch size of 1 000 000, Mac′n′Cheese achieves a per multiplication cost of approximately 144 ns for $\mathbb{F}_{2^{40}}$ and 1.5 μs for $\mathbb{F}_{2^{61}-1}$. This equates to 6.9 million multiplications per second (mmps) for $\mathbb{F}_{2^{40}}$, and 0.6 mmps for $\mathbb{F}_{2^{61}-1}$. We found that the main limiter in the arithmetic case was bandwidth, and thus also ran our microbenchmarks locally (run on the Location B server), achieving a per multiplication cost of 141 ns (7.0 mmps) for $\mathbb{F}_{2^{40}}$ and 276 ns (3.6 mmps) for $\mathbb{F}_{2^{61}-1}$.

Comparison to QuickSilver. We briefly compare to QuickSilver [YSWW21]. Recall that QuickSilver requires only a single field element per multiplication and requires only 3 rounds (cf. Table 1), but does not support communication-optimized disjunctions. When run on localhost within an Amazon EC2 instance,

Branches	Local (seconds)	Verify (seconds)	Comm. Increase (bytes)
1	34	139	—
2	81	307	+25
4	163	568	+50
8	327	1254	+75

Table 2. Performance results for disjunctions. The "Branches" column denotes the number of branches, where each branch contains 1 billion AND gates. The "Local" column denotes the time to locally compute the circuit in-the-clear, and provides a rough lower bound of performance. The "Verify" column denotes the time to verify the ZK proof. The "Comm. Increase" column denotes the amount of communication increase from the baseline of 124 MB required in the single-branch case.

QuickSilver achieves 7.6 mmps for boolean and 4.8 mmps for arithmetic when utilizing 1 thread, and 15.8 mmps for boolean and 8.9 mmps for arithmetic when utilizing 4 threads [YSWW21, Table 2]. While it is hard to make an apples-to-apples comparison here, this does suggest that QuickSilver is slightly faster, albeit at the expense of communication-optimized disjunctions. Thus, the choice of QuickSilver versus Mac′n′Cheese may come down to the characteristics of the input circuit.

Disjunctions. We also explored the effect our disjunction optimization has on the communication cost. We did so by comparing a proof of a boolean circuit containing 1 billion multiplication gates to using a boolean circuit containing two or more branches each containing 1 billion gates[7]. See Table 2 for the results.

The overall communication in all cases was essentially 124 MB: the OR proof added only an additional $25\log(m)$ bytes, where m denotes the number of branches. In terms of overall running time, we see an increase with the overall *size* of the circuit. This is due to the fact that the prover still needs to do the entire computation, and for this particular example bandwidth is *not* the bottleneck. The table also reports the time required to simply *evaluate* the circuit locally—this presents a reasonable lower bound for Mac′n′Cheese. We find that in all cases, Mac′n′Cheese takes less than 4.08× the cost of locally evaluating the circuit.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085. Any opinions,

[7] In more detail, the branched circuit contained one branch computing 150 000 iterations of AES (960 million multiplication gates) and the other branch computing 45 000 iterations of SHA-2 (1.002 billion multiplication gates). The non-branched circuit only ran the SHA-2 portion of the aforementioned circuit.

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*, 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*. 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*. Springer, Heidelberg, October / November 2016.

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.

BMN18. Alexander R. Block, Hemanta K. Maji, and Hai H. Nguyen. Secure computation with constant communication overhead using multiplication embeddings. In *INDOCRYPT 2018*. Springer, Heidelberg, December 2018.

CCH+19. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In *51st ACM STOC*. ACM Press, June 2019.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In *CRYPTO 2018, Part III*. Springer, Heidelberg, August 2018.

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

DIO21. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *Information-Theoretic Cryptography (ITC) 2021*, 2021.

dSGOT21. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPCitH-based arguments. Cryptology ePrint Archive, Report 2021/215, 2021.

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.

HK20a. David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In *CRYPTO 2020, Part II*, LNCS. Springer, Heidelberg, August 2020.

HK20b. 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.

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`.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. Cryptology ePrint Archive, Report 2021/076, 2021.

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.