Stacking Sigmas:
A Framework to Compose Σ-Protocols for Disjunctions

Aarushi Goel[1], Matthew Green[1], Mathias Hall-Andersen[2], and Gabriel Kaptchuk[3]

[1] Johns Hopkins University, Baltimore, USA, {aarushig,mgreen}@cs.jhu.edu,
[2] Aarhus University, Aarhus, Denmark, ma@cs.au.dk
[3] Boston University, Boston, USA, kaptchuk@bu.edu

Abstract. Zero-Knowledge (ZK) Proofs for disjunctive statements have been a focus of a long line of research. Classical results such as Cramer *et al.* [CRYPTO'94] and Abe *et al.* [AC'02] design generic compilers that transform certain classes of ZK proofs into ZK proofs for disjunctive statements. However, communication complexity of the resulting protocols in these results ends up being proportional to the complexity of proving all clauses in the disjunction. More recently, Heath *et al.* [EC'20] exploited special properties of garbled circuits to construct efficient ZK proofs for disjunctions, where the proof size is only proportional to the length of the largest clause in the disjunction. However, these techniques do not appear to generalize beyond garbled circuits.

In this work, we focus on achieving the best of both worlds. We design a *general framework* that compiles a large class of unmodified Σ-protocols, each for an individual statement, into a new Σ-protocol that proves a disjunction of these statements. Our framework can be used both when each clause is proved with the same Σ-protocol and when different Σ-protocols are used for different clauses. The resulting Σ-protocol is concretely efficient and has communication complexity proportional to the communication required by the largest clause, with additive terms that are only logarithmic in the number of clauses.

We show that our compiler can be applied to many well-known Σ-protocols, including classical protocols (*e.g.* Schnorr [JC'91] and Guillou-Quisquater [CRYPTO'88]) and modern MPC-in-the-head protocols such as the recent work of Katz, Kolesnikov and Wang [CCS'18] and the Ligero protocol of Ames *et al.* [CCS'17]. Finally, since all of the protocols in our class can be made non-interactive in the random oracle model using the Fiat-Shamir transform, our result yields the first generic non-interactive zero-knowledge protocol for disjunctions where the communication only depends on the size of the largest clause.

1 Introduction

Zero-knowledge proofs and arguments [26] are cryptographic protocols that enable a prover to convince the verifier of the validity of an NP statement without revealing the corresponding witness. These protocols, along with proof of

knowledge variants, have now become critical in the construction of larger cryptographic protocols and systems. Since classical results established feasibility of such proofs for all NP languages [24], significant effort has gone into making zero-knowledge proofs more practically efficient *e.g.* [9, 10, 13, 28, 31, 34, 35], resulting in concretely efficient zero-knowledge protocols that are now being used in practice [7, 41, 42].

Zero-knowledge for Disjunctive Statements. There is a long history of developing zero-knowledge techniques for *disjunctive statements* [1, 17, 21]. Disjunctive statements comprise of several *clauses* that are composed together with a logical "OR." These statements also include conditional clauses, *i.e.* clauses that would only be relevant if some condition on the statement is met. The witness for such statements consists of a witness for one of the clauses (also called the *active* clause), along with the index identifying the active clause. Disjunctive statements occur commonly in practice, making them an important target for proof optimizations. For example, disjunctive proofs are often also used to give the prover some degree of privacy, as a verifier cannot determine which clause is being satisfied. Use cases include membership proofs (*e.g.* ring signatures [37]), proving the existence of bugs in a large codebase (as explored in [31]), and proving the correct execution of a processor, which is typically composed of many possible instructions, only one of which is executed at a time [8].

An exciting line of recent work has emerged that reduces the communication complexity for proving disjunctive statements to the size of the largest clause in the disjunction [31, 36]. While succinct proof techniques exist [10, 22, 27, 28], known constructions are plagued by very slow proving times and often require strong assumptions, sometimes including trusted setup. These recent works accept larger proofs in order to get significantly faster proving times and more reasonable assumptions — while still reducing the size of proofs significantly. Intuitively, the authors leverage the observation that a prover only needs to honestly execute the parts of a disjunctive statement that pertain to their witness. Using this observation, these protocols modify existing proof techniques, embedding communication-efficient ways to "cheat" for the inactive clauses of the disjunctive statement. We refer to these techniques as *stacking* techniques, borrowing the term from the work of Heath and Kolesnikov [31].

Although these protocols achieve impressive results, designing stacking techniques requires significant manual effort. Each existing protocol requires the development of a novel technique that reduces the communication complexity of a specific base protocol. For instance, Heath and Kolesnikov [31] observe that garbled circuit tables can be additively *stacked* (thus the name), allowing the prover in [34] to *un-stack* efficiently, leveraging the topolgy hiding property of garbling. Techniques like these are tailored to optimize the communication complexity of a particular underlying protocol, and do not appear to generalize well to large families of protocols. In contrast, classical results [1, 17] succeed in designing a generic compiler that tranforms a large familily of zero-knowledge proof systems into proofs for disjunction, but fall short of reducing the size of the resulting proof.

In this work, we take a more general approach towards reducing the communication complexity of zero-knowledge protocols for disjunctive statements. Rather than reduce the communication complexity of a specific zero-knowledge protocol, we investigate *generic* stacking techniques for an important family of zero-knowledge protocols — three round public coin proofs of knowledge, popularly known as Σ-protocols. Specifically, we ask the following question:

Can we design a generic compiler that stacks any Σ-protocol without modification?

We take significant steps towards answering this question in the affirmative. While we do not demonstrate a technique for stacking all Σ-protocols, we present a compiler that *stacks* many natural Σ-protocols, including many of practical importance. We focus our attention on Σ-protocols because of their widespread use and because they can be made non-interactive in the random oracle model using the Fiat-Shamir transform [19]. However we expect that the techniques can easily be generalized to public-coin protocols with more rounds.

Benefits of a Generic Stacking Compiler. There are several significant benefits of developing generic stacking compilers, rather than developing bespoke protocols that support stacking. First, automatically compiling multiple Σ-protocols into ones supporting stacking removes the significant manual effort required to modify existing techniques. Moreover, newly developed Σ-protocols can be used to produce stacked proofs immediately, significantly streamlining the deployment process. A second, but perhaps even more practically consequential, benefit of generic compilers is that protocol designers are empowered to tailor their choice of Σ-protocol to their application — without considering if there are known stacking techniques for that particular Σ-protocol. Specifically, the protocol designer can select a proof technique that fits with the natural representation of the relevant statement (*e.g.* Boolean circuit, arithmetic circuit, linear forms or any other algebraic structure). Without a generic stacking compiler, a protocol designer interested in reducing the communication complexity of disjunctive proofs might be forced to apply some expensive NP reduction to encode the statement in a stacking-friendly way. This is particularly relevant because modern Σ-protocols often require that relations are phrased in a very specific manner, *e.g.* Ligero [2] requires arithmetic circuits over a large, finite field, while known stacking techniques [31] focus on Boolean circuits.

A common concern with applying protocol compilers is that they trade generality for efficiency (*e.g.* NP reductions). However, we note that the compiler that we develop in this work is extremely concretely efficient, overcoming this common limitation. For instance, naïvely applying our protocol to the classical Schnorr identification protocol and applying the Fiat-Shamir [19] heurestic yields a ring signature construction with signatures of length $2\lambda \cdot (2 + 2\log(\ell))$ bits, where λ is the security parameter and ℓ is the ring size; this is actually

smaller than modern ring signatures from similar assumptions [4, 12] without requiring significant optimization. [4]

1.1 Our Contributions.

In this work, we give a generic treatment for minimizing the communication complexity of Σ-protocols for disjunctive statements. In particular, we identify some "special properties" of Σ-protocol that make them amenable to "stacking." We refer to protocols that satisfy these properties as *stackable* protocols. Then we present a framework for compiling any stackable Σ-protocols for independent statements into a new, communication-efficient Σ-protocol for the disjunction of those statements. Our framework only requires oracle access to the prover, verifier and simulator algorithms of the underlying Σ-protocols. We present our results in two-steps:

Self-Stacking Compiler. First, we present our basic compiler, which we call a "self-stacking" compiler. This compiler composes several instances of the *same* Σ-protocol, corresponding to a particular language into a disjunctive proof. The resulting protocol has communication complexity proportional to the communication complexity of a single instance of the underlying protocol. Specifically, we prove the following theorem:

Informal Theorem 1 (Self-Stacking) *Let Π be a stackable Σ-protocol for an NP language $\mathcal{L}$ that has communication complexity $\mathtt{CC}(\Pi)$. There exists is a Σ-protocol for the language $(x_1 \in \mathcal{L}) \vee \ldots \vee (x_\ell \in \mathcal{L})$, with communication complexity $O(\mathtt{CC}(\Pi) + \lambda \log(\ell))$, where λ is the computational security parameter.*

Cross-stacking. We then extend the self-stacking compiler to support stacking *different* Σ-protocols for different languages. The communication complexity of the resulting protocol is a function of the largest clause in the disjunction and the similarity between the Σ-protocols being stacked. Let $f_{\mathtt{CC}}$ be a function that determines this dependence. For instance, if we compose the same Σ-protocol but corresponding to different languages, then the output of $f_{\mathtt{CC}}$ will likely be the same as that of a single instance of that protocol for the language with the largest relation function. However, if we compose Σ-protocols that are very different from each other, then the output of $f_{\mathtt{CC}}$ will likely be larger. We prove the following theorem:

Informal Theorem 2 (Cross-Stacking) *For each $i \in [\ell]$, let Π_i be a stackable Σ-protocol for an NP language $\mathcal{L}_i$ There exists is a Σ-protocol for the language $(x_1 \in \mathcal{L}_1) \vee \ldots \vee (x_\ell \in \mathcal{L}_\ell)$, with communication complexity $O(f_{\mathtt{CC}}(\{\Pi_i\}_{i\in[\ell]}) + \lambda \log(\ell))$.*

[4] Although concrete efficiency is a central element of our work, applying our compiler to applications is not our focus. The details of this ring signature construction can be found in the full version of the paper.

Examples of Stackable Σ-protocols. We show many concrete examples of Σ-protocols that are stackable. Specifically, we look at classical protocols like Schnorr [39], Guillio-Quisquater [29] and Blum [11], and modern MPC-in-the-head protocols like KKW [35] and Ligero [2]. Previously it was not known how to prove disjunction over these Σ-protocols with sublinear communication in the number of clauses. When applied to these Σ-protocols, our compiler yields a Σ-protocol which can can made non-interactive in the random oracle model using the Fiat-Shamir heurestic. For example, when instantiated with Ligero our compiler yields a concretely efficient Σ-protocol for disjunction over ℓ different circuits of size $|C|$ each, with communication $O(\sqrt{|C|} + \lambda \log \ell)$. Additionally, we explore how to apply our cross-stacking compiler to stack different stackable Σ-protocols with one another (*e.g.* stacking a KKW proof for one relation with a Ligero proof for another relation).

Partially-binding non-interactive vector commitments. Central to our compiler is a new variation of commitments called partially-binding non-interactive vector commitment schemes. These schemes allow a committer to commit to a vector of values and equivocate on a subset of the elements in that vector, the positions of which are determined during commitment and are kept hidden. We show how such commitments can be constructed from the discrete log assumption.

Extensions and Implementation Considerations. We finish by discussing extensions of our work and concrete optimizations that improve the efficiency of our compiler when implemented in practice. Specifically, we consider generalizing our work to k-out-of-ℓ proofs of partial knowledge, *i.e.* the threshold analog of disjunctions. We give a version of our compiler that works for these threshold statements. Additionally, we demonstrate the efficiency of our compiler by presenting concrete proof sizes when our compiler is applied to both a disjunction of KKW and Schnorr signatures.

1.2 Related Work

A more in depth overview of these techniques can be found in **??**

Disjunctive Compilers for Zero-Knowledge. The classic work of Cramer *et al.* [17] showed how to compile Σ-protocols into k-of-ℓ disjunctions, but does not provide any communications savings. Abe *et al.* [1] presented an alternative compiler specifically designed for signatures in the random oracle model. More recently, Ciampi *et al.* [15] show how to augment this construction to allow the prover to select instances in the disjunction during the third round. We note that although Ciampi *et al.* make use of a similar commitment scheme, the focus of their work is very different and they do not consider minimizing communication.

Communication Reduction for Disjunctive Zero-Knowledge. In [36], Kolesnikov observed that the topology of a garbled circuit could be decoupled from its tables, resulting in S-universal two party SFE (Secret Function Evaluation). Building on this idea, Heath and Kolesnikov [31] brought the

topology-decoupled paradigm to interactive zero-knowledge, based on the works of Jawurek et. al. [34] and Frederiksen et. al. [20]. This resulted in zero-knowledge with communication complexity proportional to the size of the largest clause in the disjunction, but is inherently interactive. In concurrent work, Baum et. al [5] present Mac'n'Cheese, a constant round zero-knowledge proof system that obtains "free nested disjunctions;" It is not clear how to make these protocols public coin. Heath and Kolesnikov also explored similar ideas to reduce the communication complexity of MPC protocols executed over disjunctions [30, 32].

2 Technical Overview

In this section, we give a detailed overview of the techniques that we use to design a generic framework to achieve communication-efficient disjunctions of Σ-protocols without requiring non-trivial[5] changes to the underlying Σ-protocols. Throughout this work, we consider a disjunction of ℓ *clauses*, one (or more) of which are *active*, meaning that the prover holds a witness satisfying the relation encoded into those clauses. For the majority of this technical overview, we focus on the simpler case where the same Σ-protocol is used for each clause. We will then extend our ideas to cover heterogeneous Σ-protocols.

Recall that Σ-protocols are three-round, public-coin zero-knowledge protocols, where the prover sends the first message. In the second round, the verifier sends a random "challenge" message to the prover, that only depends on the random coins of the the verifier. Finally, in the third round, the prover responds with a message based on this challenge. Based on this transcript the verifier then decides whether to accept or reject the proof.

We start by considering the approaches taken by recent works focusing on privacy-preserving protocols for disjunctive statements, *e.g.* [31]. We observe that the "stacking" techniques used in all these works can be broadly classified as taking a *cheat and re-use approach*. In particular, all of these works show how some existing protocols can be modified to allow the parties to "cheat" on the inactive clauses — *i.e.* only executing the active clause honestly — and "re-using" the single honestly-computed transcript to mimic a fake computation of the inactive clauses. Critically, this is done while ensuring that the verifier cannot distinguish the honest execution of the active clause from the fake executions of the inactive clauses.

Our Approach. In this work we extend the *cheat and re-use* approach to design a framework for compiling Σ-protocols into a communication-efficient Σ-

[5] We assume that basic, practice-oriented optimizations have already been applied to the Σ-protocols in question. For instance, we assume that only the minimum amount of information is sent during the third round of protocol. Hereafter, we will ignore these trivial modifications and simply say "without requiring modification." Note that these modifications truly are trivial: the parties only need to repeat existing parts of the transcript in other rounds. We discuss this in the context of MPC-in-the-head protocols in Section 5.

protocol for disjunctive statements without requiring modification of the underlying protocols. Specifically, we are interested in reducing the number of *third round messages* that a prover must send to the verifier, since the third round message is typically the longest message in the protocol. Intuition extracted from prior work leads us to a natural high-level template for achieving this goal: *Run individual instances of Σ-protocols (one-for each clause in the disjunction) in parallel, such that only one of these instances (the one corresponding to the active clause) is honestly executed, and the remaining instances re-use parts of this honest instance.*

There are two primary challenges we must overcome to turn this rough outline into a concrete protocol: (1) how can the prover cheat on the inactive clauses? and (2) what parts of an honest Σ-protocol transcript can be safely re-used (without revealing the active clause)? We now discuss these challenges, and the techniques we use to overcome them, in more detail.

Challenge 1: How will the prover cheat on inactive clauses? Since the prover does not have a witness for the inactive clauses, the prover can cheat by creating accepting transcripts for the inactive clauses using the simulator(s) of the underlying Σ-protocols. The traditional method (*e.g.* [17] for disjunctive Schnorr proofs) requires the prover to start the protocol by randomly selecting a challenge for each inactive clause and simulating a transcript with respect to that challenge. In the third round, the prover completes the transcript for each clause and demonstrates that it could only have selected the challenges for all-but-one of the clauses. This approach, however, inherently requires sending many third round messages, which will make it difficult to re-use material across clauses (discussed in more detail below). Similarly, alternative classical approaches for composing Σ-protocols for disjunctives, like that of Abe et al. [1], also require sending a distinct third round message for each clause. As such, we require a new approach for cheating on the inactive clauses.

Our first idea is to defer the selection of first round messages for the inactive clauses until after the verifier sends the challenge (*i.e* in the third round of the compiled protocol), while requiring that the prover select a first round message honestly for the active clause (*i.e* in the first round of the compiled protocol). To do this, we introduce a new notion called *non-interactive, partially-binding vector commitments*.[6] These commitments allow the committer to commit to a vector of values and equivocate on a hidden subset of the entries in the vector later on. For instance, a 1-out-of-ℓ binding commitment allows the committer to commit a vector of ℓ values such that that one of the vector positions (chosen when the commitment is computed) is binding, while allowing the committer to modify/equivocate the remaining positions at the time of opening. For a disjunction with ℓ clauses, we can now use this primitive to ensure that the prover computes an honest transcript for at least one of the Σ-protocol instances as follows:

[6] A similar notion for interactive commitments was introduced in [15]. Note that this notion of commitments is very different from the similarly named notion of somewhere statistically binding commitments [18].

- **Round 1:** The prover computes an honest first round message for the Σ-protocol corresponding to the active clause. It commits to this message in the binding location of a 1-out-of-ℓ binding commitment, along with $\ell - 1$ garbage values, and sends the commitment to the verifier.
- **Round 2:** The verifier sends a challenge message for the ℓ instances.
- **Round 3:** The prover honestly computes a third round message for the active clause and then simulates first and third round messages for the remaining $\ell - 1$ clauses. It equivocates the commitment with these updated first round messages, and sends an opening of this commitment along with all the ℓ third round messages to the verifier.

While this is sufficient for soundness, we need an additional property from these partially-binding vector commitments to ensure zero-knowledge. In particular, in order to prevent the verifier from learning the index of the active clause, we require these partially-binding commitments to not leak information about the binding vector position. We formalize these properties in terms of a more general t-out-of-ℓ binding vector commitment scheme, which may be of independent interest, and we provide a practical construction based on the discrete log assumption.[7]

Challenge 2: How will the prover re-use the active transcript? The above approach overcomes the first challenge, but doesn't achieve our goal of reducing the communication complexity of the compiled Σ-protocol. Next, we need to find a way to somehow re-use the honest transcript of the active clause. Our key insight is that for many natural Σ-protocols, it is possible to simulate *with respect to a specific third round message*. That is, it is often easy to simulate an accepting transcript for a given challenge and third round message. This allows the prover to create a transcript for the inactive clauses that share the third round message of the active clause. In order for this compilation approach to work, Σ-protocols must satisfy the following properties (stated here informally):

- *Simulation With Respect To A Specific Third Round Message:* To re-use the active transcript, the prover simulates *with respect to the third round message of the active transcript.* This allows the prover to send a single third round message that can be re-used across all the clauses. More formally, we require that the Σ-protocol have a simulator that can reverse-compute an appropriate first round message to complete the accepting transcript for any given third round message and challenge. While not possible for all Σ-protocols, simulating in this way—i.e., by first selecting a third round message and then "reverse engineering" the appropriate first round message—is actually a common simulation strategy, and therefore possible with most natural Σ-protocols. In order to get communication complexity that only has a logarithmic dependence on

[7] We also explore a construction that is half the size and leverages random oracles in the full-version of this paper [23].

the number of clauses, we additionally require this simulator to be deterministic.[8] We formalize this property in Section 5.
- *Recyclable Third Round Messages:* To re-use third round messages in this way, the distribution of these third round messages must be the same. Otherwise, simulating the inactive clauses would fail and the verifier could detect the active clause used to produce the third round message. Thus, we require that the distribution of third round messages in the Σ-protocol be the same across all statements of interest. We formalize this property in Section 5.

An mentioned before, most natural Σ-protocols satisfy both these properties and we refer to such protocols as *stackable Σ-protocols*. We can compile such Σ-protocols into a communication-efficient Σ-protocol for disjunctions, where the communication only depends on the size of one of the clauses, as follows: Rounds 1 and 2 remain the same as in the protocol sketch above. In the third round, the prover first computes a third round message for the active clause. It then simulates first round messages for the remaining clauses based on the active clause's third round message and the challenge messages. As before, it equivocates the commitment with these updated first round messages.[9] While this allows us to compress the third round messages, we still need to send a vector commitment of the first round messages. In order to get communication complexity that does not depend on the size of all first round messages, the size of this vector commitment should be independent of the size of the values committed. Note that this is easy to achieve using a hash function.

Summary of our Stacking Compiler. Having outlined our main techniques, we now present a detailed description of our compiler for 2 clauses, as depicted in Figure 1 (similar ideas extend for more than 2 clauses). The right (unshaded) box represents the active clause and the left (shaded) box represents the inactive clause. Each of the following numbered steps refer to a correspondingly numbered arrow in the figure: (1) The prover runs the first round message algorithm of the active clause to produce a first round message a_2. (2) The prover uses the 1-of-2 binding commitment scheme to commit to the vector $\mathbf{v} = (0, a_2)$. (3) The resulting commitment constitutes the compiled first round message a'. (4) The challenge c' is created by the verifier. (5) The prover generates the third round message z for the active clause using the first round message a_2, the challenge c', and the witness w. (6) The prover then uses the simulator for the inactive clause on the challenge c' and the honestly generated third round message z to generate a valid first round message for the inactive clause a_1. (7) The prover equivocates on the contents of the commitment a' – replacing 0 with the simulated first round message a_1. The result is randomness r' that can be used to open commitment

[8] We elaborate on the importance of this additional property in the technical sections.

[9] If the simulator computes the first round messages deterministically, then the prover only needs to reveal the randomness used in the commitment in the third round, along with the common third round message to the verifier. Given the third round message, the verifier can compute the first round messages on its own and check if the commitment was valid and that the transcripts verify.

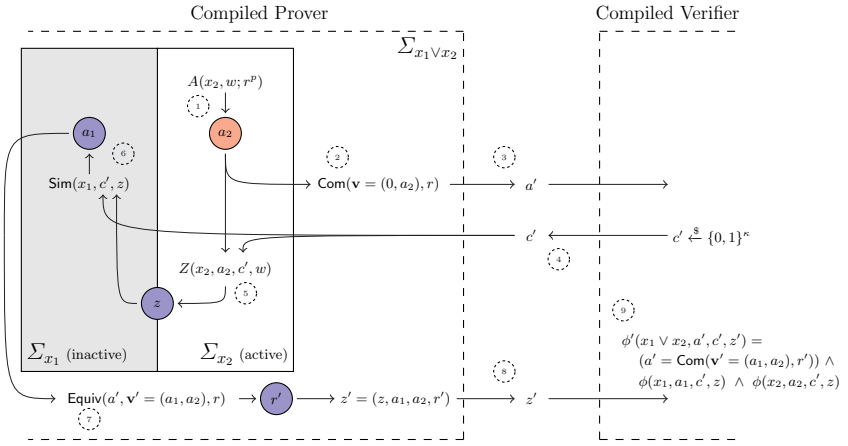

Fig. 1: High level overview of our compiler applied to a Σ-protocol $\Sigma = (A, C, Z, \phi)$ over statements x_1 and x_2. Several details have been omitted or changed to illustrate the core ideas more simply. The red circle contains a value used in the first round, while purple circles contain values used in the third round. We include a_1 and a_2 in the third round message for clarity; in the real protocol, the verifier will be able to deterministically recompute these values on their own.

a' to the vector $\mathbf{v}' = (a_1, a_2)$. (8) The compiled third round message consists of honestly generated third round message z, the randomness r' of the equivocated commitment, and the two first round messages a_1, a_2.[10] (9) The verifier then verifies the proof by ensuring that each transcript is accepting and that the first round messages constitute a valid opening to the commitment a'.

Complexity Analysis: Communication in the first round only consists of the commitment, which we show can be realized in $O(\ell\lambda)$ bits, where λ is the security parameter. In the last round, the prover sends one third round message of the underlying Σ-protocol that depends on the size of one of the clauses[11] and ℓ first round messages of the underlying Σ-protocol. Thus, naïvely applying our compiler results in a protocol with communication complexity $O(\mathtt{CC}(\Sigma) + \ell \cdot \lambda)$, where $\mathtt{CC}(\Sigma)$ is the communication complexity of the underlying stackable Σ-protocol, when executed for the largest clause. In the technical sections, we show that the resulting protocol is itself "stackable", it can be recursively compiled. This reduces the communication complexity to $O(\mathtt{CC}(\Sigma) + \log(\ell) \cdot \lambda)$.

Stackable Σ-Protocols. While not all Σ-protocols are able to satisfy the first two properties that we require, we show that many natural Σ-protocols like

[10] In the compiler presented in the main body, a_1 and a_2 are omitted from the third round message and the verifier recomputes them from z and c' directly. We make this simplification in the exposition to avoid introducing more notation.

[11] We can assume w.l.o.g. that all clauses have the same size. This can be done by appropriately padding the smaller clauses.

Schnorr [38], and Guillio-Quisquater [29] satisfy these properties. We also show that more recent state-of-the-art protocols in MPC-in-the-head paradigm [33] like KKW [35] and Ligero [2] have these properties. We formalize the notion of "$\mathcal{F}$-universally simulatable MPC protocols", which produce stackable Σ-protocols when compiled using MPC-in-the-head [33]. This formalization is highly non-trivial and requires paying careful attention to the distribution of MPC-in-the-head transcripts. Our key observation is that transcripts generated when executing one circuit can often be seamlessly *reinterpreted* as though they were generated for another circuit (usually of similar size). We refer the reader to Section 5 for more details on stackable Σ-protocols.

Stacking Different Σ-Protocols. The compiler presented above allows stacking transcripts for a single Σ-protocol, with a single associated NP language, evaluated over different statements e.g., $(x_1 \in \mathcal{L}) \vee \ldots \vee (x_\ell \in \mathcal{L})$. This is quite limiting and does not allow a protocol designer to select the optimal Σ-protocol for each clause in a disjunction. As such, we explore extending our compiler to support stacking *different* Σ-protocols with different associated NP languages, *i.e.* $(x_1 \in \mathcal{L}_1) \vee (x_2 \in \mathcal{L}_2) \vee \ldots \vee (x_\ell \in \mathcal{L}_\ell)$.

A simple approach would be to rely on NP reductions to define a "meta-language" covering all of $\mathcal{L}_1, \ldots \mathcal{L}_\ell$. Unfortunately, this approach will often result in high concrete overheads. It would be preferable to allow "cross-stacking," or using different Σ-protocols for each clause in the disjunction.[12] The key impediment to applying our self-stacking compiler to different Σ-protocols is that the distribution of third round messages between two different Σ-protocols may be very different. For example, a statement with three clauses may be composed of one Σ-protocol defined over a large, finite field, another operating over a boolean circuit, and a third that is consisting of elements of a discrete logarithm group. Thus, attempting to use the simulator for one Σ-protocol with respect to the third round message of another might result in a domain error; there may be no set of accepting transcripts for the Σ-protocols that share a third round message. As re-using third round messages is the way we reduce communication complexity, this dissimilarity might appear to be insurmountable.

To accommodate these differences, we observe that the extent to which a set of Σ-protocols can be stacked is a function of the similarity of their third round messages. In the self-stacking compiler, these distributions were exactly the same, resulting in a "perfect stacking." With different Σ-protocols, the prover may only be able to re-use a *part* of the third round message when simulating for another Σ-protocol, leading to a "partial stacking." We note, however, that the distributions of common Σ-protocols tend to be quite similar—particularly when seen as an unstructured string of bits. Due to space constraints we include our cross stacking compiler, including case studies on its use, in the full-version of this paper [23].

[12] While it might be possible to define a Σ-protocol that uses different techniques for different parts of the relation, this would require the creation of a new, purpose built protocol — something we hope to avoid in this work. Thus, the difference between self-stacking in this work is primarily conceptual, rather than technical.

Paper Organization. The paper is organized as follows: we present required preliminaries Section 3 and the interface for partially-binding commitment schemes in Section 4. In Section 5 we cover the properties of Σ-protocols that our compiler requires and give examples of conforming Σ-protocols. We present our self-stacking compiler in Section 6.

3 Preliminaries

3.1 Notation

Throughout this paper we use λ to denote the computational security parameter and κ to denote the statistical security parameter. We denote by $x \xleftarrow{\$} \mathcal{D}$ the sampling of 'x' from the distribution '$\mathcal{D}$'. We use $[n]$ as a short hand for a list containing the first n natrual numbers in order: i.e. $[n] = 1, 2 \ldots, n$. We denote by $x \xleftarrow{\$}_s \mathcal{D}$ the process of sampling 'x' from the distribution '$\mathcal{D}$' using pseudo-random coins derived from a PRG applied to the seed 's', when the expression occurs multiple times we mean that the element is sampled using random coins from disjoint parts of the PRG output. We denote by H a collision-resistant hash function (CRH). We write group operations using multiplicative notation.

3.2 Σ-Protocols

In this section, we recall the definition of a Σ-protocol.

Definition 1 (Σ-Protocol). *Let $\mathcal{R}$ be an NP relation. A Σ-Protocol Π for $\mathcal{R}$ is a 3 move protocol between a prover P and a verifier V consisting of a tuple of PPT algorithms $\Pi = (A, Z, \phi)$ with the following interfaces:*

- *$a \leftarrow A(x, w; r^p)$: On input the statement x, corresponding witness w, such that $\mathcal{R}(x, w) = 1$, and prover randomness r^p, output the first message a that P sends to V in the first round.*
- *$c \xleftarrow{\$} \{0,1\}^\kappa$: Sample a random challenge c that V sends to P in the second round.*
- *$z \leftarrow Z(x, w, c; r^p)$: On input the statement x, the witness w, the challenge c, and prover randomness r^p, output the message z that P sends to V in the third round.*
- *$b \leftarrow \phi(x, a, c, z)$: On input the statement x, prover's messages a, z and the challenge c, this algorithm run by V, outputs a bit $b \in \{0, 1\}$.*

A Σ-protocol has the following properties:

- ***Completeness:** A Σ-Protocol $\Pi = (A, Z, \phi)$ is said to be complete if for any x, w such that $\mathcal{R}(x, w) = 1$, and any prover randomness $r^p \xleftarrow{\$} \{0,1\}^\lambda$, it holds that,*

$$\Pr\left[\phi(x, a, c, z) = 1 \;\middle|\; a \leftarrow A(x, w; r^p); c \xleftarrow{\$} \{0,1\}^\kappa; z \leftarrow Z(x, w, c; r^p)\right] = 1$$

– ***Special Soundness.*** *A Σ-Protocol $\Pi = (A, Z, \phi)$ is said to have special soundness if there exists a PPT extractor $\mathcal{E}$, such that given any two transcripts (x, a, c, z) and (x, a, c', z'), where $c \neq c'$ and $\phi(x, a, c, z) = \phi(x, a, c', z') = 1$, it holds that*

$$\Pr\left[R(x, w) = 1 |\ w \leftarrow \mathcal{E}(1^\lambda, x, a, c, z, c', z')\right] = 1$$

– ***Special Honest Verifier Zero-Knowledge.*** *A Σ-Protocol $\Pi = (A, Z, \phi)$ is said to be special honest verifier zero-knowledge, if there exists a PPT simulator $\mathcal{S}$, such that for any x, w such that $\mathcal{R}(x, w) = 1$, it holds that*

$$\{(a, z) \mid c \xleftarrow{\$} \{0,1\}^\kappa; (a, z) \leftarrow \mathcal{S}(1^\lambda, x, c)\} \approx_c$$
$$\{(a, z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x, w; r^p); c \xleftarrow{\$} \{0,1\}^\kappa; z \leftarrow Z(x, w, c; r^p)\}$$

4 Partially-Binding Vector Commitments

In this section, we introduce *non-interactive partially-binding vector commitments*. These commitments allow a committer to commit to a vector of ℓ elements such that exactly t positions are binding (*i.e.* cannot be opened to another value) and the remaining $\ell - t$ positions can be equivocated. The committer must decide the binding positions of the vector before committing and the binding positions are hidden.

Definition 2 (t-out-of-ℓ Binding Vector Commitment). *A t-out-of-ℓ binding non-interactive vector commitment scheme with message space $\mathcal{M}$, is defined by a tuple of the PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{EquivCom}, \mathsf{Equiv}, \mathsf{BindCom})$ *defined as follows:*

– $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *On input the security parameter λ, the setup algorithm outputs public parameters* pp.
– $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B)$*: Takes public parameters* pp *and a t-subset of indices $B \in \binom{[\ell]}{t}$. Returns a commitment key* ck *and equivocation key* ek.
– $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}; r)$*: Takes public parameter* pp*, equivocation key* ek*, ℓ-tuple $\mathbf{v}$ and randomness r. Returns a partially-binding commitment* com *as well as some auxiliary equivocation information* aux.
– $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}, \mathbf{v}', \mathsf{aux})$*: Takes public parameters* pp*, equivocation key* ek*, original commitment value $\mathbf{v}$ and updated commitment values $\mathbf{v}'$ with $\forall i \in B : \mathbf{v}_i = \mathbf{v}'_i$, and auxiliary equivocation information* aux*. Returns equivocation randomness r.*
– $\mathsf{com} \leftarrow \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}; r)$*: Takes public parameters* pp*, commitment key* ck*, ℓ-tuple $\mathbf{v}$ and randomness r and outputs a commitment* com*. Note that this algorithm does not use the equivocation key* ek*. This algorithm plays a similar role to that of* Open *in a typical commitment scheme.*

The properties satisfied by the above algorithms are as follows:

(Perfect) Hiding: *The commitment key* ck *(perfectly) hides the binding positions* B *and commitments* com *(perfectly) hide the* ℓ *committed values in the vector. Formally, for all* $\mathbf{v}^{(1)}, \mathbf{v}^{(2)} \in \mathcal{M}^\ell$*,* $B, B' \in \binom{[\ell]}{t}$*, and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*:*

$$\left[(\mathsf{ck}, \mathsf{com}) \;\middle|\; \begin{array}{c}(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B); r \xleftarrow{\$} \{0,1\}^\lambda; \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}; r)\end{array}\right]$$
$$\stackrel{p}{=} \left[(\mathsf{ck}', \mathsf{com}') \;\middle|\; \begin{array}{c}(\mathsf{ck}', \mathsf{ek}') \leftarrow \mathsf{Gen}(\mathsf{pp}, B'); r' \xleftarrow{\$} \{0,1\}^\lambda \\ (\mathsf{com}', \mathsf{aux}') \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}', \mathbf{v}'; r');\end{array}\right]$$

(Computational) Partial Binding: *It is intractable for an adversary that generates the commitment key* ck *to equivocate on more than* $\ell - t$ *positions. To formalize this property, we consider the class of adversaries* $\mathcal{A}_k$ *that produces a single commitment key* ck *and* k *equivocations to a commitment under that key. We denote the* j^{th} *opening to the commitment as the vector* $\mathbf{v}^{(j)}$ *and the* i^{th} *element of that vector as* $v_i^{(j)}$*. Formally, for all* $k \in \mathrm{poly}(\lambda)$*, for all PPT algorithms* $\mathcal{A}_k$*:*

$$\Pr\left[\begin{array}{c}\nexists S \subset [\ell], |S| \geq t, \text{ s.t. } i \in S, v_i^{(1)} = \ldots = v_i^{(k)} \;\wedge \\ \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}^{(1)}; r_1) = \ldots = \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}^{(k)}; r_k)\end{array} \;\middle|\; \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathsf{ck}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(k)}, r_1, \ldots, r_k) \leftarrow \mathcal{A}_k(1^\lambda, \mathsf{pp})\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Partial Equivocation: *Given a commitment to* $\mathbf{v}$ *under a commitment key* $\mathsf{ck} \leftarrow \mathsf{Gen}(\mathsf{pp}, B)$*, it is possible to equivocate to any* $\mathbf{v}'$ *as long as* $\forall i \in B : v_i = v_i'$*. More formally, for all* $B \in \binom{[\ell]}{t}$*, and all* $\mathbf{v}, \mathbf{v}' \in \mathcal{M}^\ell$ *st.* $\forall i \in B : v_i = v_i'$ *then:*

$$\Pr\left[\mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}'; r') = \mathsf{com} \;\middle|\; \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); r \xleftarrow{\$} \{0,1\}^\lambda; \\ (\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B); \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}; r); \\ r' \leftarrow \mathsf{Equiv}(\mathsf{ek}, \mathbf{v}, \mathbf{v}', \mathsf{aux})\end{array}\right] = 1$$

Throughout this work we will impose the efficiency requirement that the size of the commitment is independent of the size of the elements. We note that this is easy to achieve using a collision resistant hash function, when targeting computational binding.

4.1 Partially-Binding Vector Commitments from Discrete Log

We now present a simple and concretely efficient construction of t-out-of-ℓ partially-binding vector commitments from the discrete log assumption. The idea is to have the committer use a Pedersen commitment for each element in the vector. Recall that a Pedersen commitment to the message $m \in \mathbb{Z}_{|\mathbb{G}|}$

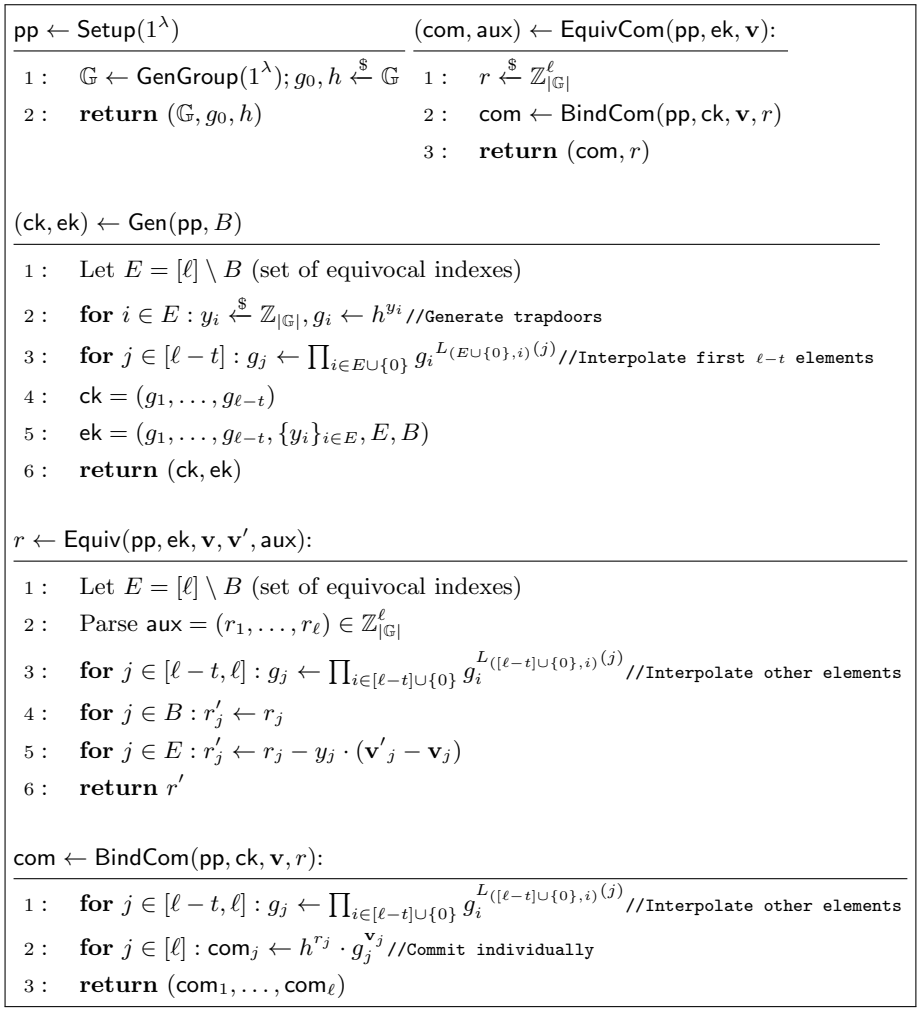

Fig. 2: t-of-ℓ binding commitment from discrete log in the CRS model.

with public parameters $g, h \in \mathbb{G}$ is computed as $g^m h^r$ for a random value r. The binding property of Pedersen commitments relies on the committer not knowing the discrete log of g with respect to h. For our partially-binding vector commitment scheme, the commitment key is a set of public parameters for the Pedersen commitments, constructed such a way that the committer knows discrete logs for exactly $\ell - t$ parameters. This is done by having the committer pick $\ell - t$ of the parameters and computing the remaining t parameters by interpolating in the exponent. More formally, let use begin by fixing some notation. Let $\mathbb{Z}_{|\mathbb{G}|}$ be a prime field. In our construction, we implicitly treat indexes $i \in [0, |\mathbb{G}| - 1]$ as field elements, *i.e.* there is an implicit bijective map between

$[0, |\mathbb{G}| - 1]$ and $\mathbb{Z}_{|\mathbb{G}|}$ (e.g. $i \mod |\mathbb{G}| \in \mathbb{Z}/(|\mathbb{G}|)$). Let $\mathcal{X} \subseteq \mathbb{Z}_{|\mathbb{G}|}$ and $j \in \mathcal{X}$, define $L_{(\mathcal{X},j)}(X) \coloneqq \prod_{m \in \mathcal{X}, m \neq j} \frac{X-m}{j-m} \in \mathbb{Z}_{|\mathbb{G}|}[X]$ i.e. the unique degree $|\mathcal{X}| - 1$ polynomial for which $\forall x \in \mathcal{X} \setminus \{j\} : L_{(\mathcal{X},j)}(x) = 0$ and $L_{(\mathcal{X},j)}(j) = 1$. The formal description of the commitment scheme can be found in Figure 2. While our construction does require a CRS, we note that the CRS is just two randomly selected group elements[13], which in practice can be generated by hashing a 'nothing-up-by-sleeve' constant to the curve by using a cryptographic hash function.

Theorem 1. *Under the discrete log assumption, for any (t, ℓ) with $t < \ell$: the scheme shown in Figure 2 is a family of (perfectly hiding, computationally binding) t-of-ℓ partially binding commitment schemes.*

The security reduction is straightforward and tight: for each position i in which the adversary $\mathcal{A}$ manages to equivocate we can extract the discrete log of g_i (as for regular Pedersen commitments), if we extract the discrete log in $\ell - t + 1$ positions, we have sufficient points on the degree $\ell - t$ polynomial to recover $f_{[\ell] \cup \{0\}}(X)$ explicitly and simply evaluate it at 0 to recover the discrete log of g_0 from pp. The full proof can be found in the full-version of this paper [23].

Remark 1. To commit to longer strings a collision resistant hash $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_{|\mathbb{G}|}$ is used to compress each coordinate before committing using $\mathsf{BindCom}/\mathsf{EquivCom}$ as a black-box: by committing to $\mathbf{v}' = (\mathsf{H}(v_1), \ldots, \mathsf{H}(v_\ell))$ instead. Note that the discrete log assumption, used above, also implies the existance of collision resistant hash functions.

5 Stackable Σ-Protocols

In this section, we present the properties of Σ-protocols that our stacking framework requires and show that many Σ-protocols satisfy these properties.

5.1 Well-Behaved Simulators

As outlined in Section 2, a critical step of our compilation framework is applying the simulator of the underlying Σ-protocols to the inactive clauses. This raises a technical (mostly definitional) concern: some of inactive clauses may not actually be true, possibly because they were adversarially chosen. At first glance, this might seem like a strange concern. For most interesting NP languages of interest, it should be hard to tell if an instance is in the language, and therefore having false instances in the disjunction should not be a problem. However, the behavior of a simulator is only defined with respect to statements that are in the NP language—that is, true instances. As such, if the disjunction contains false clauses, there is no guarantee that the simulator will produce an accepting transcript. This could cause problems with verification—the verifier will know

[13] Like regular Pedersen commitments

that one of the transcripts is not accepting, but will not know if this is due to a simulation failure or malicious prover. As such, we must carefully consider what simulators will produce when executed on a false instance.

As noted in [25], simulators commonly constructed in proofs of the zero-knowledge property will *usually* output accepting transcripts when executed on these false instances. If the simulator were able to consistently output non-accepting transcripts for false instances, it could be used to decide the NP language in polynomial time. However, it is possible to define a valid simulator that produces an output that is not an accepting transcript with non-negligible probability e.g. (1) the input instance is trivially false (*e.g.* a fully connected graph with 4 nodes is not 3-colorable), or (2) the simulator has a hard-coded set of false instances on which it deviates from its normal behavior. Indeed, a probabilistic simulator may also output a non-accepting transcript in each of these cases only occasionally, possibly depending on the challenge. This behavior will not compromise zero-knowledge, but could result in *correctness or soundness* errors.

We emphasize that this is a corner case: commonly constructed simulators will most likely produce accepting transcripts even on false instances, unless the instance is trivially false or not in the domain of the simulator. Nevertheless, we observe that any Σ-protocol can be generically transformed into one that has a simulator that outputs accepting transcripts for all statements. We refer to such simulators as *well-behaved* simulators. We give a formal definition for well-behaved simulators and present the transformation in the full version [23].

5.2 Properties of Stackable Σ-Protocols.

We now formalize the definition of a "stackable" Σ-protocol. As discussed in Section 2, a Σ-protocol is stackable (meaning, it can be used by our stacking framework), if it satisfies two main properties: (1) simulation with respect to a specific third round message, and (2) recyclable third round messages.

Cheat Property: "Extended" Honest Verifier Zero-Knowledge. We view "simulation with respect to a specific third round message" as a natural strengthening of the typical special honest verifier zero-knowledge property of Σ-protocols. At a high level, this property requires that it is possible to design a simulator for the Σ-protocol by first sampling a random third round message from the space of admissible third round messages, and then constructing the unique appropriate first round message. We refer to such a simulator as an *extended simulator*. A similar notion is considered by Abe *et. al* [1] in their definition of `type-T` signature schemes: a `type-T` signature scheme is essentially the Fiat-Shamir [19] heuristic applied to an EHVZK Σ-protocol.

Definition 3 (EHVZK Σ-Protocol). *Let $\Pi = (A, Z, \phi)$ be a Σ-protocol for the NP relation $\mathcal{R}$, with a well-behaved simulator. We say that Π is "extended honest-verifier zero-knowledge (EHVZK)" if there exists a polynomial time computable <u>deterministic</u>* "extended simulator" $\mathcal{S}^{\text{EHVZK}}$ such that for any $(x, w) \in \mathcal{R}$

and $c \in \{0,1\}^\kappa$, there exists an efficiently samplable distribution $\mathcal{D}^{(z)}_{x,c}$ such that:

$$\left\{(a,c,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \approx \left\{(a,c,z) \mid z \xleftarrow{\$} \mathcal{D}^{(z)}_{x,c}; a \leftarrow \mathcal{S}^{\text{EHVZK}}(1^\lambda,x,c,z)\right\}$$

The natural variants (perfect/statistical/computational) of EHVZK are defined depending on which class of distinguishers for which $\approx$ is defined.

Observation 1 (All Σ-protocols can be made EHVZK.) *In the full version of this paper, we present a transformation that transforms any Σ-protocol into a EHVZK Σ-protocol.*

Re-use Property: Recyclable Third Round Messages. The next property that our stacking compilers require is that the distribution of third round messages does not significantly rely on the statement. In more detail, given a fixed challenge, the distribution of possible third round messages for any pair of statements in the language are indistinguishable from each other. We formalize this property by using $\mathcal{D}^{(z)}_c$ to denote a single distribution with respect to a fixed challenge c. We say that a Σ-protocol has recyclable third round messages, if for any statement x in the language the distribution of all possible third round messages corresponding to challenge c is indistinguishable from $\mathcal{D}^{(z)}_c$. We now formally define this property:

Definition 4 (Σ-Protocol with Recyclable Third Messages). *Let $\mathcal{R}$ be an NP relation and $\Pi = (A,Z,\phi)$ be a Σ-protocol for $\mathcal{R}$, with a well-behaved simulator. We say that Π has* recyclable third messages *if for each $c \in \{0,1\}^\kappa$, there exists an efficiently sampleable distribution $\mathcal{D}^{(z)}_c$, such that for all instance-witness pairs (x,w) st. $\mathcal{R}(x,w) = 1$, it holds that*

$$\mathcal{D}^{(z)}_c \approx \left\{z \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\}.$$

This property is fundamental to stacking, as it means that the contents of the third round message do not 'leak information' about the statement used to generate the message. This means that the message can be safely re-used to generate transcripts for the non-active clauses and an adversary cannot detect which clause is active.[14] Although this property might seem strange, we will later show that many natural Σ-protocols have this property.

Stackability With our two-properties formally defined, we are now ready to present the definition of stackable Σ-protocols:

Definition 5 (Stackable Σ-Protocol). *We say that a Σ-protocol $\Sigma = (A,Z,\phi)$ is* stackable*, if it is EHVZK (see Definition 3) and has recyclable third messages (see Definition 4).*

[14] We further elaborate on this in Remark 2.

We now note a useful property of stackable Σ-protocols that follow directly from Definition 5:

Remark 2. Let $\Sigma = (A, Z, \phi)$ be a *stackable* Σ-protocol for the NP relation $\mathcal{R}$, with a well-behaved simulator. Then for each $c \in \{0,1\}^\lambda$ and any instance-witness pair (x, w) with $\mathcal{R}(x, w) = 1$, an honestly computed transcript is computationally indistinguishable from a transcript generated by sampling a random third round message from $\mathcal{D}_c^{(z)}$ and then simulating the remaining transcript using the extended simulator. More formally,

$$\left\{(a,z) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; a \leftarrow A(x,w;r^p); z \leftarrow Z(x,w,c;r^p)\right\} \\ \approx \left\{(a,z) \mid z \xleftarrow{\$} \mathcal{D}_c^{(z)}; a \leftarrow \mathcal{S}^{\text{EHVZK}}(1^\lambda, x, c, z)\right\}$$

Looking ahead, these observations will be critical in proving security of our compilers.

5.3 Classical Examples of Stackable Σ-Protocols

In this section, we show some examples of classical Σ-protocols which are stackable. Rather than considering multiple classical Σ-protocols like Schnorr and Guillou-Quisquater separately, we consider the generalization of these protocols as explored in [16]. Once we show that this generalization is stackable, it is simple to see that specific instantiations are also stackable.

Lemma 1 (Σ-protocol for ψ-preimages [16] is stackable). *Let $\mathfrak{G}_1^*$ and $\mathfrak{G}_2^*$ be groups with group operations $*_1, *_2$ respectively (multiplicative notation) and let $\psi : \mathfrak{G}_1^* \to \mathfrak{G}_2^*$ be a one-way group-homomorphism. Recall the simple Σ-protocol (Π_ψ) of Cramer and Damgård [16] for the relation of preimages $\mathcal{R}_\psi(x, w) \coloneqq x \stackrel{?}{=} \psi(w)$, where $x \in \mathfrak{G}_2^*, w \in \mathfrak{G}_1^*$. The protocol is a generalization of Schnorr [39] and works as follows:*

- *$A(x, w; r^p)$, the prover samples $r \xleftarrow{\$} \mathfrak{G}_1^*$ and sends the image $a = \psi(r) \in \mathfrak{G}_2^*$ to the verifier.*
- *$Z(x, w, c; r^p)$, the prover intreprets c as an integer from a subset $C \subseteq \mathbb{Z}$ and replies with $z = w^c *_1 r$*
- *$\phi(x, a, c, z)$, the verifier checks $\psi(z) = x^c *_2 a$.*

Completeness follows since ψ is a homomorphism: $\psi(z) = \psi(w^c *_1 r) = \psi(w)^c *_2 \psi(r) = x^c *_2 a$. The knowledge soundness error is $1/|C|$ (see [16] for more details). For any homomorphism ψ, Π_ψ is stackable:

Proof. To see that Π_ψ is stackable, define an extended simulator and check for recyclable third messages:

1. *Π_ψ is EHVZK: Let $\mathcal{D}_{x,c}^{(z)} \coloneqq \{z \mid z \xleftarrow{\$} \mathfrak{G}_1^*\}$, let $\mathcal{S}^{\text{EHVZK}}(1^\lambda, x, c, z) \coloneqq \psi(z) *_2 x^{-c}$*

2. *Π_ψ has recyclable third messages: Observe that* $\forall x_1, x_2 : \mathcal{D}^{(z)}_{x_1,c} = \mathcal{D}^{(z)}_{x_2,c} = \mathcal{U}(\mathfrak{G}_1^*)$[15].

Remark 3. The following variants of Π_ψ (with different choices of $\mathfrak{G}_1^*, \mathfrak{G}_2^*, \psi$) are captured in this generalization (along with other similar Σ-protocols):

(1) Guillou-Quisquater [29] (e-roots in an RSA group) for which $\mathfrak{G}_1^* = \mathfrak{G}_2^* = \mathbb{Z}_n^*$ for a semi-prime $n = pq$, $C = [0, e)$ and $\psi(w) \coloneqq w^e$ for some prime $e \in \mathbb{N}$.
(2) Schnorr [39] (knowledge of discrete log): for which $\mathfrak{G}_1^* = \mathbb{Z}^+_{|\mathbb{G}|}, \mathfrak{G}_2^* = \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of prime order $|\mathbb{G}|$, $C = [0, |\mathbb{G}|)$ and $\psi(w) \coloneqq g^w$ for some $g \in \mathbb{G}$.
(3) Chaum-Pedersen [14] (equality of discrete log): for which $\mathfrak{G}_1^* = \mathbb{Z}^+_{|\mathbb{G}|}, \mathfrak{G}_2^* = \mathbb{G} \times \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of prime order $|\mathbb{G}|$, $C = [0, |\mathbb{G}|)$ and $\psi : \mathbb{Z}_{|\mathbb{G}|} \to \mathbb{G} \times \mathbb{G}$, $\psi(w) \coloneqq (g_1^w, g_2^w)$ for $g_1, g_2 \in \mathbb{G}$.
(4) Attema-Cramer [3] (opening of linear forms): for which $\mathfrak{G}_1^* = \mathbb{Z}^\ell_{|\mathbb{G}|} \times \mathbb{Z}_{|\mathbb{G}|}$, $\mathfrak{G}_2^* = (\mathbb{Z}_{|\mathbb{G}|}, \mathbb{G})$, $C = [0, |\mathbb{G}|)$ and $\psi((\mathbf{x}, \gamma)) \coloneqq (L(\mathbf{x}), \mathbf{g}^{\mathbf{x}} h^\gamma)$ for some linear form $L(\mathbf{x}) = \langle \mathbf{x}, \mathbf{s} \rangle$, $\mathbf{s} \in \mathbb{Z}^\ell_{|\mathbb{G}|}$

In the full version of this paper, we also show that Blum's classic 3 move protocol [11] for graph Hamiltonicity is stackable.

5.4 Examples of Stackable "MPC-in-the-Head" Σ-Protocols

We now proceed to show that many natural "MPC-in-the-head" style [33] Σ-protocols (with minor modifications) are stackable. MPC-in-the-head (henceforth refereed to as IKOS) is a technique used for designing three-round, public-coin, zero-knowledge proofs using MPC protocols. At a high level, the prover emulates execution of an n-party MPC protocol Π virtually, on the relation function $\mathcal{R}(x, \cdot)$ using the witness w as input of the parties, and commits to the views of each party. An honest verifier then selects a random subset of the views to be opened and verifies that those views are consistent with each other and with an honest execution, where the output of Π is 1.

Achieving EHVZK. Since the first round messages in such protocols only consist of commitments to the views of all virtual partials, a subset of which are opened in the third round, a natural simulation strategy when proving zero-knowledge of such protocols is the following: (1) based on the challenge message, determine the subset of parties whose views will need be opened later, (2) imagining these as the "corrupt" parties, use the simulator of the MPC protocol to simulate their views, and, finally, (3) compute commitments to these simulated views for this subset of the parties and commitments to garbage values for the remaining virtual parties. Clearly, since the first round messages in this simulation strategy are computed after the third round messages, *these protocols are naturally EHVZK.*

[15] Uniform distribution over $\mathfrak{G}_1^*$.

Achieving recyclable third messages. To show that these Σ-protocols have recyclable third messages, we observe that in many MPC protocols, an *adversary's view can often be condensed and decoupled from the structure of the functionality/circuit* being evaluated. We elaborate this point with the help of an example protocol — semi-honest BGW [6].

Recall that in the BGW protocol, parties evaluate the circuit in a gate-by-gate fashion on secret shared inputs[16] as follows: (1) for addition gates, the parties locally add their own shares for the incoming wire values to obtain shares of the outgoing wire values. (2) For multiplication gates, the parties first locally multiply their own shares for the incoming wire values and then secret share these multiplied share amongst the other parties. Each party then locally reconstructs these "shares of shares" to obtain shares of the outgoing wire values. (3) Finally, the parties reveal their shares for all the output wires in the circuit to all other parties and reconstruct the output.

By definition, the view of an adversary in any semi-honest MPC protocol is indistinguishable from a view simulated by the simulator with access to the corrupt party's inputs and the protocol output. Therefore, to understand the view of an adversary in this protocol, we recall the simulation strategy used in this protocol:

1. For each multiplication gate in the circuit, the simulator sends random values on behalf of the honest parties to each of the corrupt parties.
2. For the output wires, based on the messages sent to the adversary in the previous step and the circuit that the parties are evaluating, the simulator first computes the messages that the corrupt parties are expected to send to the honest parties. It then uses these messages and the output of protocol to simulate the messages sent by the honest parties to the adversary. Recall that this can be done because these messages correspond to the shares of these parties for the output wire values, and in a threshold secret sharing scheme, the shares of an adversary and the secret, uniquely define the shares of the remaining parties.

Observe that the computation done by the simulator in the first part is independent of the actual circuit or function being computed (it only depends on the number of multiplication gates in the circuit). We refer to the messages computed in (1) and the inputs of the corrupt parties as the *condensed view* of the adversary. Additionally, given these simulated views, the output of the protocol, and the circuit/functionality, the simulated messages of the honest parties in (2) can be computed deterministically. Looking ahead, because the output of relation circuits — the circuits we are interested in simulating — should always be 1 to convince the verifier, this deterministic computation will be straight forward. Since the condensed view is not dependent on the function being computed, it can be used with "any" functionality in the second step to compute the remaining view of the adversary. In other words, given two arithmetic circuits with the

[16] These shares are computed using some threshold secret sharing scheme, e.g., Shamir's polynomial based secret sharing [40].

same number of multiplication gates, the condensed views of the adversary in an execution of the BGW protocol for one of the circuits can be re-interpreted as their views in an execution for the other one. We note that circuits can always be "padded" to be the same size, so this property holds more generally.

As a result, for IKOS-style protocols based on such MPC protocols, while some strict structure must be imposed upon third round messages (which are views of a subset of virtual parties) when *verifying* that they have been generated correctly, the third round messages themselves can simply consist of these condensed views (and not correspond to any particular functionality) and hence can be re-used. To make this work, we must make a slight modification to the IKOS compiler. As before, in the first round, the prover will commit to the views (where they are associated with a given function f) of all parties in the first round. However, in the third round, the prover can simply send the condensed views of the opened parties to the verifier. The verifier can deterministically compute the remaining view of these parties w.r.t. the appropriate relation function f and check if they are consistent amongst each other and with the commitments sent in the first round. Since the third round messages in this protocol are not associated with any function, it is now easy to see that they can be the distribution of these messages is independent of the instance.

Building on this intuition, we show that many natural MPC protocols produce stackable Σ-protocols for circuits of the same size when used with the IKOS compiler. Before giving a formal description of the required MPC property, we recall the IKOS compiler in more detail, assuming that the underlying MPC protocol has the following three-functions associated with it: $\mathsf{ExecuteMPC}$ emulates execution of the protocol on a given function with virtual parties and outputs the actual views of the parties, $\mathsf{CondenseViews}$ takes the views of a subset of the parties as input and outputs their condensed views, and $\mathsf{ExpandViews}$ takes the condensed views of a subset of the parties and returns their actual view w.r.t. a particular function.

IKOS Compiler. Let $f = \mathcal{R}(x, \cdot)$. In the first round, the prover runs $\mathsf{ExecuteMPC}$ on f and the witness w to obtain views of the parties and commits to each of these views. In the second round, the verifier samples a random subset of parties as its challenge message. Size of this subset is equal to the maximal corruption threshold of the MPC protocol. In the third round, the prover uses $\mathsf{CondenseViews}$ to obtain condensed views for this subset of parties and sends them to the verifier along with the randomness used to commit to the original views of these parties in the first round. The verifier runs $\mathsf{ExpandViews}$ on f and the condensed views received in the third round to obtain the corresponding original views. It checks if these are consistent with each other and are valid openings to commitments sent in the first round. Depending on the corruption threshold and the security achieved by the underlying MPC protocol, the above steps might be repeated a number of times to reduce the soundness error. Below we restate the main theorem from [33], which also trivially holds for our modified variant.

Theorem 2 (IKOS [33]). *Let $\mathcal{L}$ be an NP language, $\mathcal{R}$ be its associated NP-relation and $\mathcal{F}$ be the function set $\{\mathcal{R}(x,\cdot) : \forall x \in \mathcal{L}\}$. Assuming the existence of non-interactive commitments, the above compiler transforms any MPC protocol for functions in $\mathcal{F}$ into a Σ-protocol for the relation $\mathcal{R}$.*

Next, we formalize the main property of MPC protocols that facilitates in achieving recyclable third messages when compiled with the above IKOS compiler. We characterize this property w.r.t. a function set $\mathcal{F}$, and require the MPC protocol to be such that the condensed views can be expanded for any $f \in \mathcal{F}$. For our purposes, it would suffice, even if the condensed view of the adversary is dependent on the final output of the protocol, as long as it is independent of the functionality. This is because, in our context, the circuit being evaluated will be a relation circuit with the statement hard-coded and should always output 1 in order to convince the verifier.

Definition 6 ($\mathcal{F}$-universally simulatable MPC). *Let Π be an n-party MPC protocol that is capable of securely computing any function $f \in \mathcal{F}$ (where $\mathcal{F} : \mathcal{X}^n \to \mathcal{O}$) against any semi-honest adversary $\mathcal{A}$ who corrupts a set $\mathcal{I} \subset [n]$ of parties, such that $\mathcal{I} \in \mathcal{C}$, where $\mathcal{C}$ is the set of admissible corruption sets. We say that Π is $\mathcal{F}$-universally simulatable if there exists a 3-tuple of PPT functions* $(\mathsf{ExecuteMPC}, \mathsf{ExpandViews}, \mathsf{CondenseViews})$ *and a non-uniform PPT simulator $\mathcal{S}^{\text{F-MPC}} : \mathcal{F} \times \mathcal{C} \times \mathcal{O} \to V^*$, defined as follows*

- $(\{\mathsf{view}_i\}_{i\in[n]}, o) \leftarrow \mathsf{ExecuteMPC}(f, \{x_i\}_{i\in[n]})$*: This function takes inputs of the parties $\{x_i\}_{i\in[n]} \in \mathcal{X}^n$ and a function $f \in \mathcal{F}$ as input and returns the views $\{\mathsf{view}_i\}_{i\in[n]}$ of all parties and their output $o \in \mathcal{O}$ in protocol Π.*
- $\{\mathsf{con.view}_i\}_{i\in\mathcal{I}} \leftarrow \mathsf{CondenseViews}(f, \mathcal{I}, \{\mathsf{view}_i\}_{i\in\mathcal{I}}, o)$*: This function takes as input the set of corrupt parties $\mathcal{I} \in \mathcal{C}$, views of the corrupt parties $\{\mathsf{view}_i\}_{i\in\mathcal{I}}$ and the output of the protocol $o \in \mathcal{O}$ and returns their condensed views $\{\mathsf{con.view}_i\}_{i\in\mathcal{I}}$.*
- $\{\mathsf{view}_i\}_{i\in\mathcal{I}} \leftarrow \mathsf{ExpandViews}(f, \mathcal{I}, \{\mathsf{con.view}_i\}_{i\in\mathcal{I}}, o)$*: This function takes as input the functionality $f \in \mathcal{F}$, set of corrupt parties $\mathcal{I} \in \mathcal{C}$, condensed views $\{\mathsf{con.view}_i\}_{i\in\mathcal{I}}$ of the corrupt parties and the output of the protocol $o \in \mathcal{O}$ and returns their views $\{\mathsf{view}_i\}_{i\in\mathcal{I}}$.*
- $\{\mathsf{con.view}_i\}_{i\in\mathcal{I}} \leftarrow \mathcal{S}^{\text{F-MPC}}(f, \mathcal{I}, \{x_i\}_{i\in\mathcal{I}}, o)$*: The simulator takes as input the functionality $f \in \mathcal{F}$, set of corrupt parties $\mathcal{I} \in \mathcal{C}$, inputs of the corrupt parties $\{x_i\}_{i\in\mathcal{I}} \in \mathcal{X}^{|\mathcal{I}|}$ and the output of the protocol $o \in \mathcal{O}$ and returns simulated condensed views $\{\mathsf{con.view}_i\}_{i\in\mathcal{I}}$ of the corrupt parties.*

And these functions satisfy the following properties:

1. ***Condensing-Expanding Views is Deterministic:*** *For all $\{x_i\}_{i\in[n]} \in \mathcal{X}^n$ and $\forall f \in \mathcal{F}$, let $(\{\mathsf{view}_i\}_{i\in[n]}, o) \leftarrow \mathsf{ExecuteMPC}(f, \{x_i\}_{i\in[n]})$. For all $\mathcal{I} \in \mathcal{C}$ it holds that:*

$$\Pr\left[\mathsf{ExpandViews}(f, \mathcal{I}, \mathsf{CondenseViews}(f, \mathcal{I}, \{\mathsf{view}_i\}_{i\in\mathcal{I}}, o), o) = \{\mathsf{view}_i\}_{i\in\mathcal{I}}\right] = 1$$

2. ***Indistinguishability of Simulated Views from real execution:*** *For all* $\{x_i\}_{i\in[n]} \in \mathcal{X}^n$ *and* $\forall f \in \mathcal{F}$*, let* $(\{\mathsf{view}_i\}_{i\in[n]}, o) \leftarrow \mathsf{ExecuteMPC}(f, \{x_i\}_{i\in[n]})$*. For all* $\mathcal{I} \in \mathcal{C}$ *it holds that:*

$$\mathsf{CondenseViews}(f, \mathcal{I}, \{\mathsf{view}_i\}_{i\in\mathcal{I}}, o) \approx \mathcal{S}^{\text{F-MPC}}(f, \mathcal{I}, \{x_i\}_{i\in\mathcal{I}}, o)$$

3. ***Indistinguishability of Simulated Views for all functions:*** *For any* $\mathcal{I} \in \mathcal{C}$*, all inputs* $\{x_i\}_{i\in\mathcal{I}} \in \mathcal{X}^{|\mathcal{I}|}$ *of the corrupt parties, and all outputs* $o \in \mathcal{O}$*, there exists a function-independent distribution* $\mathcal{D}_{\{x_i\}_{i\in\mathcal{I}},o}$*, such that* $\forall f \in \mathcal{F}$*, if* $\exists\{x_i\}_{i\in[n]\setminus\mathcal{I}}$ *for which* $f(\{x_i\}_{i\in[n]\setminus\mathcal{I}}, \{x_i\}_{i\in\mathcal{I}}) = o$*, then it holds that:*

$$\mathcal{D}_{\{x_i\}_{i\in\mathcal{I}},o} \approx \mathcal{S}^{\text{F-MPC}}(f, \mathcal{I}, \{x_i\}_{i\in\mathcal{I}}, o)$$

We note that a central notion used in the "stacked-garbling literature" (for communication efficient disjunction for garbled circuit based zero-knowledge proofs) is a special case of $\mathcal{F}$-universally simulatable:

Remark 4 (Topology Decoupled Garbled Circuits and $\mathcal{F}$-universally simulatable MPC.). The notion of topology decoupled garbled circuits introduced by Kolesnikov [36] is a special case of $\mathcal{F}$-universally simulatable MPC: a topology decoupled garbled circuit (E, T) separates the cryptographic material (E, e.g. garbling tables) and topology (T, i.e. wiring) of a garbled circuit and (informally stated) requires that generating E for different topologies introduces indistinguishable distributions. Letting X be the garbled input labels[17] held by the evaluator, in $\mathcal{F}$-universally simulatable terminology (E, X) would constitute the "condensed view", while (E, X, T)[18] would constitute the "expanded views" , indistinguishablilty of simulated views for functions with the same number of gates and inputs follows easily from the "topology decoupling" of the garbled circuits and the uniform distribution of the input labels.

In the full version [23], we prove the following theorem, which states that when instantiated with an $\mathcal{F}$-universally simulatable MPC protocol, Theorem 2 yields a stackable Σ-protocol for languages with relation circuits in $\mathcal{F}$.

Theorem 3 ($\mathcal{F}$-universally simulatable implies stackable). *The IKOS compiler (see Theorem 2) yields an stackable Σ-protocol for languages with relation circuit in $\mathcal{F}$ when instantiated with an $\mathcal{F}$-universally simulatable MPC protocol (see Definition 6) with privacy and robustness against a subset of the parties.*

In the full version, we use Theorem 3 to show that two popular IKOS-based Σ-protocols are stackable, namely KKW [35] and Ligero [2].

Self-Stacking Compiler

Statement: $x = x_1, \ldots, x_n$
Witness: $w = (\alpha, w_\alpha)$

- **First Round:** Prover computes $A'(x, w; r^p) \to a$ as follows:
  - Parse $r^p = (r^p_\alpha \| r)$.
  - Compute $a_\alpha \leftarrow A(x_\alpha, w_\alpha; r^p_\alpha)$.
  - Set $\mathbf{v} = (v_1, \ldots, v_\ell)$, where $v_\alpha = a_\alpha$ and $\forall i \in [\ell] \setminus \alpha$, $v_i = 0$.
  - Compute $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B = \{\alpha\})$.
  - Compute $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}; r)$.
  - Send $a = (\mathsf{ck}, \mathsf{com})$ to the verifier.
- **Second Round:** Verifier samples $c \leftarrow \{0,1\}^\lambda$ and sends it to the prover.
- **Third Round:** Prover computes $Z'(x, w, c; r^p) \to z$ as follows:
  - Parse $r^p = (r^p_\alpha \| r)$.
  - Compute $z^* \leftarrow Z(x_\alpha, w_\alpha, c; r^p_\alpha)$.
  - For $i \in [\ell]/\alpha$, compute $a_i \leftarrow \mathcal{S}^{\text{EHVZK}}(x_i, c, z^*)$.
  - Set $\mathbf{v}' = (a_1, \ldots, a_\ell)$
  - Compute $r' \leftarrow \mathsf{Equiv}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}, \mathbf{v}', \mathsf{aux})$ (where aux can be regenerated with r).
  - Send $z = (\mathsf{ck}, z^*, r')$ to the verifier.
- **Verification:** Verifier computes $\phi'(x, a, c, z) \to b$ as follows:
  - Parse $a = (\mathsf{ck}, \mathsf{com})$ and $z = (\mathsf{ck}', z^*, r')$
  - Set $a_i \leftarrow \mathcal{S}^{\text{EHVZK}}(x_i, c, z^*)$
  - Set $\mathbf{v}' = (a_1, \ldots, a_\ell)$
  - Compute and return:

$$b = (\mathsf{ck} \stackrel{?}{=} \mathsf{ck}') \wedge \left(\mathsf{com} \stackrel{?}{=} \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}'; r')\right) \wedge \left(\bigwedge_{i \in [\ell]} \phi(x_i, a_i, c, z^*)\right)$$

Fig. 3: A compiler for stacking multiple instances of a Σ-protocol.

6 Self-Stacking: Disjunctions With The Same Protocol

We now present a self-stacking compiler for Σ-protocols, presented in Figure 3. By self-stacking, we mean a compiler that takes a *stackable* Σ-protocol Π for a language $\mathcal{L}$ and produces a Σ-protocol for language with disjunctive statements of the form $(x_1 \in \mathcal{L}) \vee (x_2 \in \mathcal{L}) \vee \ldots \vee (x_\ell \in \mathcal{L})$ with communication complexity proportional to the size of a single run of the underlying Σ-protocol (along with an additive factor that is linear in ℓ and λ). The key ingredient in our compiler is the partially-binding vector commitments (See Definition 2), which will allow the prover to efficiently compute verifying transcripts for the inactive clauses.

The compiler generates an accepting transcript (a_α, c, z^*) to the active clause $\alpha \in [\ell]$ using the witness, and then simulates accepting transcripts for each non-

[17] Obtained using an oblivious transfer.
[18] Where T can be computed from f.

active clause, using the extended simulator. Recall that this extended simulator takes in a third round message z and a challenge c and produces a first round message a such that $\phi(x, a, c, z) = 1$. Thus, the prover can *re-use* the third round message z^*, for each simulated transcript, thereby reducing communication to the size of a single third round message. For a more detailed overview, we refer the reader to Section 2.

We now present a formal description of the self-stacking compiler, which we prove in the full version of the paper [23].

Theorem 4 (Self-Stacking). *Let $\Pi = (A, Z, \phi)$ be a stackable (See Definition 5) Σ-protocol for the NP relation $\mathcal{R} : \mathcal{X} \times \mathcal{W} \to \{0,1\}$ and let $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{EquivCom}, \mathsf{Equiv}, \mathsf{BindCom})$ be a 1-out-of-ℓ binding vector commitment scheme (See Definition 2). For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, the compiled protocol $\Pi' = (A', Z', \phi')$ described in Figure 3 is a stackable Σ-protocol for the relation $\mathcal{R}' : \mathcal{X}^\ell \times ([\ell] \times \mathcal{W}) \to \{0,1\}$, where $\mathcal{R}'((x_1, \ldots, x_\ell), (\alpha, w)) := \mathcal{R}(x_\alpha, w)$.*

Communication Complexity. Let $\mathtt{CC}(\Pi)$ be the communication complexity of Π. Then, the communication complexity of the Π' obtained from Theorem 4 is $(\mathtt{CC}(\Pi) + |\mathsf{ck}| + |\mathsf{com}| + |r'|)$, where the sizes of ck, com and r' depends on the choice of partially-binding vector commitment scheme and are independent of $\mathtt{CC}(\Pi)$. With our instantiation of partially binding vector commitments, the size of $|\mathsf{ck}|, |r'|$ will depend linearly on ℓ. However since our resulting protocol Π' is also stackable, the communication complexity can be reduced further to $\mathtt{CC}(\Pi) + 2\log(\ell)(|\mathsf{ck}| + |\mathsf{com}| + |r'|)$ by recursive application of the compiler as follows: let $\Pi_1 = \Pi$ and for $n > 1$ let Π_{2n} be the outcome of applying the compiler from Theorem 4 with $\ell = 2$ to Π_n. Note that Π_ℓ only applies the stacking compiler $\lceil \log(\ell) \rceil$ times and that $\mathtt{CC}(\Pi_{2n}) = \mathtt{CC}(\Pi_n) + |\mathsf{ck}| + |\mathsf{com}| + |r'|$. Therefore $\mathtt{CC}(\Pi_\ell) = \mathtt{CC}(\Pi) + 2\log(\ell)(|\mathsf{ck}| + |\mathsf{com}| + |r'|)$.

Computational Complexity. In general, the computation complexity of this protocol is ℓ times that of Π. However, in many protocols, the simulator is much faster than computing an honest transcript. We note that for such protocols, our compiler is expected to also get savings in the computation complexity.

6.1 Extending to Multiple Languages

Many known constructions of Σ-protocols work for more than one language. For instance, most MPC-in-the-head style Σ-protocols (e.g. KKW [35] , Ligero [2]) can support all languages with a polynomial sized relation circuit, as long as the underlying MPC protocol works for any polynomial sized function. However, because Σ-protocols are defined w.r.t. a particular NP language/relation, instantiating [35] for two different NP languages $\mathcal{L}_1$ and $\mathcal{L}_2$ will (by definition) result in two distinct Σ-protocols. Therefore, applied naïvely, our compiler could only be used to stack Σ-protocols from [35] for the exact same relation circuit.

We explore two alternatives for overcoming this limitation. First, we note that in many situations it is trivial to generalize our technique to cover "Σ-protocols based on a particular technique," *e.g.* protocols based on [35]. This can

be done using a "meta-language" like circuit satisfiability that generalizes across multiple NP languages without introducing a high NP-reduction cost. In the full-version of this paper [23], we explore a conceptually different approach, we call *cross-stacking*. Our cross-stacking compiler applies the self-stacking techniques to different Σ-protocols (both based on a single technique and otherwise) without modifying the Σ-protocols. The major barrier is that the third round message distributions of the Σ-protocols are different, so third round messages may not be recyclable. To overcome this, we define a distribution which captures the "union" of the third round message distributions of each Σ-protocol, and map messages into and out of this distribution. The communication complexity of the resulting protocol is determined by the size of this distribution. We give concrete case studies of cross-stacking various Σ-protocols.

7 Acknowledgments

We would like to thank the anonymous reviewers of CRYPTO 2021 for their helpful comments on our initial construction of the partially-binding commitments. Additionally, we would like to thank Nicholas Spooner for his helpful comments on the definition of our commitments.

The first and second authors are supported in part by NSF under awards CNS-1653110, and CNS-1801479, and the Office of Naval Research under contract N00014-19-1-2292. The first author is also supported in part by NSF CNS grant 1814919, NSF CAREER award 1942789 and the Johns Hopkins University Catalyst award. The second author is also funded by DARPA under Contract No. HR001120C0084, as well as a Security and Privacy research award from Google. The third author is funded by Concordium Blockhain Research Center, Aarhus University, Denmark. The forth author is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and is supported by DARPA under Agreements No. HR00112020021 and Agreements No. HR001120C0084. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

1. Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 415–432. Springer, Heidelberg, December 2002.
2. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.
3. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas

Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.
4. Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. 2020. `https://eprint.iacr.org/2020/753`.
5. Carsten Baum, Alex J. Malozemoff, Marc Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for arithmetic circuits with nested disjunctions. Cryptology ePrint Archive, Report 2020/1410, 2020. `https://eprint.iacr.org/2020/1410`.
6. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.
7. 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*, pages 459–474. IEEE Computer Society Press, May 2014.
8. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.
9. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.
10. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.
11. Manuel Blum. How to prove a theorem so no one else can claim it. pages 1444–1451, 1987.
12. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on DDH. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *ESORICS 2015, Part I*, volume 9326 of *LNCS*, pages 243–265. Springer, Heidelberg, September 2015.
13. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.
14. David Chaum and Torben P. Pedersen. Transferred cash grows in size. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 390–407. Springer, Heidelberg, May 1993.
15. Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 63–92. Springer, Heidelberg, May 2016.
16. Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 424–441. Springer, Heidelberg, August 1998.
17. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

18. Prastudy Fauzi, Helger Lipmaa, Zaira Pindado, and Janno Siim. Somewhere statistically binding commitment schemes with applications. Cryptology ePrint Archive, Report 2020/652, 2020. https://eprint.iacr.org/2020/652.
19. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.
20. Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, Heidelberg, April 2015.
21. Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 177–194. Springer, Heidelberg, May 2003.
22. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.
23. Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose σ-protocols for disjunctions. Cryptology ePrint Archive, Report 2021/422, 2021. https://eprint.iacr.org/2021/422.
24. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.
25. Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.
26. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.
27. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.
28. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.
29. Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 216–231. Springer, Heidelberg, August 1990.
30. David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792. Springer, Heidelberg, August 2020.
31. David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, Heidelberg, May 2020.
32. David Heath, Vladimir Kolesnikov, and Stanislav Peceny. MOTIF: (almost) free branching in GMW - via vector-scalar multiplication. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 3–30. Springer, Heidelberg, December 2020.

33. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.
34. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013.
35. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.
36. Vladimir Kolesnikov. Free `IF`: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 34–58. Springer, Heidelberg, December 2018.
37. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Heidelberg, December 2001.
38. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.
39. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.
40. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
41. swisspost evoting. E-voting system 2019. `https://gitlab.com/swisspost-evoting/e-voting-system-2019`, 2019.
42. Greg Zaverucha. The picnic signature algorithm. Technical report, 2020. `https://github.com/microsoft/Picnic/raw/master/spec/spec-v3.0.pdf`.