Papers

Supported By

The Broadcom Foundation

and

Tel Aviv University Authentication Initiative

February 2015

Papers

Supported By

The Broadcom Foundation

and

Tel Aviv University Authentication Initiative

February 2015

TEL AVIV UNIVERSITY אוניברסיטת תל-אביב

February 2015

Dear Dr. Samueli and the Broadcom Foundation,

It is our great pleasure to present you with the recent academic work produced by the Broadcom Foundation Scholars at Tel Aviv University.

Real and tangible progress has been accomplished by our researchers engaged in developing a new generation of authentication methods that are critical to ensure proper authorization and access to systems and services. As you will see through the papers presented, their basic research is expected to attain new heights in the implementation and protection of security and information authentication in the increasingly changing and complex cyber-based world.

We are grateful that you have chosen to invest in Tel Aviv University; your support has been a major reinforcement of our research efforts. We are looking forward to continuing our cooperation in additional research fields that will further advance electronic identity protection in the coming years.

Sincerely,

Prof. Joseph Klafter, President
Tel Aviv University

Prof. Yaron Oz, Dean
Raymond and Beverly Sackler
Faculty of Exact Sciences

Contents

Dr. Daniel Deutch and Prof. Tova Milo

The Blavatnik School of Computer Science
Raymond and Beverly Sackler Faculty of Exact Sciences

Paper

- **Approximated Provenance for Complex Applications**

Approximated Provenance for Complex Applications

Eleanor Ainy
Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University
eleanora@cs.tau.ac.il

Susan B. Davidson
University of Pennsylvania
susan@cis.upenn.edu

Daniel Deutch
Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University
danielde@post.tau.ac.il

Tova Milo
Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University
milo@post.tau.ac.il

Abstract

Many applications now involve the collection of large amounts of data from multiple users, and then aggregating and manipulating it in intricate ways. The complexity of such applications, combined with the size of the collected data, makes it difficult to understand how information was derived, and consequently difficult to asses its credibility, to optimize and debug its derivation, etc. Provenance has been helpful in achieving such goals in different contexts, and we illustrate its potential for novel complex applications such as those performing *crowd-sourcing*. Maintaining (and presenting) the full and exact provenance information may be infeasible for such applications, due to the size of the provenance and its complex structure. We propose some initial directions towards addressing this challenge, through the notion of approximated provenance.

1. Introduction

Applications that involve the collection, aggregation and manipulation of large amounts of data, interacting with multiple users in intricate ways, have become increasingly popular. A notable example are *crowd-sourcing applications* like Wikipedia, social tagging systems for images, traffic information aggregators like Waze, hotel and movie ratings like TripAdvisor and IMDb. and platforms for performing complex tasks like protein folding via the online game FoldIt.

Several questions arise relating to *how data was derived*: as a consumer, what is the basis for trusting the information? If the information seems wrong, how can the provider debug why it is wrong? And if some data is found to be wrong, e.g. created by a spammer, how can it be "undone'" if it affected some aggregated result? At its core, the answer to these questions is based on the *provenance* of the collected data and resulting information, that is, *who* provided the information in *what* context and *how* the information was manipulated. We note in this context that there is an extensive study of dedicated tools and techniques for such questions (e.g. spammer identification, analysis of trust), see e.g. [10] for a survey. As we demonstrate below, this study is complementary to our approach, both in the sense that such techniques may (depending on the intended use of provenance) be incorporated to guide our (more general framework of) provenance maintenance, as well as in the sense that they may be employed using the maintained portion of provenance as input.

We start by providing two concrete examples of provenance for complex applications, focusing on the crowd-sourcing domain.

TripAdvisor aggregates crowd-sourced travel-related reviews (of restaurants, hotels, etc.), and presents average ratings of reviewed destinations. The individual reviews thus stand as the *provenance* of the average rating. TripAdvisor also provides users with several different views of this provenance: viewing ratings and individual reviews, or aggregating them separately for different trip types (e.g. family, couple, or solo). Through such views, we may understand better how the aggregated rating was computed, who provided the information, when they traveled, etc.

Wikipedia keeps extensive information about provenance of edited pages. This includes the ID of the user who generated the page and information regarding changes to pages, recording when the change was made, who made the change, a (natural language) summary of the change, etc. Wikipedia also provides several views on this provenance information, e.g. view by page (through clicking on the "View History" tab) or by editor (which gives information about them and all their contributions to Wikipedia pages).

As exemplified above, current systems use a simple solution for provenance management, namely recording provenance and presenting some given views of it upon request. While provenance is indeed useful for *presentation and explanation*, we will discuss below how it is also useful for *maintenance and optimization* of the application. We claim that simply recording all provenance is an inadequate approach for these purposes in our setting, due to the *complexity* of processes and the typically *large size* of provenance information. For instance, in a crowd-sourcing setting, the process is complex since data is collected through a variety of means, ranging from simple questions (e.g. "In which picture is this person the oldest?"or "What is the capital of France?"), to Datalog-style reasoning [7], to dynamic processes that evolve as answers are received from the crowd (e.g. mining association rules from the crowd [4]). The complexity of the process, together with the number of user inputs involved in the derivation of even a single value, lead to an

Tapp '14, June 12–13, 2014, Cologne, Germany.

http://dx.doi.org/10.1145/

extremely large provenance size, which leads to further difficulties in viewing and understanding the information.

We next identify challenges in leveraging provenance for presentation and maintenance, accounting for its size and complexity.

Summarization. A consequence of the large size of provenance is the need for ways of summarizing or abstracting provenance information. For example, in heavily edited Wikipedia pages we might wish to view a higher level summary of the changes, such as "entries x_1, x_2, x_3 are formatting changes, entries y_1, y_2, y_3, y_4 add content, and entries z_1, z_2 represent divergent viewpoints", or "entries u_1, u_2, u_3 represent edits by robots, and entries v_1, v_2 represent edits by Wikipedia administrators." As another example, in a movie-rating application we may wish to summarize the provenance of the average rating for "Blue Jasmine" by saying that Europeans aged 20-30 gave high ratings (8-10) whereas American teenagers aged 15-18 gave low ratings (3-5).

Support of Multidimensional Views. The environment in complex applications such as those performing crowdsourcing consists of multiple components, each of which provides a perspective through which provenance can be viewed or mined. For example, Wikipedia allows you to see the edit history of a user, as well as of an entry. In TripAdvisor, if there is an "outlier" review (i.e. very negative), it would be nice to be able to see what other reviews the person has provided to be able to calibrate their response. In applications where questions are posed to the crowd (e.g. in FoldIt), a *question* perspective could be used to determine which questions were bad or unclear, i.e. those where many users did not answer the question or where the spread of answers was large; or we may wish to determine why a particular question was asked of a particular user. Such views may be incorporated as part of provenance visualization tools.

Mining. Conventional management of provenance typically addresses questions such as "What data or processes contributed to this result (and in what ways)". For instance, in database provenance, provenance may be used to identify whether a piece of data is a component of an input tuple; in workflow provenance it may be used to identify whether the data is a parameter or input data. We aspire to use provenance in additional ways, in particular to mine it and potentially combine it with contextual information. Returning to the rating application, we may have (contextual) information about users, e.g. age, nationality, and sex; we may also have information about hotels, e.g. chain name, location, etc. By mining the provenance, we may be able to identify demographics of the users (e.g. Europeans aged 20-30, or Americans aged 15-18) that correlate with certain answers (e.g. ranges of scores) for classes of hotels, providing a more in-depth analysis of the application behavior and results.

Compact Representation for Maintenance and Cleaning. The ability to mine provenance is useful not only for presentation, but also for data maintenance and cleaning. For example, mining could be used to identify spammers, whose "bad" answers should then be removed from an aggregate calculation. Provenance management techniques should therefore enable hypothetical reasoning and update propagation, with respect to the effect of the removal or deletion of certain users, questions and/or answers on the computation. A main goal is to obtain a compact provenance representation for these tasks. Then, depending on the concrete task / application, known techniques for trust assessment, spammers detection etc. may also be employed, taking (some form of) the provenance as input. This is of particular importance since the mining of provenance may lag behind the aggregate calculation; for example, detecting a spammer may only be possible when they have answered enough questions, or when enough answers have been obtained from other users. Note that the aggregate calculation may in turn have already been used in a downstream calculation, or have been used to direct the process itself.

Perspective and Scope Provenance models have been extensively studied in multiple lines of research such as provenance for database transformations (see [5] for an overview), for workflows (see [6] for an overview), for the web [1], for data mining applications (initial ideas can be found in [8]), and many others. The basic provenance model that we will rely on in Section 2 is borrowed from these foundations; the challenges listed above, however, were not addressed but are particularly crucial in pursuit of a provenance framework for complex applications.

2. Approximated Provenance

We propose some initial directions towards a solution, based on a novel generic notion of *approximated provenance*, whose computation is guided by the intended use. Our starting point is the (exact) provenance model presented in [2], which "marries" database style and workflow-style provenance. We first explain this model through an example. We then discuss the need for approximations, identify considerations in defining good approximations, and provide initial computational results.

2.1 Workflow and Provenance Model

We start by presenting the notion of workflows that query an underlying database via an example from the crowd-sourcing domain.

EXAMPLE 2.1. *Consider a movie reviews aggregator platform, whose logic is captured by the workflow in Figure 1. Inputs for the platform are reviews (scores) from users, whose identifiers and names are stored in the* Users *table. Users have different roles (e.g. movie critics, directors, audience, etc.); information about two such roles,* Critics *and* Audience*, is shown in the corresponding relations. Reviews are fed through different reviewing modules, which "crawl" different reviewing platforms such as IMDB, newspaper web-sites etc. Each such module updates statistics in the* Stats *table, e.g. how many reviews the user has submitted (*NumRate*), what their average score is (computed as* SumRate *divided by* NumRate*), etc. A reviewing module also consults* Stats *to output a* "sanitized review" *by implementing some logic. The sanitized reviews are then fed to an aggregator, which computes an aggregate movies scores. There are many plausible logics for the reviewing modules; we exemplify one in which each module "sanitizes" the reviews by joining the users, reviews and audience/critic relation (depending on the module), keeping only reviews of users listed under the corresponding role (audience/critic) and who are "active", i.e. who have submitted more than 2 reviews. The aggregator combines the reviews obtained from all modules to compute overall movie ratings (sum, num, avg).*

An *execution* of such a workflow is a repeated application of the modules, updating the database accordingly.

Provenance By looking only at the database obtained after workflow executions, we obtain only partial information about what happened. For instance, in our example it is hard to understand details behind the overall rating of a given movie, i.e. how was it computed, what kinds of users liked it more than others, etc. Similarly, if we conclude (e.g. using dedicated tools) that a particular user was a "spammer", it may be difficult to estimate the effect of the spammer on the overall movie rating, hence the rating may need to be re-calculated.

However, using the provenance semirings approach of [2, 3, 9] we may track this information. We start by assigning *"provenance tokens"* to input tuples in the workflow database. These can be thought of as abstract variables identifying the tuples. Provenance

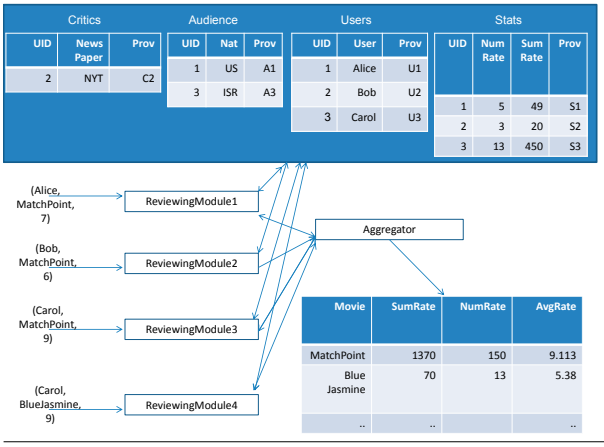


Figure 1. Crowdsourcing Application Workflow

is then propagated through manipulations of the data, creating provenance-aware output tuples. There is an intuitive correspondence between algebraic operations in the *semi-module structure* [3] and data manipulation. In particular, $+$ corresponds to the *alternative use* of data (as in union and projection), $\cdot$ to the *joint use* of data (as in join), $\otimes$ is used to "pair" annotations and values for aggregation queries, and $\oplus$ is used to capture the aggregation function. Additional correspondences include the use of 1 to annotate data that is always available (we do not track its provenance), and 0 for data that is absent. We further introduce special "condition tokens", such as $[(d_1 \cdot d_2) \otimes m > 2]$. Intuitively such tokens are kept abstract and can be used in conjunction with other tokens. Given concrete values for d_1, d_2 and m one may then test the truth value of the equality.

EXAMPLE 2.2. *Returning to Figure 1, note now the* Prov *column storing provenance of the different tuples. Tuples in the* Users *relation are associated with a token U_i, those in the* Audience *relation are associated with A_i etc. The provenance-aware value stored as* SumRate *for the "MatchPoint" tuple would be:*

$$(U_1 \cdot A_1) \cdot [S_1 \cdot U_1 \otimes 5 > 2] \otimes 7 \oplus (U_2 \cdot C_2) \cdot [S_2 \cdot U_2 \otimes 3 > 2] \otimes 6 \oplus (U_3 \cdot A_3) \cdot [S_3 \cdot U_3 \otimes 13 > 2] \otimes 9 \oplus \ldots$$

Intuitively, each numeric rating is associated with the provenance of the tuple obtained as the output of the reviewing module, namely the "U_i" annotation identifying the user, multiplied by either an A or C annotation, representing the join query applied by the module to check whether the user's role is "critic" or "audience". Each such sub-expression is multiplied by an inequality term serving as a conditional "guard", indicating that the number of reviews recorded for the user is above the threshold (2). Applying aggregation then results in coupling values (numeric reviews) with annotations to form the expression above.

Using Provenance Provenance may serve as the basis for addressing issues listed in the introduction. In particular, maintaining and visualizing provenance expressions such as those shown above enable users/administrators of the workflow to *understand* how specific movie ratings were computed. Provenance expressions may also be used for efficient *data maintenance and cleaning*: e.g. realizing that user U_2 is a spammer, we may "map" its provenance annotation to false (0). This will have the effect of mapping the entire expression $(U_2 \cdot C_2) \cdot [S_2 \cdot U_2 \otimes 3 > 2] \otimes 6$ to 0, thus disabling its effect on the overall aggregate ratings.

However, storing, or showing to users, the exact and full provenance expression may be infeasible. For example, there may be many reviewers for a movie, thus the provenance expression may be very large. We propose to address this challenge using *approximated provenance*, in which there is some loss of information to allow compact representation.

2.2 Approximations via Mappings

There are several plausible directions for addressing the size blowup of provenance. For instance in [11] the authors propose a factorized (exact) representation of query results. Our approach is different in that we are willing to settle for approximation in pursuit of smaller-size provenance. Note that (boolean) provenance approximation for use in probabilistic databases was considered in [12]. Here we will consider more general provenance expressions, specifically ones obtained through the use of aggregation in the tracked computation, and in different use cases, guiding different notions of approximation.

In particular, we propose a general framework of provenance approximation based on *mappings*. Let Ann be a domain of annotations and let Ann' be a domain of annotation "summaries". Typically, we expect that $|\ Ann'\ | << |\ Ann\ |$. To model approximation, we define a mapping $h : Ann \mapsto Ann'$ which maps each annotation to a corresponding "summary". This extends naturally to expressions over Ann, via the definition of a homomorphism. Essentially, to apply h to a provenance expression p (we denote the result by $h(p)$), each occurrence of $a \in Ann$ in p is replaced by $h(a)$.

EXAMPLE 2.3. *Recall Example 2.2, and let $Ann = \{U_1, \ldots\} \cup \{A_1, ..\} \cup \{C_1, \ldots\} \cup \{S_1, \ldots\}$). Consider a homomorphism that maps all U_i and S_i annotations to* 1*, all A_i annotations to A and all C_i annotations to C (so $Ann' = \{C, A\}$). Note that the mapping is applied only to annotations and not to numeric values. By applying the homomorphism to the provenance-aware values stored as movie ratings, and applying the congruence rules of the semimodule, we obtain for each movie expressions of the sort:*

$$C \otimes V_C \oplus A \otimes V_A$$

where V_C (V_A) is some concrete value which is the sum of scores of critics (audience) for which the "threshold" condition holds. The condition itself was mapped to 1 *for such users (thus disappeared), and to* 0 *for others (causing the summand corresponding to their review to disappear).*

To understand how the above simplification was obtained note that, for instance, for the $SumRate$ value corresponding to MatchPoint we will get

$$(1 \cdot A) \cdot [1 \otimes 5 > 2] \otimes 7 \oplus (1 \cdot C) \cdot [1 \otimes 3 > 2] \otimes 6 \oplus (1 \cdot A) \cdot [1 \otimes 13 > 2] \otimes 9 \oplus \ldots$$

which simplifies to

$A \otimes 7 \oplus C \otimes 6 \oplus A \otimes 9 \equiv A \otimes 16 \oplus C \otimes 6 \oplus \ldots$.

Note that we are interested in the average *score. To this end, a similar transformation will be applied to the values in the $NumRate$ column, resulting in an expression of the same flavor, where V_C would be the* number *of critics reviews on the movie and V_A would be the number of audience reviews (details are omitted). These quantities may then be used to compute averages by critics and by audience.*

In the example, we used one possible mapping h that clusters reviews based on role. In general there may be many possible mappings and the challenge is, given a provenance expression p, to (a) define what a "good" mapping h is (and consequently what a "good" approximation $h(p)$ is), and (b) find such "good" h.

Quantifying Approximation Quality Several, possibly competing, considerations need to be combined.

Provenance size. Since the goal of approximation is to reduce the provenance size, it is natural to use the size of the obtained expression as a measure of its quality.

Semantic Constraints. The obtained provenance expression may be of little use if it is constructed by identifying multiple unrelated annotations; it is thus natural to impose constraints on which annotations may be grouped together. One simple example of such a constraint is to allow two annotations $a, a' \in Ann$ to be mapped to the same annotation in Ann' (with the exception of 0 and 1) only if they annotate tuples in the *same input table*, intuitively meaning that they belong to the same domain. E.g. in the above example it allows mapping annotations R_i, R_j to the same token, but disallows mapping R_i, S_j to the same token. Other more complicated constraints may be based, e.g., on clustering w.r.t similar same values in possibly multiple attributes.

Distance. Depending on the *intended use* of the provenance expression, we may *quantify the distance* between the original and approximated expression. As an example, consider a distance function designed with the intention to use provenance for *recomputation in the presence of spammers* (whose identification is performed using certain techniques). Recall that provenance expressions enable this using *truth valuations* for the tokens. Intuitively, specifying that u_1 is a spammer corresponds to mapping it to $false$ (and that u_1 is reliable to mapping it to $true$), and recomputing the derived value w.r.t this valuation. Such valuation can again be extended in the standard way to a valuation $V : N[Ann] \mapsto \{true, false\}$. Now let $\mathcal{V}_{Ann}$ be the set of all such valuations for a set Ann of annotations. A central issue is how we "transform" a valuation in $\mathcal{V}_{Ann}$ to one in $\mathcal{V}_{Ann'}$. We propose that this will be given by a "combiner" function ϕ. We can then define the distance between a provenance expression p and its approximated version $h(p)$ as an average over all truth valuations, of some property of p, $h(p)$, and the valuation. This property is based on yet another function, whose choice depends on the intended provenance use. For *maintenance*, we may e.g. use a function that returns the absolute difference between the two expressions values under the valuation or, alternatively, a function whose value is 0 if the two expressions agree under the valuations, and 1 otherwise (so the overall distance is the fraction of disagreeing valuations).

Putting it all together. Several computational problems are then of interest. Given a provenance expression p (and a fixed ϕ) we may wish to find a mapping h, s.t. the approximated expression p' defined by p, h, ϕ (a) satisfies the semantic constraints and (b) either (1) minimizes the distance from p out of all such p' of bounded size, or (2) minimizes the expression size, out of all expressions (obtained through some h, ϕ) within a bounded distance from p. Variants include treating the constraints as "soft" ones, assigning weights to the different factors etc.

Computing Approximations A natural question is how difficult it is to compute such approximations. Our initial results show that even computing the goodness of an approximation defined by a *given* h is already $\sharp P$*-hard*. This holds even if the combiner $\phi = +$ and p includes no tensor elements (there is no aggregation in the workflow), and even if there are no semantic constraints. On the other hand, we can show an *absolute approximation* algorithm for computing distance between two such provenance expressions. This allows greedy heuristics that constructs the homomorphism h gradually, by choosing at each step to identify a pair of variables that leads to minimal increase in distance.

Incorporating Additional Knowledge The search space defined by possible mappings may be quite large in practice. One may add additional constraints that guide the search using additional knowledge. For instance, an estimation of how much we trust different users (derived via dedicated techniques) can be used as an additional constraints over mapping / provenance approximations, disqualifying ones that discard "too many" contributions of trusted users. As another example, additional semantic knowledge such as ontologies can be used to limit the scope of considered mappings, intuitively restricting the attention to mappings that group together "semantically related" annotations.

3. Conclusion

In this short vision paper, we discussed the challenges associated with provenance for complex applications, and proposed the use of a provenance model which "marries" database and workflow-style provenance, combined with a novel notion of approximated provenance as the basis for addressing these challenges. We note that the way in which the approximated provenance expression is computed essentially involves clustering annotations – the clusters being defined by the choice of homomorphism – but, unlike standard clustering techniques, here the choice of clusters is guided by the particular provenance expression which is in turn effected by the tracked computation (e.g. which aggregation function was used). The approximation-via-homomorphism approach also has the advantage of compatibility with the underlying semiring-based provenance model, thereby allowing for robust foundations. Adaptation to specific contexts can be achieved through the use of application-depended distance functions.

However, this is just a first step and much remains to be done. In particular: (1) How should we generate the mapping function h for a particular approximation usage? In the examples we have given, this could be done by mining the combined contextual and provenance information to find patterns in the ratings, e.g. demographics of the users that correlate with certain answers. (2) How does the mining process interact with the distance function? (3) How can approximated provenance be used for repair? Hypothetical reasoning is employed here with only partial information. Efficient implementation of these ideas is also an open challenge.

Acknowledgments This research was partially supported by the European Research Council under the European Community 7th Framework Programme (FP7/2007-2013) / ERC grant MoDaS, grant agreement 291071, by the Israeli Ministry of Science, the Israeli Science Foundation (ISF), by the US-Israel Binational Science Foundation (BSF), the Broadcom Foundation and Tel Aviv University Authentication Initiative.

References

[1] Provenance working group. `http://www.w3.org/2011/prov/`.

[2] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting Lipstick on Pig: Enabling Database-style Workflow Provenance. *PVLDB*, 2012.

[3] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.

[4] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD Conference*, 2013.

[5] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[6] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD Conference*, pages 1345–1350, 2008.

[7] D. Deutch, O. Greenshpan, B. Kostenko, and T. Milo. Declarative platform for data sourcing games. In *WWW*, pages 779–788, 2012.

[8] B. Glavic, J. Siddique, P. Andritsos, and R. J. Miller. Provenance for Data Mining. In *Theory and Practice of Provenance (TAPP)*, 2013.

[9] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[10] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[11] D. Olteanu and J. Zavodny. Factorised representations of query results: size bounds and readability. In *ICDT*, pages 285–298, 2012.

[12] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.

Prof. Slava Krylov

School of Mechanical Engineering
The Iby and Aladar Fleischman Faculty of Engineering

Papers

- **Experimental Investigation of the Snap-Through Buckling of Electrostatically Actuated Initially Curved Pre-Stressed Micro Beams**
- **Dynamic Trapping in BI-Stable Electrostatically Actuated Curved Micro Beams**
- **Dynamic Trapping Experiment in an Electrostatically Actuated Initially Curved Beam**

Abstracts

- **Experimental Investigation of the Snap-Through Buckling of Electrostatically Actuated Initially Curved Pre-Stressed Micro Beams**
- **Bi-Stable Micro Beams Under Electrostatic Load – An Overview**

Sensors and Actuators A 220 (2014) 323–332

Contents lists available at ScienceDirect

Sensors and Actuators A: Physical

journal homepage: www.elsevier.com/locate/sna

Experimental investigation of the snap-through buckling of electrostatically actuated initially curved pre-stressed micro beams

Lior Medina[a,*], Rivka Gilat[b], Bojan Ilic[c], Slava Krylov[a]

[a] Faculty of Engineering, School of Mechanical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel
[b] Department of Civil Engineering, Faculty of Engineering, Ariel University, Ariel 44837, Israel
[c] School of Engineering and Applied Physics and Cornell Nanoscale Facility, Cornell University, Ithaca, NY, USA

ARTICLE INFO

Article history:
Received 30 April 2014
Received in revised form 14 October 2014
Accepted 14 October 2014
Available online 27 October 2014

Keywords:
Bistable micro beam
Electrostatic actuation
Snap-through
Pull-in
MEMS/NEMS
Experiments

ABSTRACT

The experimental study of the stability properties of initially curved micro beams subjected to an axial pre-stressing load and transversal deflection-dependent distributed electrostatic force is presented. The devices were fabricated from single crystal silicon on insulator (SOI) wafer using deep reactive ion etching (DRIE). The in-plane quasi-static beam response was video recorded and analyzed by means of image processing. The beam behavior was theoretically predicted using a reduced order model with the axial pre-stressing force assessed on the basis of the deviation of the beam actual initial curvature from the nominal designed one. The experimental results are consistent with the theoretical symmetric and asymmetric snap-through buckling criteria.

1. Introduction

Curved beams (arches) loaded by concentrated or distributed transverse forces may exhibit the existence of two different stable equilibria under the same loading. The transition between two stable states is commonly referred to as a snap-through buckling. The behavior of beams liable to the snap-through buckling due to prescribed deflection-independent "mechanical" loads is a well established topic in structural mechanics [1–9].

In the case of the electrostatic actuation, the snap-through behavior is affected by the nonlinearity of the electrostatic force parameterized by the actuating voltage and dependent on the initial distance between the beam and the electrode. Microfabricated double-clamped beams attached to fully constrained (unmovable) anchors, are often pre-loaded by an axial force. This force originates in a residual stress resulting from the fabrication process [10], appearing due to a temperature variation [5,11,12] or applied intentionally in order to control the stiffness. In particular, the compressive axial load reduces the stiffness of an initially straight beam only in a pre-buckling configuration. In the post-bucking state the compression actually increases the stiffness [13,14], thus changing the natural frequency and the pull in voltage of the structure [15–17].

The theoretical stability analysis of curved beams actuated by a distributed deflection-dependent electrostatic force was presented in Refs. [18–22] for static and dynamic loadings. Symmetry breaking in an arch type structure was analyzed in detail in Ref. [18] where, by means of numerical experiments, it was found that under specific circumstances the arch manifests an asymmetric response which is favored over the symmetric one. This was supported also by the finding that the snap-through voltage is lower than predicted by the symmetric model. In Ref. [23], where the electromechanical behavior of a string which serves as a special reduction of a beam was considered, a conclusion was made that the third symmetric mode dominates the symmetric response if the string is in very close proximity to the electrode. These has led to the investigation of both symmetric and non-symmetric buckling of the curved beam, presented in Ref. [22] where explicit symmetric buckling and symmetry breaking criteria were formulated in terms of the beam geometric characteristics, namely its thickness, elevation and its distance from the electrode. It was shown, that if the ratio of the beam elevation to thickness is high enough, an asymmetric response manifests itself. In addition, it was shown that in the case of the electrostatic loading, both symmetric and asymmetric snap-through may take place in beams with lower initial elevation/curvature when compared to the

* Corresponding author. Tel.: +972 036405299.
E-mail address: liormedi@post.tau.ac.il (L. Medina).

http://dx.doi.org/10.1016/j.sna.2014.10.016
0924-4247/

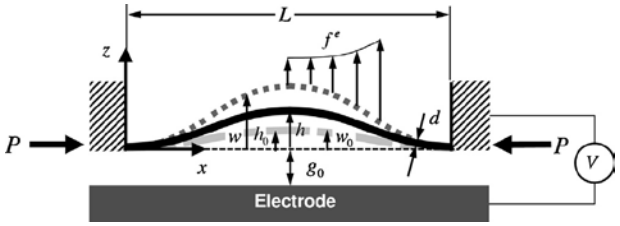


Fig. 1. Model of an initially curved axially loaded double-clamped beam actuated by distributed electrostatic force f^e. The low light gray dashed line corresponds to the designed initial shape with initial nominal elevation h_0. The black solid line represents beam actual initial shape with an increased elevation h developed due to the presence of internal axial compression force P. The upper gray dashed line corresponds to the deformed configuration caused by the electrostatic loading. Positive directions of the beam deflection and of the force, consistent with the sign convention are shown.

case of "mechanical" deflection-independent loading. An initially straight beam with curvature granted by buckling it using an applied axial force at its edges, was studied in Ref. [24]. The study derived symmetric buckling and symmetry breaking criteria in terms of the beam geometry and axial load. The extension of both cases to a general model was presented in Ref. [25]. It was deduced that the axial compression, which increases the curvature of the beam, reduces the minimal initial stress-free (as designed) values of the elevation guaranteeing bistability and symmetry breaking. It should be noted that while relatively large number of theoretical results were reported (e.g., see [26–30]), only a few experimental investigations of the behavior of curved beams, electrostatically actuated by parallel plate electrode were reported [27,31].

In this work, the results of an experimental study are presented and compared with the theoretical predictions of a previously derived model for a pre-stressed initially curved micro beam subjected to electrostatic actuation.

The schematics of the micro beam under consideration is shown in Fig. 1. It is a flexible initially curved, axially loaded, double clamped prismatic micro beam of length L having a rectangular cross-section of width b and thickness d. The beam is assumed to be made of homogeneous isotropic linearly elastic material with Young's modulus E, and Poisson's ratio ν (an effective plain strain modulus $\tilde{E} = E/(1-\nu^2)$ is used). The initial as-designed shape of the stress-free beam is described by the function $w_0(x) = h_0 z_0(x)$, where h_0 is the elevation of the beam central point above it's ends, and $z_0(x)$ is a non-dimensional function such that $\max_{x\in[0,L]}[z_0(x)] = 1$. As previously mentioned, an axial force, P, which is expected to appear upon fabrication, changes the beam designed initial elevation h_0 to a different elevation designated as h.

2. Theoretical background

The model assumes that $d \ll L$, $h \ll L$ and that the deflections are small with respect to the beam length. Under these assumptions, the beam behavior is described in the framework of the Euler–Bernoulli theory combined with the shallow arch approximation. The integro-differential equilibrium equation of the beam is as follows

$$\tilde{E}I_{yy}\left(\frac{\partial^4 w}{\partial x^4} - \frac{\partial^4 w_0}{\partial x^4}\right) + \left(P - \frac{\tilde{E}A}{2L}\int_0^L \left(\left(\frac{\partial w}{\partial x}\right)^2 - \left(\frac{\partial w_0}{\partial x}\right)^2\right) dx\right)\frac{\partial^2 w}{\partial x^2} - f^e = 0 \quad (1)$$

Table 1
Non-dimensional quantities.

$\hat{x} \triangleq x/L$	Coordinate
$\hat{w} \triangleq w/g_0$, $\hat{w}_0 \triangleq w_0/g_0$	Elevation/initial elevation
$\hat{h}_0 \triangleq h_0/g_0$	Initial midpoint elevation
$\hat{h} \triangleq h/g_0$	Midpoint elevation in post axial load application
$\hat{d} \triangleq d/g_0$	Thickness
$\alpha \triangleq (g_0^2 A)/(2I_{yy})$	Stretching parameter
$\hat{P} \triangleq (PL^2)/(\tilde{E}I_{yy})$	Axial load
$\beta \triangleq (\epsilon_0 b V^2 L^4)/(2g_0^3 \tilde{E} I_{yy})$	Voltage parameter

and it is subjected to the homogeneous boundary conditions

$$w(0) = w(L) = 0, \quad \frac{\partial w}{\partial x}(0) = \frac{\partial w}{\partial x}(L) = 0$$

Here A and I_{yy} are the cross-section's area and moment of inertia, respectively; $w(x)$ is the elevation of the beam axis above its edges and $f^e(x)$ represents the applied distributed electrostatic load provided by an electrode located at a distance g_0 (the gap) from the beam ends and extended beyond its ends [19]. For the sake of simplicity, the electrostatic load is taken, in this stage, in the form of the parallel plate approximation

$$f^e = -\frac{\epsilon_0 b V^2}{2(g_0 + w)^2} \quad (2)$$

where $\epsilon_0 = 8.854 \times 10^{-12}$ F/m is the permittivity of the free space and V is the voltage difference between the beam and the electrode. Note that P is taken positive when compressive and the positive direction of f^e coincides with the positive direction of the z-axis, as illustrated in Fig. 1.

2.1. Reduced order model

A reduced order (RO) model resulting from the Galerkin decomposition which is based on the following representation of the beam shape

$$\hat{w}(\hat{x}) = \Sigma q_i \varphi_i(\hat{x}) \quad (3)$$

with buckling modes of a straight beam used as the base functions, φ_i (normalized in such a way that $\max_{\hat{x}\in[0,1]}\left[\varphi\left(\hat{x}\right)\right] = 1$) is constructed [25]. Here $\hat{w}$ is the non-dimensional elevation of the beam above its ends, $\hat{x}$ is a non-dimensional coordinate and q_i is the non-dimensional generalized (modal) coordinate. Note that hereafter we use non-dimensional representation until otherwise is stated. The non-dimensional parameters used in the development are listed in Table 1. The Galerkin decomposition converts Eq. (1) to a system of coupled nonlinear algebraic equations. As was shown in Ref. [25], for the investigation of the symmetric and asymmetric snap-through, it is sufficient to include the first symmetric and the first anti-symmetric degrees of freedom (DOF), yielding the following system of two equations

$$b_{11}(\hat{q}_1 - \hat{h}_0) - (\hat{P} - \alpha(s_{11}(\hat{q}_1^2 - \hat{h}_0^2) + s_{22}\hat{q}_2^2))s_{11}\hat{q}_1 = -\beta\int_0^1 \frac{\varphi_1}{\left(1 + \hat{q}_1\varphi_1 + \hat{q}_2\varphi_2\right)^2} d\hat{x} \quad (4)$$

$$b_{22}\hat{q}_2 - \left(\hat{P} - \alpha\left(s_{11}\left(\hat{q}_1^2 - \hat{h}_0^2\right) + s_{22}\hat{q}_2^2\right)\right)s_{22}\hat{q}_2 = -\beta\int_0^1 \frac{\varphi_2}{\left(1 + \hat{q}_1\varphi_1 + \hat{q}_2\varphi_2\right)^2} d\hat{x} \quad (5)$$

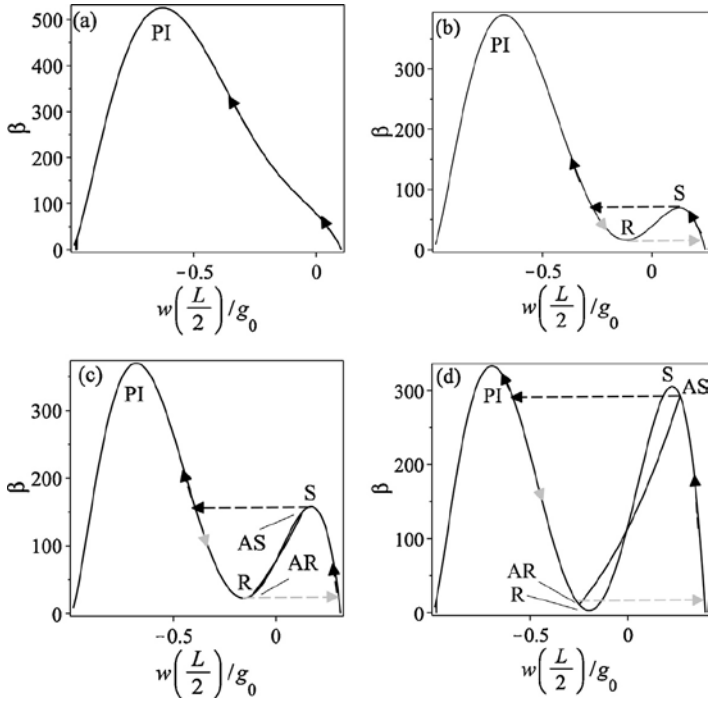


Fig. 2. Typical buckling diagrams of the electrostatically loaded beam. The scenario depicted varies depending on the beam geometrical parameters (thickness to gap ratio d/g_0, initial elevation to gap ratio h_0/g_0) and on the axial force P/P_E: (a) $d/g_0 = 0.2$, $h_0/g_0 = 0.2$ and $P/P_E = -1.5$, (b) $d/g_0 = 0.2$, $h_0/g_0 = 0.1$ and $P/P_E = 1.5$, (c) $d/g_0 = 0.2$, $h_0/g_0 = 0.2$ and $P/P_E = 1.5$, (d) $d/g_0 = 0.2$, $h_0/g_0 = 0.3$ and $P/P_E = 1.5$. Points *S* and *R* are the snap-through and release limit points; points *AS* and *AR* are the bifurcation points of the asymmetric snap-through and release and point PI is the pull-in point. The segments S-R, AS-AR, and that to the left of PI represent unstable equilibrium states. The black arrows represent the direction of loading along the stable equilibrium branches under quasi-static loading. The gray arrows represent the unloading path. The dashed lines correspond to the snap-through and snap-back, be it symmetric or asymmetric.

Here, the coefficients b_{ij} and s_{ij} are associated with the bending and stretching stiffness of the beam, respectively, and are given by the expressions

$$b_{ij} = \delta_{ij} \int_0^1 \varphi_i'' \varphi_j'' dx \quad s_{ij} = \delta_{ij} \int_0^1 \varphi_i' \varphi_j' dx \tag{6}$$

where δ_{ij} is the Kronecker delta. For the adopted base functions, $b_{11} = 2\pi^4$, $b_{22} = 1667.962$, $s_{11} = \pi^2/2$ and $s_{22} = 20.653$. Note that for a single mode representation, the normalized location of the beam is constrained in $-1 \le q_1 \le h/g_0$. Also note that since the base functions are the exact buckling modes of an initially straight beam, we have $b_{11}/s_{11} = 4\pi^2 = \hat{P}_E^{(1)} \triangleq \hat{P}_E$ as the lowest non-dimensional buckling load and $b_{22}/s_{22} = \hat{P}_E^{(2)}$ as the second non-dimensional buckling load. Eqs. (4) and (5) indicate that the relation between the voltage parameter β and the deflection, depends on the geometric parameters h_0, α and the axial force P.

2.2. Snap-through criteria

Typical responses of an initially curved pre-stressed electrostatically loaded beam, obtained by the solution of Eqs. (4) and (5) are presented in Fig. 2. The response may exhibit one or three limit

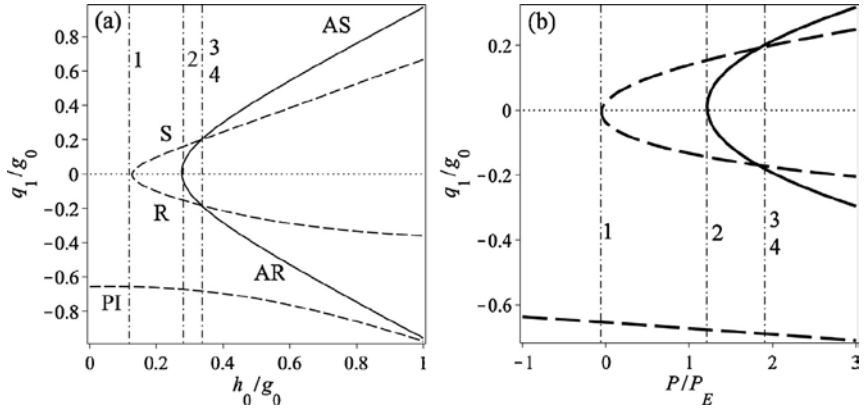


Fig. 3. Buckling maps: location of the critical points of the electrostatically loaded beam for $d/g_0 = 0.2$. (a) Beam midpoint location (q_1/g_0) vs. initial elevation (h_0/g_0) for $P/P_E = 0.5$. (b) Beam midpoint location (q_1/g_0) vs. axial force (P/P_E) for $h_0/g_0 = 0.2$. The dashed lines correspond to the symmetric snap-through (*S*), symmetric release (*R*) and pull-in (*PI*) limit points of the buckling diagram, and the solid lines correspond to the asymmetric snap-through (*AS*), and release (*AR*) bifurcation points of the buckling diagram. The vertical dashed-dotted lines correspond to the symmetric snap-through criterion (1) and the asymmetric snap-through criteria (2)–(4).

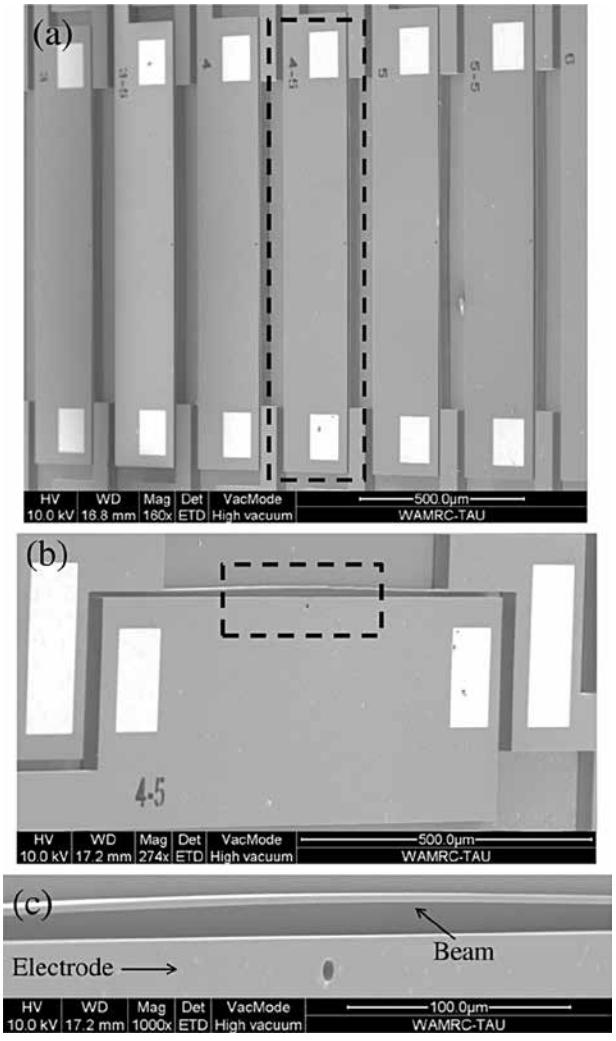


Fig. 4. Scanning electron microscope micrographs of the fabricated devices: (a) the set of the beams of differing initial elevations (numbers), (b) a single beam and the electrode and (c) the beam and the electrode in the vicinity of the beam center.

points: symmetric snap-through (S), release (R) and pull-in (PI), and two bifurcations points: asymmetric snap-through (AS) and release (AR). While the existence of three limit points implies bistable, the appearance of bifurcation points results in an asymmetric snap-through provided that the bifurcations are located on a stable part of the symmetric response curve. As the scenario depends on the beam geometrical parameters and on the axial compression force, criteria defining the conditions for which symmetric and asymmetric snap-through exist, in terms of α, h_0 and P.

The location, q_1, of the symmetric snap-through (S), release (R) and pull-in (PI) limit points are derived from the single DOF equation (Eq. (4) with $q_2 = 0$) governing the symmetric response [25]. The bifurcation points are derived by linearizing Eqs. (4) and (5) in terms of $q_2/g_0 \ll 1$ around the path $q_2/g_0 = 0$. The requirement that the resulting eigenvalue problem has a non-trivial solution provides the location, q_1 of the asymmetric snap-through (*AS*) and the asymmetric release (*AR*) points at which asymmetric branches of the response bifurcate from the symmetric one. The resulting buckling maps for a beam of $d/g_0 = 0.2$, are shown in Fig. 3(a) and (b) for a specific axial load and a specific elevation parameter, respectively (see Ref. [25]).

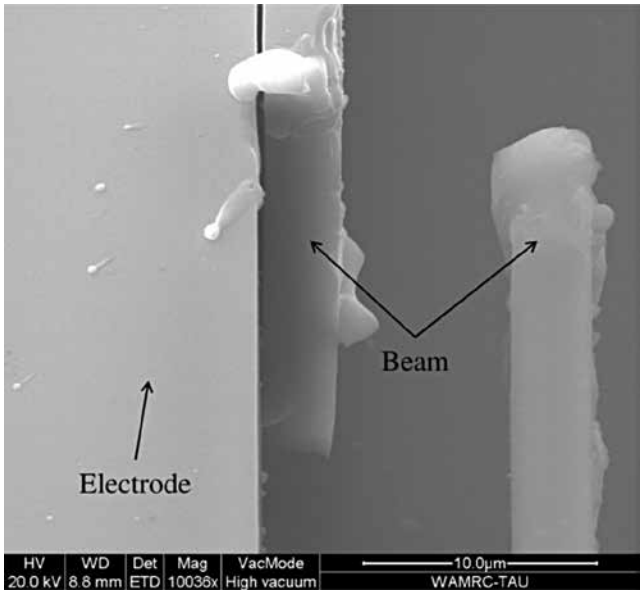


Fig. 5. A broken beam section revealing a non uniform thickness *d*.

Fig. 3 shows, in terms of q_1, the location of critical points associated with the limit point and bifurcation buckling. The left vertical dashed-dotted line, 1, represents the symmetric snap-through criterion, as for the conditions corresponding to a point placed in the region to its right, three limit points exist guaranteeing bistable (namely a response of the type shown in Fig. 2(b–d). The next vertical dashed-dotted line, 2, represents the necessary conditions for the appearance of bifurcation points (namely the response of the type shown in Fig. 2(c and d)). The right vertical dashed-dotted lines, 3, 4, which in the present case coincide, represent the sufficient asymmetric snap-through and release criteria, respectively. For the conditions corresponding to a point placed in the region to its right, the bifurcation points are placed on the stable part of the symmetric response branch and precede the limit points (like in Fig. 2(d)). From Fig. 3 it is possible to deduce that a beam of a specific thickness exhibits asymmetric response when its elevation is high enough or when it is subjected to a high enough axial force. Upon these observations, criteria guaranteeing symmetric and asymmetric snapping were developed and presented with full details in Ref. [22,25] together with a comprehensive case study mapping various possible scenarios.

In the following section, the results of the experimental investigation of the beam snap-though buckling are presented. It is shown that the beam responses are in accordance with the mathematical model and the theoretical snap-through criteria can be used for the prediction of the location of the critical snap-through and bifurcation points.

3. Experiment

The experimental study was carried out using two different dies comprised of forty curved beams each. The dies were fabricated of highly doped single crystal Si using silicon on insulator (SOI) wafers as a starting material and micromachined using deep reactive ion etching (DRIE) based process. Further description of the fabrication stages can be found in Refs. [19,32]. The beams have equal nominal width $b = 20\,\mu m$, thickness $d = 3.5\,\mu m$, gap $g_0 = 10\,\mu m$, length $L = 1000\,\mu m$ and various nominal initial elevations ranging from $h_0 = 0\,\mu m$ to $h_0 = 7\,\mu m$, Fig. 4. A cut section of one of the beams is visible in Fig. 5 revealing a variation of its thickness from 3.22 µm

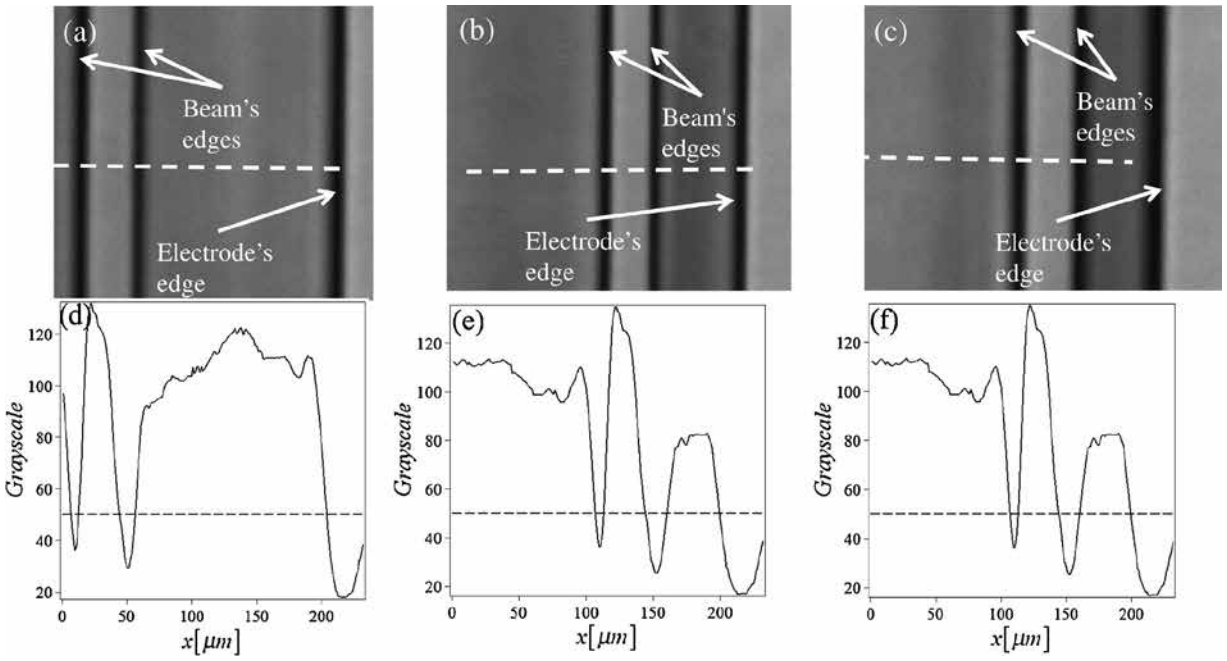


Fig. 6. Snapshots and corresponding gray level profiles along the beam midpoint section (marked by the white dashed line), for a beam with nominal initial elevation h_0 = 3 μm, actual initial elevation h = 3.41 μm, thickness d = 3.23 μm and gap g_0 = 10.69 μm. The selected threshold level is 50 and is depicted by the dashed line. (a) and (d) Initial position, corresponding to an actuation voltage of 0 V, (b) and (e) under actuation of 140 V after the occurrence of snap-through, (c) and (f) under actuation of 112 V before the release (snap-back).

at its upper edge to 3.14 μm at the bottom edge indicating, that the beams thickness is not necessarily uniform and its cross-section is not necessarily rectangular.

The dies were mounted on a wafer prober Karl Suss (PSM6) and were operated at room temperature and in ambient air conditions. The electrical signal was generated by an arbitrary function generator TGA1241, monitored by Tektronix TDS2004B oscilloscope, and amplified by one of two different amplifiers with gains of ×20 (custom built Pitronot Ltd.) or ×100 (Trek PZS350 A dual channel). The visualized in-plane (parallel to the wafer surface) motion of the beams was registered using an optical trinocular microscope (Mitutoyo FS70L-S0/100 switchable microscope) and a 1600 × 1200 pixels CCD camera UEye UI-225SE-M mounted on the microscope.

Every beam was loaded by an electrostatic force until it experienced a pull-in or, alternatively, a snap-through collapse (be it symmetric or asymmetric). An electric current limitation circuit was used in order to prevent excessive heating and damaging of the beam due to the electric shortening of the beam with the electrode. This allowed release of the beam to its initial configuration after switching off the voltage. Both the micro beams and silicon substrate were placed at ground voltage, while a triangular voltage signal was applied to the electrode in the following manner

$$V\left(\hat{t}\right) = V_0\left(\frac{\hat{t}}{\hat{t}_0} - 2\frac{\hat{t}-\hat{t}_0}{\hat{t}_0}H\left(\hat{t}-\hat{t}_0\right)\right), \quad 0 \le \hat{t} \le 2\hat{t}_0 \tag{7}$$

where V_0 is the maximal voltage and $H(\hat{t})$ is the Heaviside step function. The rise time, $\hat{t}_0$, stands on 30 s, so that the signal can be considered as quasi-static. This quasi-static voltage signal does not initiate a premature dynamic snap-through [21] but moves the beam along its equilibrium path, such as those presented in

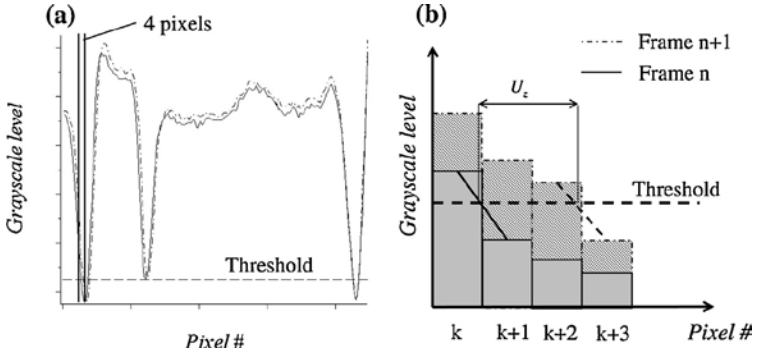


Fig. 7. An illustration of the interpolation between two adjacent pixels. (a) The gray scale profile extracted from two subsequent frames corresponding to different actuation voltages. (b) Close up on the gray scale of four pixels for which the gray scale level is around the threshold value (dashed line), with the dark and light gray corresponding to the solid and dotted profiles, respectively. The displacement of the beam is defined by following the point where the interpolation line crosses the threshold line, and is the distance between it position on two subsequent frames (denoted by U_z in the figure).

Table 2
Average values and maximum and relative errors of Chip # 1.

	Nominal [μm]	Average [μm]	Measurement error [μm]		Relative error [%]	
d	3.5	3.63	Δd	0.06	$\Delta d/d$	1.59
g_0	10	10.42	Δg_0	0.06	$\Delta g_0/g_0$	0.54
$h+g_0$			$\Delta(h+g_0)$	0.06	$\Delta(h+g_0)/(h+g_0)$	0.51
h			Δh	0.08	$\Delta h/h$	1.41
P/P_E		0.83	$\Delta(P/P_E)$	0.12	$\Delta(P/P_E)/(P/P_E)$	19.21

Fig. 2. As the beam moves, it either snaps symmetrically at the limit point or asymmetrically at the bifurcation point (if it manifest itself before the limit point). If the pull-in voltage is higher than the snap-through, the beam will snap to the second stable equilibrium, then, as the signal decreases it will return along that stable path and snap-back to the initial stable path leading to its initial state. However, if the pull-in voltage is lower than the snap-through, than the beam will snap to the electrode.

In order to build the voltage-deflection dependence, we used a simple image processing procedure implemented in MAPLE and based on edge tracking [19,32,33]. The response of the beam was video recorded and the obtained video file was split into separate frames, each one corresponding to a specific time and hence to a specific voltage value. Three such frames are presented in Fig. 6(a)–(c). Next, a one-dimensional array (row) of pixels which is one pixel wide (marked by the white dashed line in Fig. 6) was extracted from each of the frames. The quantitative description of the gray scale level along this array forms the gray scale profile shown in Fig. 6(d) and (e) (lower levels (notches) of the gray level profile correspond to the edges of the beam and of the electrode, which look more dark on the image). Since we are looking for the displacements of the beam, it is to say for the increment of the beam edge coordinate (in terms of pixels) between two subsequent frames rather than for the precise location of the edge, the coordinate of the edge is defined approximately as a cross-section of the gray level profile with a threshold line, Fig. 7. The threshold value

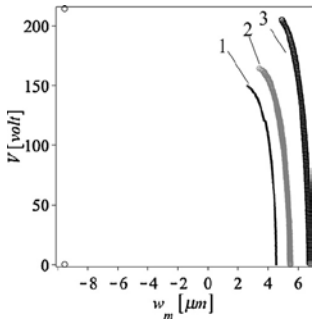


Fig. 8. Experimental results for three beams from chip # 1. Line #1: $d = 3.4$ μm, $h_0 = 3.5$ μm, $h = 4.58$ μm, $g_0 = 10.61$ μm, $P/P_E = 0.81$ Line #2: $d = 3.34$ μm, $h_0 = 4.5$ μm, $h = 5.52$ μm, $g_0 = 10.61$ μm, $P/P_E = 0.87$ Line #3: $d = 3.66$ μm, $h_0 = 6$ μm, $h = 6.77$ μm, $g_0 = 10.4$ μm, $P/P_E = 0.67$.

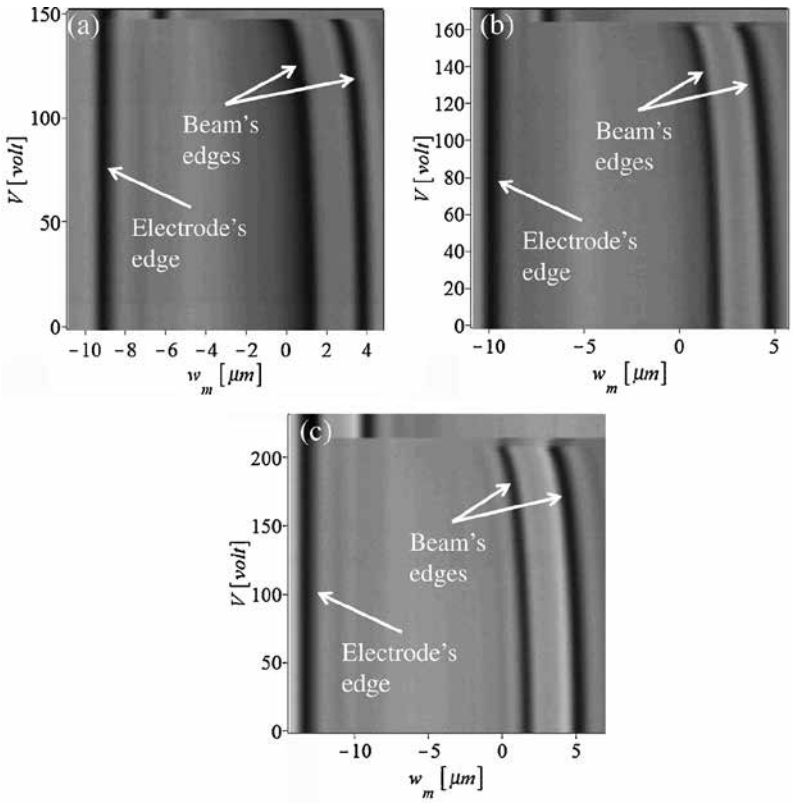


Fig. 9. Experimental results: Direct visualization of the beam movement as a result of the electrostatic load for (a) $d = 3.4$ μm, $h_0 = 3.5$ μm, $h = 4.58$ μm, $g_0 = 10.61$ μm, $P/P_E = 0.81$, (b) $d = 3.34$ μm, $h_0 = 4.5$ μm, $h = 5.52$ μm, $g_0 = 10.61$ μm, $P/P_E = 0.87$, (c) $d = 3.66$ μm, $h_0 = 6$ μm, $h = 6.77$ μm, $g_0 = 10.4$ μm, $P/P_E = 0.67$.

Table 3
Average values and maximum and relative errors of Chip # 2.

	Nominal [μm]	Average [μm]	Measurement error [μm]		Relative error [%]	
d	3.5	3.42	Δd	0.05	$\Delta d/d$	2.31
g_0	10	10.43	Δg_0	0.06	$\Delta g_0/g_0$	0.7
$h+g_0$			$\Delta(h+g_0)$	0.06	$\Delta(h+g_0)/(h+g_0)$	0.59
h			Δh	0.08	$\Delta h/h$	0.04
P/P_E		0.41	$\Delta(P/P_E)$	0.14	$\Delta(P/P_E)/(P/P_E)$	26.28

of the gray level is defined on an ad-hoc manner and is chosen by trial and error to provide lowest noise level in the final curve. In order to achieve sub-pixel resolution, a linear interpolation of the gray scale levels between the two adjacent pixels – one of which is of a higher gray level and the other is of a lower grey level than the threshold value – is used (see [34] for thresholding of a single image). The coordinate of the beam edge (in pixels) corresponds to the point where the interpolation line and the threshold line cross each other, as is shown in Fig. 7(b).

Finally, by using known spatial (μm/pixel) and temporal (V/frame number) conversion factors, the information extracted from each image was converted into a voltage-deflection relations, Fig. 8. Hence, this procedure yields the curves describing the beam response through the dependency between the voltage and the beam midpoint location. Three such curves, following the downward movement of the beam central point from its initial position defined by the elevation h, are shown in Fig. 8. The advantage of the described procedure, which is based on a common edge detection algorithm, is its simplicity, its ability to directly visualize the beam deformation and the ability to obtain reliable voltage-deflection curves with a resolution of several tens of nanometers [19,33,35]. Note that the synchronization between the video recording and the beginning of the frequency sweep was done manually, which may result in a synchronization error. For the camera frame rate of 10 FPS, this error is estimated to be of the order of several frames which in voltage is interpreted to be as high as 5 V. An example of a direct visualization of the beam movement extracted from the video captured during the experiment is shown in Fig. 9. In these images, each row of pixels corresponds to a specific different frame of the video file and consequently to a specific voltage (see Ref. [36]).

4. Results and discussion

Prior to the beams operation, we first characterized the actual geometric parameters of the tested beams by analyzing scanning electron microscope (SEM) images of every one of them. The beam thickness, d, and the gap g_0, between the electrode and the beam surface facing the electrode, were measured at the locations close to the beam ends. The distance from the same beam surface to the electrode was measured also at the midpoint of the beam. As this

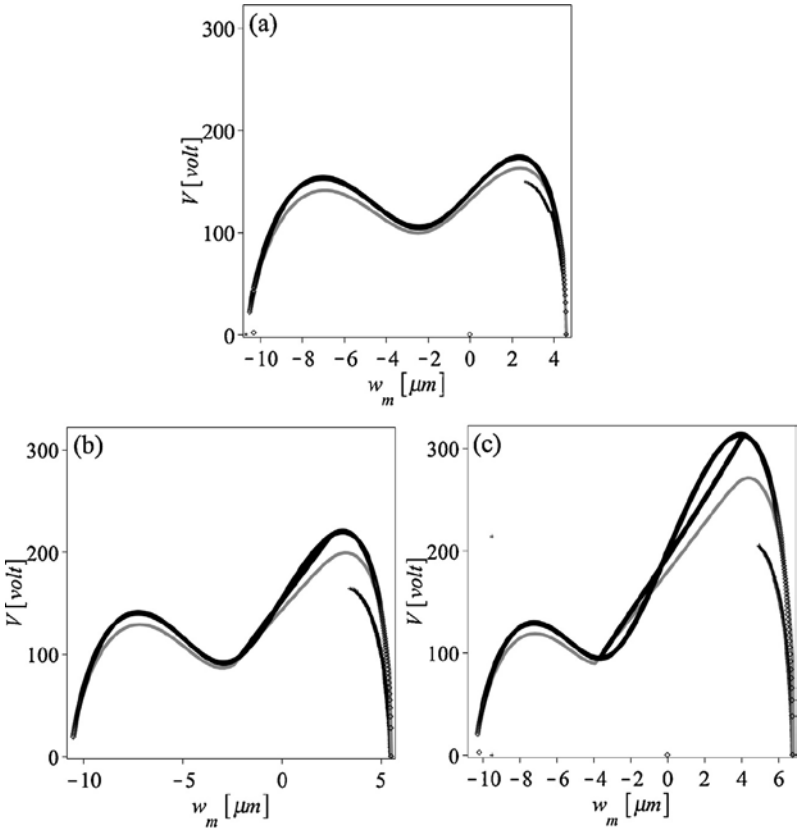


Fig. 10. Experimental and theoretical results for four beams from dies # 1 (a) $d = 3.4$ μm, $h_0 = 3.5$ μm, $h = 4.58$ μm, $g_0 = 10.61$ μm, $P/P_E = 0.81$, (b) $d = 3.34$ μm, $h_0 = 4.5$ μm, $h = 5.52$ μm, $g_0 = 10.61$ μm, $P/P_E = 0.87$, (c) $d = 3.66$ μm, $h_0 = 6$ μm, $h = 6.77$ μm, $g_0 = 10.4$ μm, $P/P_E = 0.67$. The thin black line represents the experimental result obtained through image processing, the thick black line represents the 2 DOF RO model with the parallel plate approximation and the thick gray line represents the numerical analysis (finite differences) with the Meijs–Fokkema fit.

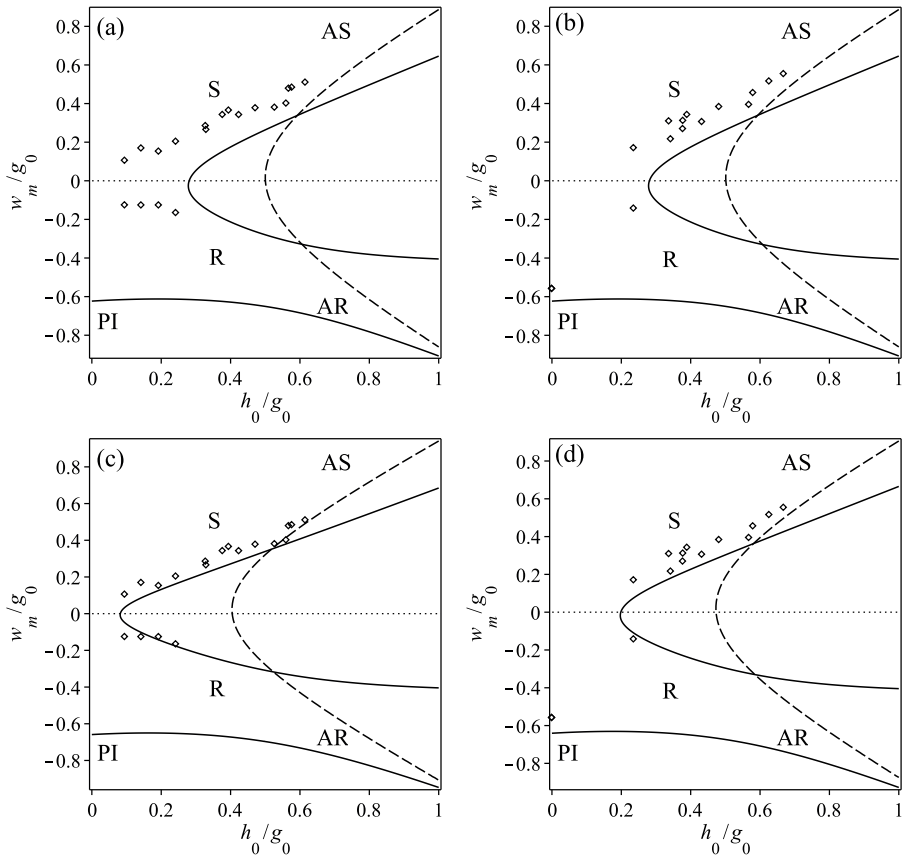


Fig. 11. Buckling maps, location of the critical points of the electrostatically loaded beam obtained through the experiments (circles) and the 2 DOF model (lines), for chip # 1 with average thickness to gap ratio of $d/g_0 = 0.34$ and an average axial load of (a) $P/P_E = 0$ and (c) $P/P_E = 0.81$, and chip # 2 with average thickness to gap ratio of $d/g_0 = 0.32$ and an average axial load of (b) $P/P_E = 0$ and (d) $P/P_E = 0.41$. The solid lines show the theoretically predicted location of the symmetric snap-through (*S*), symmetric release (*R*) and pull-in (*PI*) limit points, and the dashed lines correspond to the asymmetric snap-through (*AS*), and release (*AR*) bifurcation points.

quantity encompasses the sum of the beam central elevation and the gap, (measured at the beam end), subtraction of the latter yields the actual central elevation, h.

In general, it was found that the evaluated actual initial elevation, h, is larger than the nominal initial as-designed elevation h_0 in a stress-free beam. This indicates the presence of a residual compressive axial force, which can be estimated on the basis of the difference between the actual h and nominal h_0 initial central elevation of the beam by using Eq. (4), with $\beta = 0$, $\hat{q}_2 = 0$, $\hat{q}_1 = \hat{h}$

$$\frac{P}{P_E} = 1 - \frac{h_0}{h} + \frac{3}{4}\frac{1}{d^2}\left(h^2 - h_0^2\right) \tag{8}$$

Note that assuming the initial shape of the beam is an actual buckling mode, Eq. (8) is exact (Ref. [9]). To estimate the possible error of the evaluated force, note that the error of a measured length reflects the resolution of the image and is one pixel. Its value, Δ_L, in terms of μm, is defined by the average μm to pixel conversion. The maximal length measurement error in each of the dies is given in Tables 2 and 3 along with the average value of the measured parameters. The error of a parameter y which depends on measured parameters, x_i is given by

$$\Delta y = \sqrt{\sum_{i=1}^{n}\left(\frac{\partial y}{\partial x_i}\Delta x_i\right)^2} \tag{9}$$

which for the initial elevation h, calculated as the difference between the measured sum of the elevation and the gap $h + g_0$ and the measured gap g_0, yields the expression $\Delta h = \sqrt{2}\Delta_L$. For the axial load (derived from Eq. (8)) the error is

$$\Delta\left(\frac{P}{P_E}\right) = \frac{1}{d^2}\sqrt{2\left(\frac{d^2 h_0}{h^2} + \frac{3h}{2}\right)^2 + \frac{9\left(h^2 - h_0^2\right)^2}{4d^2}}\,\Delta_L \tag{10}$$

assuming that the error of the initially designed beam elevation is $\Delta h_0 \equiv 0$, namely, the nominal value exactly defines the initial elevation of the curved stress-free beam. The maximal values of this error are given in Tables 2 and 3.

Having acquired the measured geometric parameters of the tested beams and the evaluated residual compressive force, it is possible to employ the model (Section 2) to derive response curves which can be compared with the experimental ones shown in Fig. 8. The latter are shown by thin black lines in Fig. 10, together with the theoretical predictions based on the lumped two DOF model, Eqs. (4) and (5) which are shown by a thick black line. The comparison between these curves implies that, in general, the two DOF model predicts well the location, w_m, at the critical (limit or bifurcation snap-through) points, while the predicted critical voltage is higher than the measured one. This discrepancy can be attributed to several reasons. The use of low order models typically results in over-estimation of the stiffness. Consequently, as was found in Ref. [25], the direct numerical solution of the equilibrium equations (finite difference in conjunction with the arc-length continuation method used for the tracking of the unstable branches of the equilibrium curve) provides similar position of the critical points but

lower and thus more accurate corresponding voltage parameters. In addition, fringing field effects, which are neglected by the parallel plate approximation of the electrostatic force, may have significant contribution into the electrostatic force. In order to include these effects, we use the Meijs–Fokkema fit [19,26,37]

$$f^e = \frac{\epsilon_0 b V^2}{2(g_0+w)^2}\left(1+0.265\left(k\frac{g_0+w}{b}\right)^{3/4} + k\ \ 0.53\left(\frac{d}{b}\sqrt{\frac{g_0+w}{d}}\right)\right) \quad (11)$$

for the calculation of the electrostatic force in the framework of the finite difference models. In order to take into account the fact that due to the chip design [19] fringes at one side of the beam width (the one facing the substrate) are blocked, $k = 1/2$ is used instead of $k = 1$ in the original expression. The predictions of the finite differences solution of the equilibrium equations (stated in Eqs. (1) and (2) in Ref. [25]) including the fringing effects are shown in Fig. 10 by the thick grey line. An additional source of discrepancy is attributed to the previously observed non uniformity of the actual thickness of the beam (see also Ref. [38]), which affects its stiffness as well as the electrostatic force. For example, the theoretical analysis of the beams behavior from Fig. 10 with their thickness reduced by 30%, results in a 55% decrease of the predicted voltage, demonstrating its high sensitivity to the beam geometry. However, it is observed that the location of the limit or bifurcation points is predicted with much higher accuracy even by the two DOF model based on the parallel plate approximation.

For every beam, the positions of the beam midpoint, w_M, at snap-through and release were extracted from the experimental response plots, such as shown in Fig. 8. In order to evaluate the accuracy of to the theoretical model, these values were normalized with respect to the measured gap, g_0, and presented vs. the relation between the beam non-dimensional nominal elevation and the measured gap h_0/g_0 in Fig. 11(a) and (c) for chip # 1 and in Fig. 11(b) and (d) for chip # 2, which depicts, in essence, the bifurcation map. In addition to the experimental values, which are marked by circles, the location of the limit and bifurcation points predicted by the RO two DOF model, are also presented. However, a bifurcation map is force dependent, and since the evaluated axial load ranges from $P = 0.33\ P_E$ to $P = 1.24\ P_E$ in die # 1 and from $P = 0$ to $P = 1.05\ P_E$ in die # 2, each one of the dies has to be analyzed separately. While Fig. 11(a) and (b) depict the theoretical curves for initially stress-free beams, theoretical predictions shown in Fig. 11(c) and (d) include the effect of the compressive axial force, which was evaluated on the basis of the actual initial elevations of the beams in each chip. Since the estimated axial loads are scattered over a previously defined range, the average value for each of the dies is used for the theoretical analysis.

The results show that the theoretical model incorporating the axial force provides a better correlation with the acquired experimental results than the theoretical model disregarding the axial force. This indicates that such a force is present and illustrates its significant influence on the beam behavior. Furthermore, the distribution of the experimental results is consistent with the previously formulated symmetric and asymmetric snap-through criteria based on the theoretical model. A beam, having a nominal initial elevation which is smaller than that defined by the symmetric snap-through criterion, does not snap. Moreover, the points representing the experimentally monitored behavior of beams having high initial elevations approach the theoretical curve corresponding to the bifurcation points indicating the applicability of the theoretically derived symmetry breaking criterion.

Note that the maximal relative error of the evaluated axial force is larger by an order of magnitude than the maximal relative measurement error (Tables 2 and 3). Moreover, as was previously mentioned, the estimated values of the initial axial compression force in each chip are scattered over a large range. This may explain the fact that the experimental results lie close to but yet above the theoretical curves of the buckling map, which is sensitive to the value of the prestressing axial force.

5. Summary and conclusions

In this work, the snap-through buckling of an initially curved single crystal Si (silicon on insulator wafer) beams subjected to a transverse nonlinear, distributed electrostatic force was experimentally studied. The experiments were carried out on two different dies comprising of forty beams each. The beams were 1000 μm long, nominally 4 μm thick and 20 μm wide, with initial nominal designed elevations ranging from $h_0 = 0$ μm to $h_0 = 7$ μm. Direct measurements of the actual beam geometry by SEM revealed the difference between the nominal and the actual initial elevations, which was attributed to the presence of an axial force originated in the fabrication process and the SOI wafer bonding. The actual value of the axial force was estimated using the nominal and measured values of the midpoint elevations and the model describing the post buckled configuration of the beam. Each beam response was recorded and analyzed by means of image processing in order to identify the position at which the snap-through and/or snap-back occur. The experimental responses were compared with the results provided by the reduced order (Galerkin) model [25] as well as by numerical finite differences simulations both incorporating the actual measured geometric parameters. We found that while the critical snap-through and release voltages provided by the model are consistently higher than the measured values (possibly due to the interaction of the fringing fields with the actual cross-section not being exactly rectangular, and as a result of increased sensitivity of the critical voltages to the geometry and loading parameters, which are highly uncertain due to low tolerances of micro fabrication), the location of the critical points in terms of deflections are predicted by the model with better accuracy. The comparison of the experimental and theoretical bifurcation maps (built in terms of the critical snap-through, release and pull-in deflections of the beam) clearly indicate that the axial force is present and has a significant influence on the beam snap-through behavior. Moreover, the emerging of the non-symmetric buckling can be identified on the experimental buckling maps. One can conclude therefore that the snap-through behavior of the beam is in accordance with the model prediction. The theoretical criteria (in terms the location buckling map) for symmetric snap-through and symmetry breaking, which were derived in [25] are valuable and can be used for the design of microstructures incorporating initially curved bistable microbeams.

Acknowledgments

The devices were fabricated at the Cornell University NanoScale Science & Technology Facility (CNF). The authors would like to thank David Schriber, Stella Lulinski and Leeya Engel in giving their hand in the presented work and for offering the most valuable commodity a person has to offer, time. The research is supported by the Broadcom Foundation and Tel Aviv University Authentication Initiative and Ariel University.

References

[1] G.J. Simitses, Dynamic Stability of Suddenly Loaded Sturctures, Springer-Verlag, New York, 1989.
[2] P. Villagio, Mathematical Models for Elastic Structures, Cambridge University Press, Cambridge, 1997.

[3] G.J. Simitses, D.H. Hodges, Fundamentals of Structural Stability, Butterworth-Heinemann, 2006.
[4] Y. Chandra, I. Stanciulescu, L.N. Virgin, T.G. Eason, S.M. Spottswood, A numerical investigation of snap-through in a shallow arch-like model, J. Sound Vib. 332 (2013) 2532–2548.
[5] B. Moghaddasie, I. Stanciulescu, Equilibria and stability boundaries of shallow arches under static loading in a thermal environment, Int. J. Non-Linear Mech. 51 (2013) 132–144.
[6] L. Virgin, R. Wiebe, S. Spottswood, T. Eason, Sensitivity in the structural behavior of shallow arches, Int. J. Non-Linear Mech. 58 (2014) 212–221.
[7] B. Camescasse, A. Fernandes, J. Pouget, Bistable buckled beam and force actuation: experimental validations, Int. J. Solids Struct. 51 (2014) 1750–1757.
[8] G. Schmidt, J. Yu, Vibrations of arches and onset of chaos, ZAMM-J. Appl. Math. Mech.[[nl]]Zeitschrift für Angewandte Mathematik und Mechanik 73 (1993) 349–356.
[9] S. Emam, A. Nayfeh, On the nonlinear dynamics of a buckled beam subjected to a primary-resonance excitation, Nonlinear Dyn. 35 (2004) 1–17.
[10] V. Kaajakari, Practical MEMS, Small Gear Publishing, 2009 http://books.google.co.il/books?id=nX5mPgAACAAJ
[11] Y. Zhu, H.D. Espinosa, Effect of temperature on capacitive RF MEMS switch performance – a coupled-field analysis, J. Micromech. Microeng. 14 (2004) 1270.
[12] I. Stanciulescu, T. Mitchell, Y. Chandra, T. Eason, M. Spottswood, A lower bound on snap-through instability of curved beams under thermomechanical loads, Int. J. Non-Linear Mech. 47 (2012) 561–575.
[13] Z.P. Bažant, L. Cedolin, Stability of Structures. Elastic, Inelastic, Fracture and Damage Theories, Dover Publications Inc., Mineola, New-York, 1991.
[14] D.J. Joe, Y. Linzon, V.P. Adiga, R.A. Barton, M. Kim, B. Ilic, S. Krylov, J.M. Parpia, H.G. Craighead, Stress-based resonant volatile gas microsensor operated near the critically buckled state, J. Appl. Phys. 111 (2012) 104517.
[15] L.D. Gabbay, S. Senturia, Computer-aided generation of nonlinear reduced-order dynamic macromodels. I. Non-stress-stiffened case, J. Microelectromech. Syst. 9 (2000) 262–269.
[16] L. Zhang, Y.-P. Zhao, Electromechanical model of rf mems switches, Microsyst. Technol. 9 (2003) 420–426.
[17] Y. Zhang, Y.-p. Zhao, Numerical and analytical study on the pull-in instability of micro-structure under electrostatic loading, Sensors Actuat. A: Phys. 127 (2006) 366–380.
[18] K. Das, R.C. Batra, Symmetry breaking, snap-through and pull-in instabilities under dynamic loading of microelectromechanical shallow arches, Smart Mater. Struct. 18 (2009), article number 115008.
[19] S. Krylov, B. Ilic, D. Schreiber, S. Seretensky, H. Craighead, The pull-in behavior of electrostatically actuated bistable microstructures, J. Micromech. Microeng. 18 (2008) 055026.
[20] M. Younis, MEMS Linear and Nonlinear Statics and Dynamics, Microsystems Series, Springer, 2011 http://books.google.co.il/books?id=5wznjKOsmiQC
[21] S. Krylov, N. Dick, Dynamic stability of electrostatically actuated initially curved shallow micro beams, Continuum Mech. Thermodyn. 22 (2010) 445–468.
[22] L. Medina, R. Gilat, S. Krylov, Symmetry breaking in an initially curved micro beam loaded by a distributed electrostatic force, Int. J. Solids Struct. 49 (2012) 1864–1876.
[23] S. Krylov, S. Seretensky, Higher order correction of electrostatic pressure and its influence on the pull-in behavior of microstructures, J. Micromech. Microeng. 16 (2006) 1382.
[24] L. Medina, R. Gilat, S. Krylov, Symmetry breaking criteria in electrostatically loaded bistable curved/prebuckled micro beams, in: Spontaneous Symmetry Breaking, Self-Trapping, and Josephson Oscillations, Springer, Berlin, Heidelberg, 2013, pp. 679–705.
[25] L. Medina, R. Gilat, S. Krylov, Symmetry breaking in an initially curved pre-stressed micro beam loaded by a distributed electrostatic force, Int. J. Solids Struct. 51 (2014) 2047–2061.
[26] R.C. Batra, M. Porfiri, D. Spinello, Electromechanical model of electrically actuated narrow microbeams, J. Microelectromech. Syst. 15 (2006) 1175–1189.
[27] Y. Zhang, Y. Wang, Z. Li, Y. Huang, D. Li, Snap-through and pull-in instabilities of an arch-shaped beam under an electrostatic loading, J. Microelectromech. Syst. 16 (2007) 684–693.
[28] H.M. Ouakad, M.I. Younis, F.M. Alsaleem, R. Miles, W. Cui, The static and dynamic behavior of MEMS arches under electrostatic actuation, in: ASME, International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2009, San Diego, CA, USA, 2009, pp. DETC2009-87024.
[29] E.M. Abdel-Rahman, M.I. Younis, A.H. Nayfeh, Characterization of the mechanical behavior of an electrically actuated microbeam, J. Micromech. Microeng. 12 (2002) 759.
[30] S.A. Alkharabsheh, M.I. Younis, Statics and dynamics of mems arches under axial forces, J. Vib. Acoust. 135 (2013) 021007.
[31] S. Krylov, B.R. Ilic, S. Lulinsky, Bistability of curved micro beams actuated by fringing electrostatic fields, Nonlinear Dyn. 66 (2011) 403–426.
[32] N. Dick, Dynamic Stability of Electrostatically Actuated Curved Micro beams, The Iby and Aldar Fleischman Faculty of Engineering, The Zandman-Slaner School of Graduate Studies – Tel Aviv University, Ramat Aviv, Tel Aviv, 2010 (Master of Science Thesis).
[33] A. Ya'akobovitz, D. Copic, J.D. Beroz, A.J. Hart, Nanoscale displacement measurement of microdevices via interpolation-based edge tracking of optical images, J. Micromech. Microeng. 23 (2013) 045004.
[34] R. Gonzalez, R. Woods, Digital Image Processing, Pearson/Prentice Hall, 2008 http://books.google.co.il/books?id=8uGOnjRGEzoC
[35] A. Ya'akobovitz, S. Krylov, Y. Hanein, Nanoscale displacement measurement of electrostatically actuated micro-devices using optical microscopy and digital image correlation, Sensors Actuat. A: Phys. 162 (2010) 1–7.
[36] Y. Gerson, I. Sokolov, T. Nachmias, B. Ilic, S. Lulinsky, S. Krylov, Pull-in experiments on electrostatically actuated microfabricated MESO scale beams, Sensors Actuat. A: Phys. 199 (2013) 227–235.
[37] K. Das, R.C. Batra, Pull-in and snap-through instabilities in transient deformations of microelectromechanical systems, J. Micromech. Microeng. 19 (2009), article number 035008.
[38] L. Banks-Sills, Y. Hikri, S. Krylov, V. Fourman, Y. Gerson, H.A. Bruck, Measurement of poisson's ratio by means of a direct tension test on micron-sized specimens, Sensors Actuat. A: Phys. 169 (2011) 98–114.

Biographies

Lior Medina received both B.Sc. (2010) and M.Sc. (2012) in mechanical engineering from Tel-Aviv University. The M.Sc. thesis was on the subject "Symmetry breaking in electrostatically loaded curved bistable micro beams" under the supervision of Professor Slava Krylov & Doctor Rivka Gilat. In December 2012 he began his Ph.D. studies in mechanical engineering at the School of Mechanical Engineering, Faculty of Engineering in Tel-Aviv University, under the combined supervision of Professor Slava Krylov & Doctor Rivka Gilat. His Ph.D. research is focused on bistable microstructures under electrostatic loading.

Rivka Gilat is an associated professor in the department of Civil Engineering in Ariel University. She received her B.Sc. in Civil Engineering from The Technion, Haifa, and M.Sc. and Ph.D. in Solid Mechanics Materials and Structures from Tel-Aviv University. Her research interest includes composite materials, structural modeling and structural stability.

Bojan Ilic received the M.S. and Ph.D. degrees in applied physics from Cornell University, Ithaca, NY, in 2002 and 2006, respectively. From 2002, he was a Research Associate and a User Program Manager at the Cornell Nanoscale Science and Technology Facility, Cornell University. In October 2014, Rob became a project leader at the National Institute of Standards and Technology in Gaithersburg. He is the author or coauthor of more than 180 journal publications in the field of optics, quantum electronics, scanning probes, electrochemical and biological sensors, micro- and nanoelectromechanical systems, nanofluidics, flux pinning in thin-film superconductors, and nanomagnetics. His current research interests include the development of nanofabrication technologies for building fully integrated molecular scale devices, use of micro- and nanomechanical resonant sensors for novel chemical and biological detection schemes, nanofluidics, atomic force probes, and nanomagnetics.

Slava Krylov holds M.Sc. (1989) and Ph. D. (1993) in applied mechanics, both from the State Marine Technical University of St. Petersburg, Russia. Between 1994 and 2002 he was Colton postdoctoral fellow at the Department of Solid Mechanics, Materials and Systems, Tel Aviv University, worked as a R & D engineer for Israel Aircraft Industries and as a principal scientist and co-founder of a start-up company developing optical MEMS. He joined Tel Aviv University as a faculty member in 2002 where he is currently a Professor at the School of Mechanical Engineering, Faculty of Engineering. He was visiting professor (multiple occasions) of the School of Applied and Engineering Physics, Cornell University. In parallel with his academic career he was consultant and member of an advisory board of several MEMS start-up companies. His research focuses in the area of design and modeling of micro- and nano-electromechanical systems (MEMS/NEMS), optical MEMS, electrostatic, magnetic and electro thermal actuators, inertial sensors, dynamics and stability of micro- and nanoelectromechanical devices, nano resonators for sensing applications, polymeric MEMS as well as developing of modeling and characterization tools for micro and nano devices.

Proceedings of the ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2014
August 17-20, 2014, Buffalo, New York, USA

DETC2014-34880

DYNAMIC TRAPPING IN BI-STABLE ELECTROSTATICALLY ACTUATED CURVED MICRO BEAMS

Lior Medina*
Faculty of Engineering
School of Mechanical Engineering
Tel Aviv University
Ramat Aviv 69978, Israel
Email: liormedi@post.tau.ac.il

Rivka Gilat
Department of Civil Engineering
Faculty of Engineering
Ariel University
Ariel 44837, Israel
Email: gilat@eng.tau.ac.il

Slava Krylov
Faculty of Engineering
School of Mechanical Engineering
Tel Aviv University
Ramat Aviv 69978, Israel
Email: krylov@post.tau.ac.il

ABSTRACT

Micro and nano devices incorporating bi-stable structural elements such as micro beams are designed to exploit the fact that the latter possess two stable configurations at the same actuation force. Generally, the transition of a micro beam from one table state to another, namely the snap-through which is essentially dynamic phenomenon, can be initiated by either static or dynamic activations. In this work, results of theoretical and numerical investigations of the transient dynamics of a pre-stressed initially curved double clamped micro beams actuated by a time dependent electrostatic load are presented. We show by means of a reduced order model of a shallow beam, derived using the Galerkin procedure, that the beam may exhibit various types of responses. For certain beam characteristics, the second stable state is inaccessible under a static loading but is attainable only by means of a specially tailored dynamic actuation. This gives way to the possibility of trapping the dynamically bi-stable beam at a stable configuration which is close to the electrode by applying special loading sequences.

INTRODUCTION

Curved beams (arches) loaded by concentrated or distributed transverse forces may exhibit bi-stability, namely the existence of two different stable equilibria under the same loading. The transition between the two stable states in these structures is commonly referred to as a snap-through buckling. The behavior of beams liable to the snap-through buckling, due to prescribed deflection-independent "mechanical" loads, is a well established topic in structural mechanics [1–3] and continues to attract attention of researches [4–6].

In the case of the electrostatic actuation, the snap-through behavior is affected by the nonlinearity of the electrostatic force parameterized by the initial distance between the beam and the electrode. The criteria for symmetric (limit-point) snap-through and for asymmetric (bifurcation) snap-through of an initially curved pre-stressed bell-shaped beam were obtained in [7]. It was shown, that in the case of the electrostatic loading, both symmetric and asymmetric snap-through may take place in beams with lower initial elevation/curvature when compared to the case of “mechanical” deflection-independent loading. In addition, it was found that as the axial compression increases the minimal values of the initial elevation guaranteeing bi-stability and symmetry breaking reduce.

While even under slow loading rates the snap-through transition is a dynamic phenomenon, the significance of the dynamic nature of the response becomes paramount under dynamic loading. For that reason, the static stability analysis was extended to an initially curved beam which is dynamically actuated in [8]. The model included a step function actuation with the help of a reduced order (RO) model of the device using the Galerkin decomposition to a single degree lumped model. The conditions of dynamic instabilities for the case of a step-function loading

*Address all correspondence to this author.

by a suddenly applied voltage were formulated and it was shown that the initial elevation of the beam required for the dynamic snap-through to appear is higher than in the case of quasi-static loading. The critical dynamic snap-through and dynamic pull-in values of the voltage parameters are consistently lower and critical deflections are higher than their static counterparts.

In the present work, the stability analysis is extended to two different loading scenarios, both of which emphasize that a beam can have different responses depending on the characteristics of the dynamic electrostatic actuation. The beam may vibrate and bounce back to the original stable state, it can snap-through and then be trapped at the second stable configuration, or it may experience a snap-through followed by electrostatic (pull-in) instability. In one scenario, the single step actuation was simulated for a variety of configurations to show that on the loading-damping parameters plane lies a transient fractal region in the threshold between having a dynamic snap-through and the lack of it.

In the second scenario, the model is extended to a two step actuation signal for a configuration in which the beam's second stable region is inaccessible under static loading but is attainable by means of dynamic actuation. The analysis shows that a specially tailored, time dependent two step actuation can bring the beam to this statically inaccessible second stable condition. This gives way to the possibility of trapping the dynamically bi-stable beam at a stable configuration which is close to the electrode by applying special loading sequence.

In both instances, the phase plane analysis and the time history of the response, obtained on a basis of a lumped model, were used to build a map defining the characteristic parameters required for the beam to exhibit each of the possible scenarios. A rich behavior is observed in the boundary between the areas which define the parameters required for the beam to rest at one of its stable configurations.

THE MODEL OF A CURVED BEAM

Formulation

We consider a flexible initially curved, axially loaded, double clamped prismatic micro beam of length L having a rectangular cross-section A of width $\hat{b}$ and thickness $\hat{d}$ as shown in Fig. 1. The beam is assumed to be made of homogeneous isotropic linearly elastic material with Young's modulus E and density ρ. Since the width $\hat{b}$ of a micro-beam is typically larger than it's thickness $\hat{d}$, an effective (plain strain) modulus of elasticity $\tilde{E} = E/(1-\nu^2)$ is used, where ν is Poisson's ratio. The initial shape of the stress-free beam is described by the function $\hat{w}_0(\hat{x}) = \hat{h}_0 z_0(\hat{x})$, where $\hat{h}_0$ is the initial elevation of the beam's central point above it's ends, and $z_0(\hat{x})$ is a non-dimensional function such that $\max_{\hat{x}\in[0,L]}[z_0(\hat{x})] = 1$. The main assumption of the model is that an axial load appearing upon fabrication changes the beam's initial elevation $\hat{h}_0$ to a different elevation designated as $\hat{h}$. The beam is subjected to a distributed electrostatic force provided by an electrode located at a distance g_0 (the gap) from the beam's ends and extended beyond it's ends ([9]).

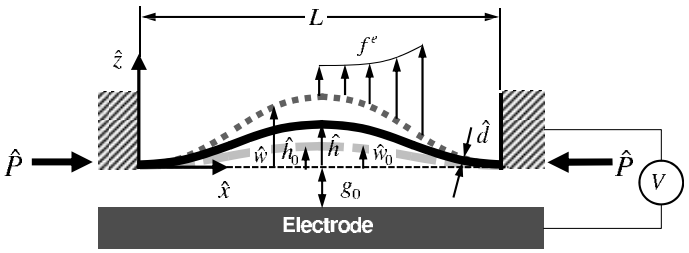


FIGURE 1: Model of an initially curved axially loaded double-clamped beam actuated by distributed electrostatic force $\hat{f}^e$. The low light gray dashed line corresponds to the designed initial shape with initial nominal elevation $\hat{h}_0$. The black solid line represents the beam's actual initial shape with an increased elevation $\hat{h}$ developed due to the presence of internal axial compression force $\hat{P}$. The upper gray dashed line corresponds to the deformed configuration caused by the electrostatic load. Positive directions of the beam's deflection and of the loading are shown.

We assume that $\hat{d} \ll L, \hat{h} \ll L$ and that the deflections are small with respect to the beam's length. Under these assumptions, the beam's behavior is described in the framework of the Euler-Bernoulli theory combined with the shallow arch approximation. Note that while the influence of the fringing fields on the electrostatic force acting on the curved beam could be taken into considerations (e.g., [9,10]), we use an electrostatic force presented in the form of a simple parallel capacitor formula for the sake of simplicity.

The dimensional equilibrium equation of the beam (see [7] for the case of electrostatic time independent loading) is as follows (with the axial load P being positive for compression)

$$\rho A\frac{\partial^2\hat{w}}{\partial\hat{t}^2} + \hat{c}\frac{\partial\hat{w}}{\partial\hat{t}} + \tilde{E}I_{yy}\left(\frac{\partial^4\hat{w}}{\partial\hat{x}^4} - \frac{\partial^4\hat{w}_0}{\partial\hat{x}^4}\right) + \left(\hat{P} - \frac{\tilde{E}A}{2L}\int_0^L\left(\left(\frac{\partial\hat{w}}{\partial\hat{x}}\right)^2 - \left(\frac{\partial\hat{w}_0}{\partial\hat{x}}\right)^2\right)d\hat{x}\right)\frac{\partial^2\hat{w}}{\partial\hat{x}^2} + \frac{\varepsilon_0\hat{b}V^2}{2(\hat{g}_0+\hat{w})^2} = 0 \tag{1}$$

with the following initial and boundary conditions

$$\hat{w}(0,\hat{x}) = \hat{w}_p \qquad \frac{\partial\hat{w}}{\partial\hat{t}}(0,\hat{x}) = 0 \tag{2}$$

$$\hat{w}(\hat{t},0)=\hat{w}(\hat{t},L)=0 \qquad \frac{\partial \hat{w}}{\partial \hat{x}}(\hat{t},0)=\frac{\partial \hat{w}}{\partial \hat{x}}(\hat{t},L)=0 \qquad (3)$$

where $\varepsilon_0 = 8.854\times10^{-12}$ F/m is the permittivity of the free space and $V(t)$ is the voltage difference between the beam and the electrode.

Reduced Order Model

In order to analyze the snap-through and pull-in behavior of the beam, a RO model based on the Galerkin decomposition is constructed. The deformed and initial shape of the beam are represented by the series $w(x)\approx\sum_{i=1}^{n} q_i\varphi_i(x)$ and $w_0(x)=\sum_{i=1}^{n} q_{0i}\varphi_i(x)$, where φ_i represent the buckling eigenmodes of a straight double-clamped beam.

The Galerkin procedure converts Eq. (1) in its non dimensional form to a system of coupled nonlinear ordinary differential equations for which the first symmetric term is taken for further analysis. The procedure yields a single non-dimensional nonlinear differential equation in terms of the general coordinates q (for further details see [7] for the static case), namely

$$\ddot{q}+c\dot{q}+\left(\frac{b_{11}}{m_{11}}-\left(P+\alpha s_{11}h_0^2\right)\frac{s_{11}}{m_{11}}\right)q \qquad (4)$$
$$+\frac{\alpha s_{11}^2}{m_{11}}q_1^3-\frac{b_{11}}{m_{11}}h_0+\frac{\beta}{2m_{11}\sqrt{1+q^3}}=0$$

with the following initial conditions

$$q(0)=h \qquad \dot{q}(0)=0 \qquad (5)$$

where $m_{11}=3/8, b_{11}=2\pi^4, s_{11}=\pi^2/2$. Note that since the base functions are those of exact buckling modes of an initially straight beam, we have $b_{11}/s_{11}=4\pi^2\triangleq P_E$ as the lowest non-dimensional buckling load. The non-dimensional quantities used in Eq. (4) are defined in Tab. 1 where the damping parameter is taken, for the sake of simplicity, with an approximation of the first frequency (ω_1) of a double clamped straight beam upon which an axial force exists [11]. The expression of which, is as follows for a case in which $P<P_E$ and for $P>P_E$, separately

$$c=\frac{\omega_1}{Q}P_E\sqrt{\frac{1}{3}(1-P)} \qquad P<P_E \qquad (6)$$

$$c=\frac{\omega_1}{Q}P_E\sqrt{\frac{7}{8}(P-1)} \qquad P>P_E \qquad (7)$$

where Q is the quality factor (Q - factor).

It is to note that although the geometric parameter α is used in the formulation for the sake of generalization, all subsequent figures are given for a rectangular cross section for which $\alpha=6/d^2$.

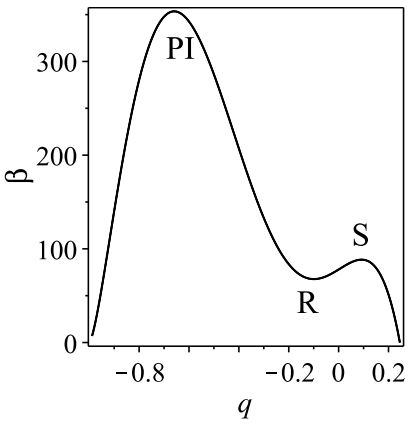


FIGURE 2: Buckling diagram under quasi-static electrostatic load for $d=0.22$, $h_0=0.2$, $P/P_E=0.5$. In this configuration the snap-through voltage is higher than that of the pull-in point. Points P, R and PI designate the snap-through, release and pull-in points, respectively.

In the next sections, we show two different technics to capture the bi-stable beam at its second stable state via different actuation signals depending on the beam's configuration. The first section deals with one step actuation and the later deals with a two step signal applied to beams of a special configuration.

ONE STEP ACTUATION

Bi-stable elctrostaticlly actuated beams can be characterized by the position and actuation voltage of the limit points, namely the snap-through, release and pull-in. For a configuration in which the actuation voltage of the snap-through points is lower than the pull-in, as shown in Fig. 2 it is possible to shift the beam from one stable state to the next using either static actuation [7,9,12,13] or dynamic one step signal actuation [8] defined by the signal

$$\beta(t)=\beta_1 H(t) \qquad (8)$$

where $H(t)$ is the Heaviside function and β_1 is its amplitude. As discussed in [8], the dynamic snap-through voltage is lower than the static one.

However, the dynamic transition from one state to the next is also damping dependent, meaning that the beam's passage to its second stable form will not necessarily occur. Fig. 3 shows the time history and phase plane response of two beams characterized by the buckling diagram given in Fig. 2 and having two slightly different damping values (represented by different

TABLE 1: Non-dimensional quantities.

$x \triangleq \hat{x}/L$	Coordinate
$t \triangleq \hat{t}/\sqrt{\left(\tilde{E}I\right)/\left(\rho AL^4\right)}$	Time
$w \triangleq \hat{w}/g_0$, $w_0 \triangleq \hat{w}_0/g_0$	Elevation/initial elevation
$u \triangleq \hat{u}L/g_0^2$	Axial displacement
$h_0 \triangleq \hat{h}_0/g_0$	Initial midpoint elevation
$h \triangleq \hat{h}/g_0$	Midpoint elevation in post axial load application
$d \triangleq \hat{d}/g_0$	Thickness
$\alpha \triangleq \left(g_0^2 A\right)/\left(2I_{yy}\right)$	Stretching parameter
$P \triangleq \left(\hat{P}L^2\right)/\left(EI_{yy}\right)$	Axial load
$c \triangleq \hat{c}L^2/\sqrt{\rho A\tilde{E}I}$	Damping parameter
$\beta \triangleq \left(\varepsilon_0 \hat{b}V^2L^4\right)/\left(2g_0^3\tilde{E}I_{yy}\right)$	Voltage parameter

Q-factors) under the same actuation voltage. The time history and the phase plane were obtained by numerically solving Eq. (4) using the dsolve function in MAPLE. The solver result was then segregated to form the phase plane of $\dot{q} = f(q)$. From the results it is evident, that a small change in the Q-factor can lead to a different response with the final position of the beam being either the first or the second stable state, depending on the Q-factor. It is noticeable from Figs. 3 that both beams oscillate between the two states before coming to rest at one of the stable configurations, signifying that the damping parameter plays a key role in determining the end result of the beam's time dependent response.

In order to see the full dependence and the effect of the Q-factor on the general response, the simulations were broadened for to the full spectrum of operation voltage given by Fig. 2 and a Q-factor in the range of $1 \leq Q \leq 1000$. The result is given in Fig. 4, which shows the response of the beam depending on the actuation voltage and the Q-factor while discerning between three different responses the beam may exhibit. The first is the one seen in Figs. 3(a)(c) in which the beam returns to its initial state (illustrated as the white region); The second response is seen in Figs. 3(b)(d) in which the beam bounces to its second stable configuration and thus achieves a dynamic snap-though (illustrated as the black region) and the third response is the dynamic pull-in in which the beam collapses upon the electrode (illustrated in grey).

From the given map one is able to discern that the threshold between the dynamic snap-through and the lack of it, is of a fractal nature since any small change to either the actuation voltage or the Q-factor may change the beam's response. In addition, the transition to the pull-in is not fractal. However, as shown in Fig. 4, pull-in will not transpire for lower Q-factors, namely a relatively high damping may eliminate the pull-in instability. Furthermore, the effect of the pull-in is also discernable, since for every Q-factor as the voltage rises, the fractality is decreased and the number of points bound to bring the beam to a dynamic snap-through is increased, making the path more denser as one arrives to the third instability point.

TWO STEP ACTUATION

In general, the transition of a micro beam from one stable configuration to another can be actuated by either static or dynamic means. However, for certain geometric parameters, the second stable configuration is inaccessible under a static loading but is attainable by a specially tailored dynamic signal. Such a case is depicted in Fig. 5, in which the pull-in actuation voltage is lower than the snap-through point, meaning that by static actuation, upon reaching the snap-through point, the beam will "skip" the pull-in point and will overshoot to an eventual collapse upon the electrode.

Furthermore, one step actuation will give the same result, since the signal will either keep the beam in its first stable configuration, or will result in a dynamic snap-through which overshoots the beam to the electrode. However, it is possible to change the signal while the beam is in motion to a much lower voltage in the range of the second stable branch, thus enabling a dynamic trapping of the beam such that it gets to a stable state very close to the electrode but doesn't collapse upon it. For that sake we consider the actuation voltage of the form

$$\beta(t) = \beta_1 H(t) - (\beta_1 - \beta_2) H(t - t_0) \tag{9}$$

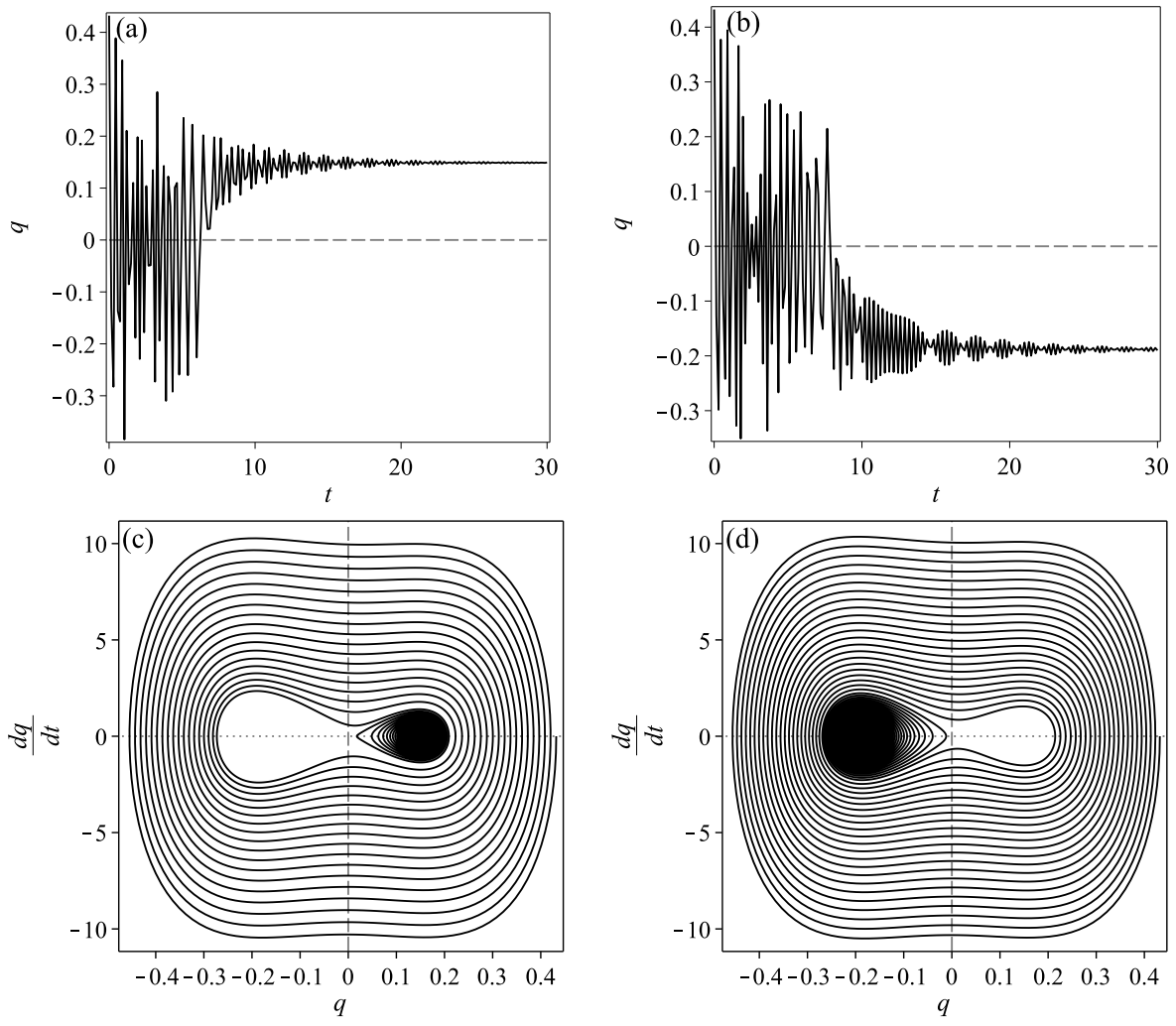


FIGURE 3: Time history response and phase plane pertaining to the configuration presented in Fig. 2 for two different Q-factors under the same suddenly applied step of $\beta = 80H(t)$: (a)(c) $Q = 40$ at which the beam returns to its initial position and (b)(d) $Q = 50$ at which dynamic snap-through is achieved and the beam bounces to its second stable position. The dashed grey and dotted lines represent $q = 0$ and $\dot{q} = 0$, respectively.

where β_1 is the initial first step voltage which is dropped to the subsequent second step voltage $beta_2$ after the time interval t0 so that $\beta_2 < \beta_1$.

Figure 6 illustrates the time history and the phase plane responses of the beam and shows that a signal created in the form of Eq. (9) enables the transition to the second stable state for the discussed configuration, shown in Fig. 5. Both the time history and phase plane responses were created in the same manner as was described for the one step actuation scenario. From the given results it is possible to discern that the response can change depending on a small change in the actuation voltage of the second step as the result given in Fig. 6 pertains to the same actuation voltage of the first step and its duration but differs in the strength of the second step. In addition, a closer inspection of the phase plane reveals that since the signal is comprised of two different steps, the trajectory breaks from its initial trajectory which should have caused its eventual collapse upon the electrode. Ergo that in order to cause the trapping effect in this type of a configuration, it is essential that the beam will "skip" the snap-through instability via dynamic pull-in so that a sudden drop in the electrostatic load will result in the eventual rest of the beam at the second stable region. Meaning that for the transition to be possi-

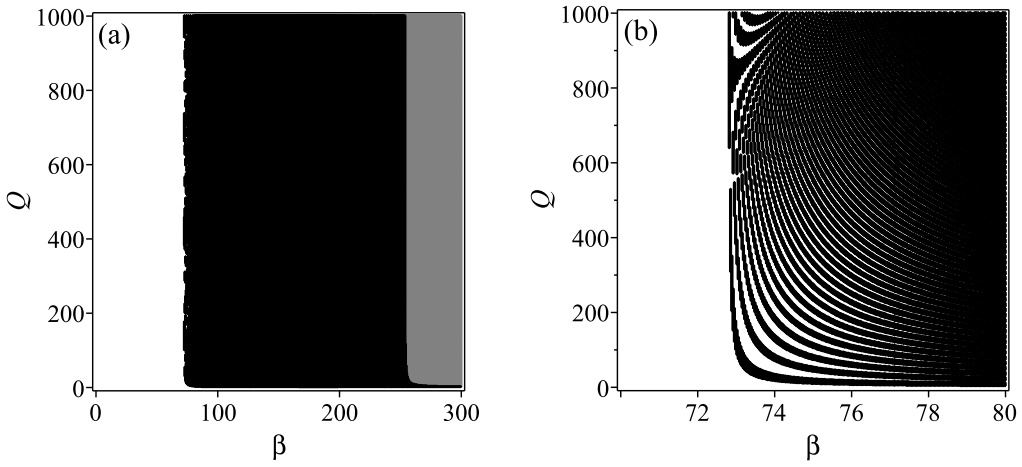


FIGURE 4: Actuation map pertaining to the configuration presented in Fig. 2, illustrating the three regions signifying the three different possible responses depending on the existing damping (Q) and the strength of the signal (β): The white region represents the conditions for which the beam remains at its initial state; The black region represents the conditions for which the beam bounces to its second stable state and the grey region represents the conditions in which the beam experiences pull-in instability. (a) Shows the general map with the three regions while (b) displays the threshold between the white and black region in a close up.

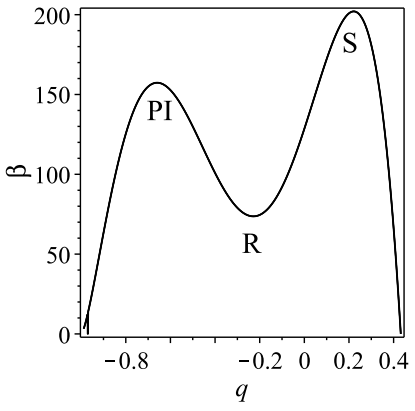


FIGURE 5: Buckling diagram under quasi-static electrostatic load for $d = 0.32, h_0 = 0.33, P/P_E = 0.808$. In this configuration the beam's snap-through voltage point is higher than the pull-in, making the transition to the second stable position impossible via static or one step actuation. Points P, R and PI designate the snap-through, release and pull-in points, respectively.

ble, the first step have to be high enough and short enough in its duration.

In order to see the dependence of all three parameters, namely β_1, β_2 and t_0, a map illustrating the conditions required for a dynamic trapping to transpire was built for a specific Q-factor of $Q = 10$. The simulations were conducted for two different duration times t_0 for a spectrum of actuation voltages of β_1 and β_2 ranging from 0 to the pull-in voltage as defined by its general equilibrium response given in Fig. 5. The outcome of these simulations is given in Fig 7.

From the result of Fig. 7 it is possible to discern that the conditions for dynamic trapping are of a fractal nature as well. It is noticeable that as the second step rises, the tolerance of the first step required for the desired effect is increased and as the first step is higher, the tolerance for the second step is decreased. In addition, it is evident that as the duration time of the first step t_0 is increased, the black region is decreasing in size and as of consequence, the overall tolerance of the parameters is decreased. Furthermore, as the Q-factor rises, the fractality of the region which enables the beam's trapping is decreases as the region becomes more condensed. An observation which is in line with the one step configuration.

SUMMARY AND CONCLUSIONS

In this work, the dynamic behavior of a bi-stable micro beam actuated by a single and a double step dynamic signal for the purpose of achieving a dynamic trapping was investigated. The investigation was comprised of simulations conducted on two different configurations of a bi-stable beam actuated by a an electrostatic load. The two configurations have qualitatively different characteristic quasi-static buckling curves. While for first configuration, the snap-through actuation voltage is lower than the pull-in voltage, for the second configuration the relations are reversed as the snap-through voltage is higher than the pull-in.

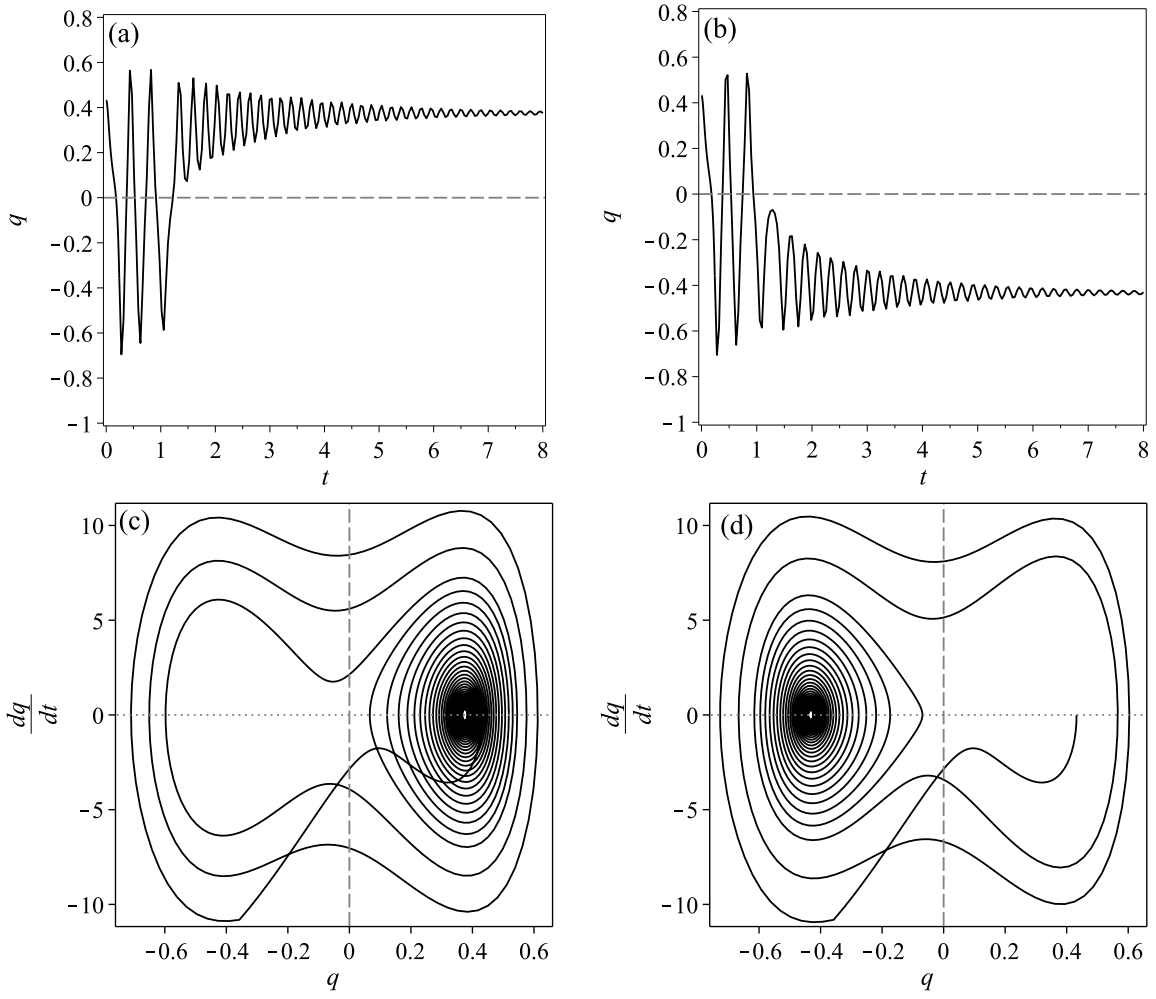


FIGURE 6: Time history response and phase plane pertaining to the configuration presented in Fig. 5 for two loadings which are similar in the first step, namely $\beta_1 = 172$ and $t_0 = 0.24$, but differ slightly in the second step: (a)(c) $\beta_2 = 66$ at which the beam is not able to make the transition to the second stable branch and (b)(d) $\beta_2 = 62$ at which the beam bounces to its second stable position.

For the first configuration, an actuation voltage composed of a single step was simulated for a range of voltages and damping factors characterized by the Q-factor. The results showed that a transition to the second stable state depends not only on the actuation voltage but also on the Q-factor. This dependence is fractal in nature in the threshold to the dynamic snap-through, since a small change introduced to the system via the actuation voltage or the Q-factor could change the results entirely. It is to note, that under electrostatic load, the dynamic snap-through is upper bounded by the dynamic pull-in and the fractal behavior diminishes as the actuation voltage rises.

In the second configuration, where the snap-through actuation voltage is higher then the pull-in, a single step signal would not produce the dynamic trapping effect since such a signal will either leave the beam in its initial state, or will result in an eventual pull-in of the beam. For that reason, a double step signal signal was simulated including a reduction of the first step actuation voltage after a time period t_0 to a lower actuation voltage which is in the range of the quasi-static actuation at second stable branch. A signal not causing a pull-in led to one of two distinguishable scenarios, one being the beam returning to its initial state and the second is the eventual trapping of the beam in the second stable state. This signal has presented a fractal behavior as well, depending on the the actuation voltage of the two steps and the duration time of the first. As the first voltage is higher, the range of the needed second step voltage is decreased and as

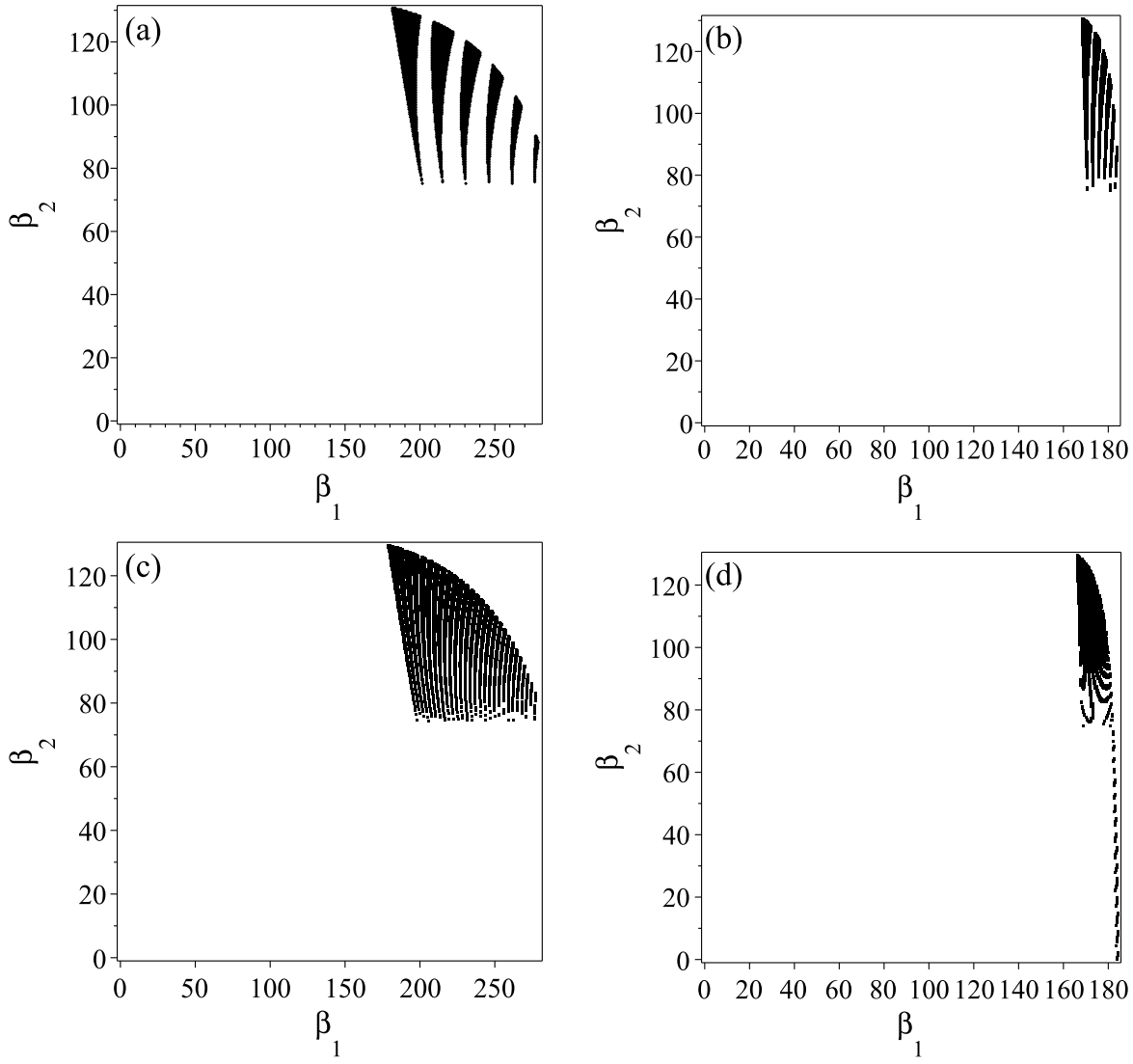


FIGURE 7: Actuation maps pertaining to the configuration presented in Fig. 5, illustrating regions of snap-through responses depending on the the signal's first step (β_1), its duration time (t_0) and second step (β_2), for two different Q-factors. For $Q = 10$ and two different first step durations (a) $t_0 = 0.1$ (b) $t_0 = 0.2$. For $Q = 50$ and two different first step durations (c) $t_0 = 0.1$ (d) $t_0 = 0.2$. The black region represents the conditions for which the beam bounces to its second stable state.

the second step strength is increased, the first step is to be decreased. Moreover, as the duration of the first step gets higher the tolerance of the two steps signal decreases.

In addition, it was observed that in the two trapping scenarios, caused by the single step and double steps signals, the fractality of the region which enables the beam's trapping decreases as a function of the Q-factor.

Acknowledgments

The research is supported by the Broadcom Foundation and Tel Aviv University Authentication Initiative.

REFERENCES

[1] Simitses, G. J., 1989. *Dynamic Stability of Suddenly Loaded Sturctures*. New York Springer-Verlag.

[2] Villagio, P., 1997. *Mathematical Models for Elastic Structures*. Cambridge, Cambridge University Press.
[3] Simitses, G. J., and Hodges, D. H., 2006. *Fundamentals of Structural Stability*. Butterworth-Heinemann.
[4] Chandra, Y., Stanciulescu, I., Virgin, L. N., Eason, T. G., and Spottswood, S. M., 2013. "A numerical investigation of snap-through in a shallow arch-like model". *Journal of Sound and Vibration,* ***332***(10), pp. 2532–2548.
[5] Moghaddasie, B., and Stanciulescu, I., 2013. "Equilibria and stability boundaries of shallow arches under static loading in a thermal environment". *International Journal of Non-Linear Mechanics,* ***51***, pp. 132–144.
[6] Virgin, L., Wiebe, R., Spottswood, S., and Eason, T., 2014. "Sensitivity in the structural behavior of shallow arches". *International Journal of Non-Linear Mechanics,* ***58***, pp. 212–221.
[7] Medina, L., Gilat, R., and Krylov, S., 2014. "Symmetry breaking in an initially curved pre-stressed micro beam loaded by a distributed electrostatic force". *International Journal of Solids and Structures,* ***51***(11), pp. 2047 – 2061.
[8] Krylov, S., and Dick, N., 2010. "Dynamic stability of electrostatically actuated initially curved shallow micro beams". *Continuum Mechanics and Thermodynamics,* ***22***(6-8), pp. 445–468.
[9] Krylov, S., Ilic, B., Schreiber, D., Seretensky, S., and Craighead, H., 2008. "The pull-in behavior of electrostatically actuated bistable microstructures". *Journal of Micromechanics and Microengineering,* ***18***, p. 055026.
[10] Das, K., and Batra, R. C., 2009. "Symmetry breaking, snap-through and pull-in instabilities under dynamic loading of microelectromechanical shallow arches ". *Smart Materials and Structures,* ***18***, November, p. article number 115008.
[11] Joe, D. J., Linzon, Y., Adiga, V. P., Barton, R. A., Kim, M., Ilic, B., Krylov, S., Parpia, J. M., and Craighead, H. G., 2012. "Stress-based resonant volatile gas microsensor operated near the critically buckled state". *Journal of Applied Physics,* ***111***(10), pp. 104517–104517.
[12] Medina, L., Gilat, R., and Krylov, S., 2012. "Symmetry breaking in an initially curved micro beam loaded by a distributed electrostatic force". *International Journal of Solids and Structures,* ***49***(13), pp. 1864 – 1876.
[13] Medina, L., Gilat, R., and Krylov, S., 2012. "Symmetry breaking criteria in electrostatically loaded bistable micro beams". In *Spontaneous Symmetry Breaking, Self-Trapping, and Josephson Oscillations in Nonlinear Systems*, B. Melamed, ed., Progress in Optical Science and Photonics. Springer London, Limited.

DYNAMIC TRAPPING EXPERIMENT IN AN ELCTROSTATICLLY ACTUATED INITIALLY CURVED BEAM

Novelty/ Progress Claims

We report on a first experimental demonstration of dynamic trapping of electrostatically actuated curved micro beam in an isolated post-buckled configuration, which shows enhanced gap usage which cannot be reached under static actuation. This is achieved using a dynamic snap-through induced by a tailored two-steps time dependent electrostatic loading.

Background/ State of the Art

Initially curved micro beams, such as the one shown in Fig. 1, may exhibit two sequential instabilities under quasi-static electrostatic loading. The transition from one stable region to the second is called snap-through. Micro and nano electromechanical systems (NEMS and MEMS) incorporating bistable structural elements show advantages in applications such as switches, sensors and non-volatile memories. Furthermore, bistable electrostatic actuators can use a larger part of the gap before collapsing upon the electrode as opposed to a straight beam [1].

However, it may be possible to attain even higher gap usage. Depending on the beam geometry, the first critical (snap-through) voltage is either lower or higher than the second one (pull-in) creating different possible gap usage percentage, as illustrated in Fig. 2. In the first case, the transition to the post-buckled state can be induced by both quasi-static or dynamic actuation. In the second case however, both loadings can generate only the pull-in collapse. It was shown theoretically in [2] that the second, statically inaccessible stable configuration, distinguished by much larger gap usage and therefore attractive for applications, can be achieved dynamically by application of a specially tailored two-steps signal. Here we present the first experimental demonstration of this phenomenon.

Operational principle

Our device is an initially curved, double clamped micro beam of length $L=1000\ \mu$m, width $b=20\ \mu$m and thickness d with initial central elevation h, located at a distance g_0 from the electrode (Fig. 1). The beam is actuated by a time dependent voltage signal, operating between the beam and the electrode given by

$$V(t) = V_1 H(t) - (V_1 - V_2) H(t - t_1) \qquad (1)$$

where V_1 and V_2 are the first and second step, respectively, H(t) is the Heaviside step function and t_1 is the time duration of the first step.

The time dependent signal can have several effects. To illustrate it, the beam shown in Fig. 2(b) was subjected for three signals with identical V_1 and t_1, while changing V_2, thus yielding the responses presented in Fig. 3(a) with its corresponding signals in Fig. 3(b). While for the lower V_2 the beam vibrates near to its initial state (line 1), under a higher V_2 the beam snaps to its second stable state (line 2) while the highest V_2 yields a pull-in (line 3).

Experimental Results

To show proof of concept, a beam with a geometry corresponding to one presented in Fig. 2(b) was first loaded quasi-statically until it experienced a pull-in collapse. The result is given in Fig. 4, which shows both the buckling curve and a direct visualization of the beam center movement (see [3] for details). Upon obtaining the static characterization of the beam first stable region, the beam was then subjected to the two step dynamic signal. We used fast video recording techniques (Phantom V5.2 camera, 17,000 FPS) to acquire the image. Pictures showing the beam center before and after successful and unsuccessful transitions are shown in Fig. 5. Fig. 5(b) shows that under the suitable actuation signal, which is consisted with the predicted by the model, the beam is halted at a distance of 6.68μm from the electrode crating a gap usage of 78%. A transition to even higher gap usage percentage is possible with implementation of the above signal on beams with higher elevations.

Word Count: 590

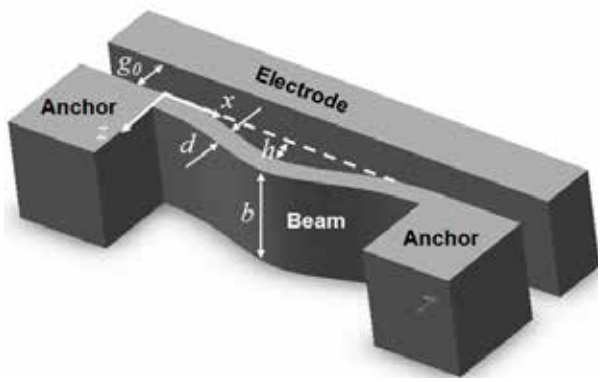


Figure 1: Schematics of an initially curved beam with elevation h, thickness d and width b, actuated by an electrode operating from a distance of g_0.

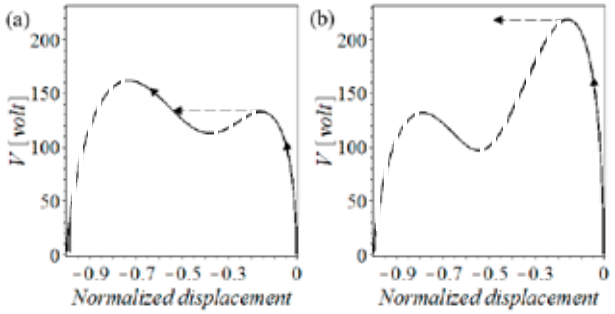


Figure 2: Buckling curves of beams model results: (a) d=3.32μm, h = 3.6 μm and g_0 = 10.58 μm. A configuration allowing bistability with the usage of both static and dynamic loads. (b) d = 3.62μm, h = 5.46μm and g_0 = 10.11μm. A configuration in which the second state cannot be reached with either static or suddenly applied load. Solid and dashed lines represent stable and unstable branches, respectively. The arrows along the curves depict the beam midpoint moment under static loading.

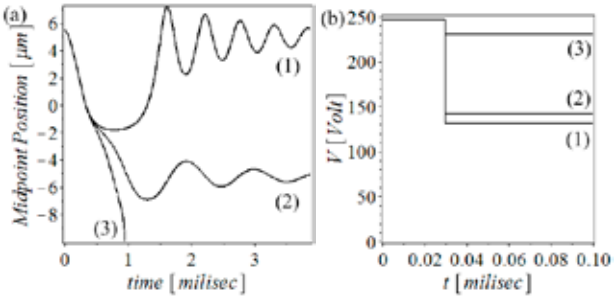


Figure 3: Model result: (a) The different responses to (b) suddenly applied two step signals operating on the configuration from Fig. 2(b) with V_1 =246.71 Volt and t_1 = 0.0003 sec and varying V_2: (1) The beam remains at its initial state with V_2 = 131.11; (2) The beam snap-through to the second stable state with V_2 = 141.89 Volt; (3) Pull-in response for V_2=230.35 Volt.

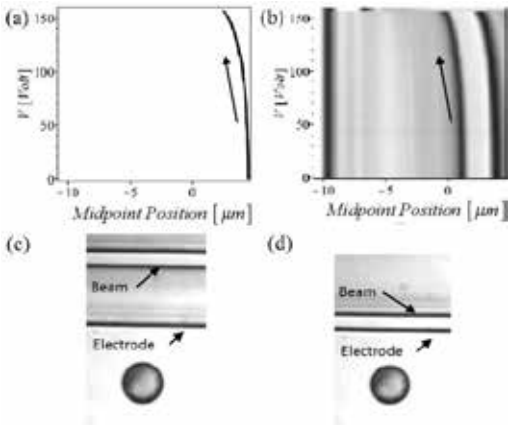


Figure 4: (a) Experimental buckling curve obtained under static loading (b) Direct visualization of the beam movement as a result of the electrostatic load. (c) The beam initial state (d) Configuration after pull-in experienced under quasi-static loading.

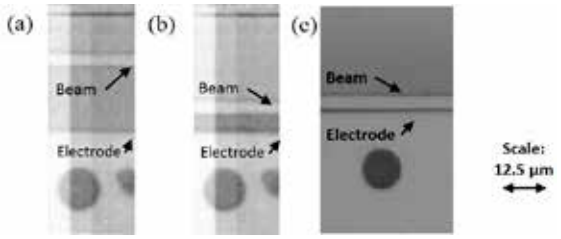


Figure 5: Dynamic experiment: (a) The beam initial elevation before actuation (b) The beam trapped close to the electrode as a result of a snap-through induced by the correct two-step signal sequence of V_1 =220 Volt, V_2 =110 Volt and t_1 = 0.0002 sec; (c) Pull-in response resulting from the incorrect signal sequence V_1 =270 volt, V_2 =140 Volt and t_1 = 0.001 sec.

REFERENCES:

[1] Zhang, W. M., Yan, H., Peng, Z. K., & Meng, G. (2014). Electrostatic pull-in instability in MEMS/NEMS: A review. *Sensors and Actuators A: Physical*, *214*, 187-218.

[2] L. Medina, R. Gilat, S. Krylov, "Dynamic Trapping in Bi-Stable Electrostatically Actuated Curved Micro Beams," Proceedings of the ASME 2014 International Design Engineering Technical Conferences &Computers and Information in Engineering Conference IDETC/CIE E 2014 August 17-20, 2014, Buffalo, New York, USA

[3] Medina, L., Gilat, R., Ilic, B., & Krylov, S. (2014). Experimental investigation of the snap-through buckling of electrostatically actuated initially curved pre-stressed micro beams. *Sensors and Actuators A: Physical*.

Experimental investigation of the snap-through buckling of electrostatically actuated initially curved pre-stressed micro beams

L. Medina[a(*)], R. Gilat[b], and S. Krylov[a]

a - School of Mechanical Engineering, The Iby and Aladar Fleischman Faculty of Engineering, Tel Aviv University, Ramat Aviv 69978, Israel

b - Department of Civil Engineering, Faculty of Engineering, Ariel University, Ariel 44837, Israel

(*) – liormedi@post.tau.ac.il

The experimental study of the symmetric and asymmetric buckling of initially curved micro beams subjected to an axial pre-stressing load and transversal distributed electrostatic force is presented. The devices were fabricated from single crystal silicon on insulator (SOI) wafer using deep reactive ion etching (DRIE). The in-plane quasi-static beam's response was video recorded and analyzed by means of image processing. The influence of geometrical and operational parameters on the beam's response was investigated using the reduced order model. The axial pre-stressing force was assessed on the basis of the deviation of the beam's actual initial curvature from the nominal designed one. The experimental results, shown in Fig. 1, are consistent with the theoretical symmetric and asymmetric snap-through buckling criteria [1].

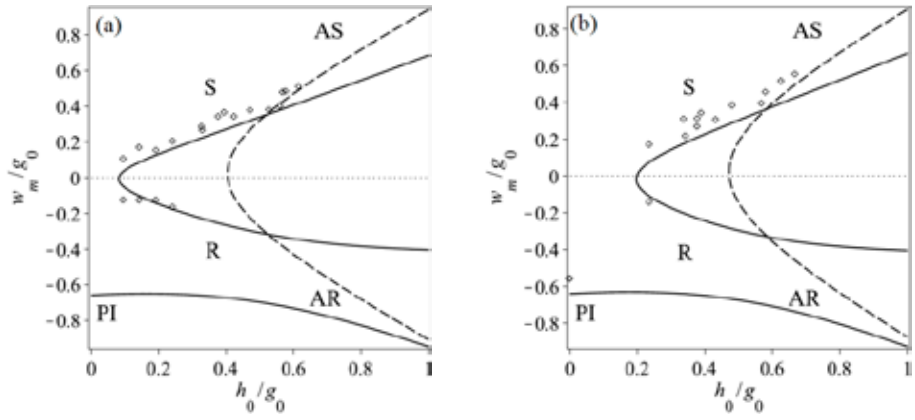


Figure 1. Buckling maps, location of the critical points of the electrostatically loaded beam obtained through the experiments (circles) and the 2 DOF model (lines), for chip # 1 with average thickness to gap ratio $d/g_0 = 0.34$ and an average axial load of (a) $P/P_E = 0.81$, and chip \# 2 with average thickness to gap ratio of $d/g_0 = 0.32$ and an average axial load of (b) $P/P_E = 0.41$. The dashed lines show the theoretically predicted location of the symmetric snap-through (S), symmetric release (R) and pull-in (PI) limit points, and the solid lines correspond to the asymmetric snap-through (AS), and release (AR) bifurcation points.

References:

[1] Medina, L., Gilat, R., Krylov, S., (2014). Symmetry breaking in an initially curved pre-stressed micro beam loaded by a distributed electrostatic force. *Int. J. Solids Structures.*

Bi-stable micro beams under electrostatic load – an overview

מיקרו קורות בעלות שני מצבים יציבים תחת עומס אלקטרוסטאטי - סקירה

L. Medina[1], R. Gilat[2], and S. Krylov[1]
1 -School of Mechanical Engineering, The Iby and Aladar Fleischman Faculty of Engineering, Tel Aviv University, Ramat Aviv 69978, Israel
2 - Department of Civil Engineering, Faculty of Engineering, Ariel University, Ariel 44837, Israel
(*) – liormedi@post.tau.ac.il

ליאור מדינה[a], רבקה גילת[b] וסלבה קרילוב[a]
1 – הפקולטה להנדסה, בית הספר להנדסה מכנית, אוניברסיטת תל-אביב, רמת אביב 69978, ישראל
2 – המחלקה להנדסה אזרחית, הפקולטה להנדסה, אוניברסיטת אריאל, אריאל 44837, ישראל

The buckling of initially curved micro beams subjected to an axial pre-stressing load and transversal distributed electrostatic force is presented. The theoretical analysis is based on a reduced order model resulting from the Galerkin decomposition with buckling modes of a straight beam used as base functions.

In order to investigate the symmetric and asymmetric buckling of the beam under quasi-static electrostatic loading, at lease the two first symmetric and asymmetric modes should be included in the RO model. The 2 DOF model, in conjunction with the arc-length method, provides the beam response and the location of the symmetric limit points as well as the location of the points at which the asymmetric response bifurcates from the symmetric one. The comparison between the results provided by the RO model and those obtained by direct numerical analyses confirms the accuracy of the predicted locations of the critical buckling points while the prediction of corresponding voltages is found to be less accurate. On the basis of this information, the location-based criteria for symmetric limit point buckling and for symmetry breaking are derived in terms of the beams geometric parameters and the pre-stressing axial force. It is shown that while the symmetry breaking conditions are affected by the nonlinearity of the electrostatic force, its influence is less pronounced than in the case of the symmetric snap-through.

In order to study the beam behavior under dynamic electrostatic loads, the RO model equations, including the inertia and damping effects, were integrated using the Runge-Kutta method. Specifically, the behavior of two types of beam configurations was studied, configurations which exhibit bi-stability under both quasi-static and dynamic conditions and configurations for which the second stable branch cannot be reached by quasi static loading. The intensive numerical investigation shows that the beam behavior depends on the damping and that the dynamic snap-through criterion is not conclusive, namely, voltage parameters higher than the snap-through threshold parameter may yield bouncing of the beam back to the primary stable branch. Moreover, it is found that by tailoring the time dependent electrostatic actuation, beams of the second type can be trapped at the second stable branch which is inaccessible under quasi-static conditions.

Prof. David Mendlovic

School of Electrical Engineering
The Iby and Aladar Fleischman Faculty of Engineering

Paper

- **System Design for Light Efficient Multispectral Compressed Light Field Imaging System**

System Design for Light Efficient multispectral Compressed light field imaging system

Schleyen Ran,[1] Mendlovic David,[2]

Department of Electrical Engineering-Systems, Faculty of Engineering, Tel-Aviv University, Ramat Aviv 69978, Israel
**Corresponding author: ranschleyen@mail.tau.ac.il*

Received Month X, XXXX; revised Month X, XXXX; accepted Month X, XXXX; posted Month X, XXXX (Doc. ID XXXXX); published Month X, XXXX

In this paper we present a system intended for high resolution / High light efficiency light field acquisition. The optical design is based on standard digital imaging system with inclusion of a multi-spectral component for compressing angular and spectral information into a 2D CMOS / CCD sensor. The system design is based on previously proven system but we significantly enhanced the performances by 3 elements at the system design. 1) Improved light efficiency by 66% through replacement of the light compressor element with multispectral coder, and avoiding the sensors color filter array. 2) Well-adjusted algorithm for post capture color light-field reconstruction. 3) Optimizing all the system design components by analyzing the system and simulating it.

1. Introduction

Computational photography utilizes the synergy between the imaging systems to an efficient image processing algorithms thus generating a superior imaging system. The joint design of optics & algorithm enhances the mutual capabilities and overcomes limitations of standard digital photography. Good examples are extended Dynamic range by multi-exposure or the Light Field imaging system. Light field (LF) cameras have been gaining popularity by various commercial companies. These cameras provides a unique user experience and photographic capabilities, such as - digital post capture refocus, increased depth of field (DOF), 3D depth estimation and correction for lens aberration such as chroma & astigmatism. These kinds of cameras had limited success till now due to their two main inherent drawbacks: the high resolution penalty alongside additional computational effort. Both gaps are rapidly becoming less important for mass users as the exponential growth in computer power and shrinkage of sensor pixels size continues. State of the art performances in capturing light field imaging on a portable single exposure device were presented by combining the mathematical capabilities of Compressive Sensing theorem with coded aperture camera's configuration [1-3]. The Following work is largely based on the successful results of the above mentioned to restore Light Field resolution.

Background

Light Field imaging [4] much like "integral imaging" of [5] and Plenoptic camera [6] deals with recording of the additional angular information, the same information which is lost on the traditional cameras / 2D projection imaging system. The original methods to capture Light Field imaging consists of multi-exposure shifted camera [7], multi-camera array [8] and micro-lens array [9-13]. The multi-exposure method is slow, mechanically complicated and inapplicable for moving scenes. The micro-lens based methods, while avoiding all previous drawback thus enabling true single snapshot handheld imaging device, trades its resolution for the angular information. To gain some of this resolution back Broxton – 2013 [9] approach was to provide a wave optics module and use it to achieve super resolution through 3D de-convolution. Another approach to acquire light field imaging is through Coded aperture [1, 2, 14, 16] though good results were demonstrated, to acquire full sensors resolution several exposures are required, and much of the light passing through the optics is being attenuated by the mask. In Marwah K Paper from MIT Media lab 2013 [1] aiming to resolve the multi-exposure requirement a compressed signal reconstruction, relatively new mathematical tool, was utilized to reconstruct full resolution light field image using single exposure. These novel impressive results were the starting point to this paper.

Contribution

The Lenslet array method suffers from severe reduction in resolution and the coded aperture method attenuates much of the light for coding the information. Our work deals with improving the system to resolve these two. To achieve this we merged the functionality of the Color filter array (CFA) with the coded aperture thus enabling us to completely remove the CFA. The restoration algorithm had to be adjusted and optimized accordingly. We optimized the relevant parameters and simulated their effect using Matlab Light Field imaging simulator.

Navigation through

Chapter 1 – The *Introduction chapter* deals with a brief overview and related work for Light field imaging techniques. Chapter 2 – provides *basic background / formulation* for light field imaging & compressed sensing. Chapter 3 elaborates about our *design considerations* for optimal system and Simulation results. Chapter 4 is the final *discussion* for this paper.

2. Basics

Light-Field 4D representations

In our work we adopted the 4D Light field representation [7]. In the 4D-LF each ray is represented by an intensity value traveling through 3D space. The 4-Dimensions are 2 sets of axis indexes U,V & R,S defining two 2D planes which the ray crosses. It is easy to understand why this convention is intuitive as we usually deal with two main physical planes, the first would be the lens plane and the latter would be the sensor plane given to us in pixels units.

Image Formation

In Conventional Camera, the image forms as the light power accumulates on the sensor. Each pixel is affected by Light rays from every location of the main iris plane. This can be visualized as seen at Figure 2.

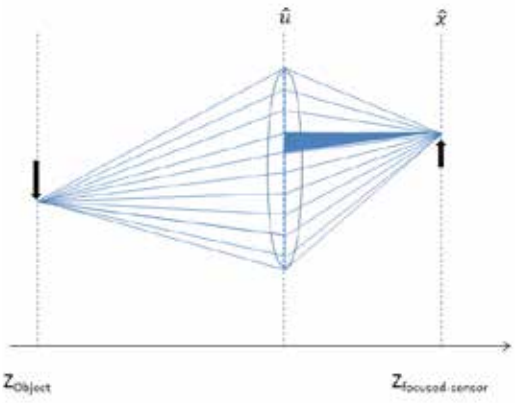


Fig. 1 - 2D simplified image formation visualization.

Light projected from a point of object (on the left) distributed evenly across the lens, then collected from the lens axis "*u*" to the focal plane on the right marked "*x*" axis. This can be formulated as integration of the light field function over specific range of lens area:

$$I(x) = \int L_f(x,u)du \qquad Eq.\ 1$$

"L_f" being the 2D light Field representation.

* The above description is idealized and simplified to a 2D world a slightly more rigorous formulation can be found at [17].

Coded Aperture

There could be many configurations for coded aperture [1, 2], and the reconstruction algorithm, will change accordingly. The Basic idea is that some pre-known spatially varying gain factors are applied to the lights rays as they travel to the sensor.

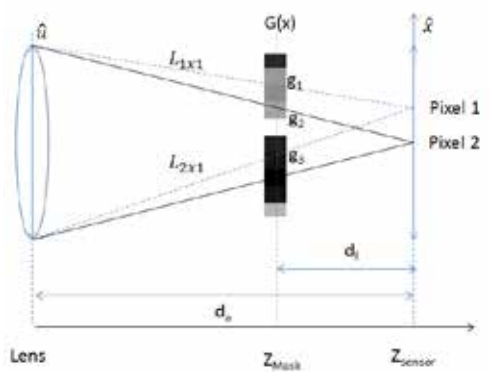


Fig. 2 - Coded aperture placed at the optical path

The drawing shows a mask placed at distance dl from the sensor attenuating each light ray with attenuation factor G(u,x) u being the location within lens axis and x being the mask axis.

Mathematically this can be described as applying gain to Eq. 1 as follows:

$$I(x) = \int_v G(u,x)L_f(x,u)du \qquad Eq.\ 2$$

Let us describe how this configuration preserves angular information. Let L_{nxm} represent a ray of light. Denote n in L_{nxm} as the index position on the lens from which light flows and m the pixel position on the sensor. Let $G(x) \in [0,1]$ be attenuation value at masks pixels location x. expressing this image formation as a discrete matrix formulation:

$$\begin{bmatrix} pix_1 \\ pix_2 \end{bmatrix} = \begin{bmatrix} g_1 & 0 & g_3 & 0 \\ 0 & g_2 & 0 & g_4 \end{bmatrix} \begin{bmatrix} l_{1x1} \\ l_{1x2} \\ l_{2x1} \\ l_{2x2} \end{bmatrix} \qquad Eq.\ 3$$

This equation is an under-determined set of equations which can't be solved. One method for finding a solution is through the usage of additional sensor exposures each with orthogonal mask. We refer to this method as "multi-exposures". Using this method a full sensor resolution can be recovered by simple Matrix inversion, but it is less suitable for "standard"/ handheld camera or moving scene. The same underdetermined set can still be solved if we apply a prior assumption that true image data is usually smooth meaning that light rays coming from the same area on the lens and arriving to proximate location on the sensor have very similar value. This assumption is a bit similar in nature to the more advanced compressive methods which are detailed on the next pages; this simplification merely provided for its intuitive understanding. Now we can rewrite the same equation as follows:

$$\left.\begin{matrix} l_{1x1} \approx l_{2x1} \equiv l_1 \\ l_{1x2} \approx l_{2x2} \equiv l_2 \end{matrix}\right\} \rightarrow \begin{bmatrix} pix_1 \\ pix_2 \end{bmatrix} = \begin{bmatrix} g_1 & g_3 \\ g_2 & g_4 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \qquad Eq.\ 4$$

This set of equations can be solved given that the two sets of gains are not linearly dependent, that is $\det|G| \neq 0$. It is easy to see that this requirement immediately doesn't hold on the two extreme scenarios:

Scenario 1: The mask is located directly over the sensor – the above equations turns into

$$\begin{bmatrix} pix_1 \\ pix_2 \end{bmatrix} = \begin{bmatrix} g_1 & g_1 \\ g_2 & g_2 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \qquad Eq.\ 5$$

Scenario 2: The mask is located directly over the lens – the above equations turns into

$$\begin{bmatrix} pix_1 \\ pix_2 \end{bmatrix} = \begin{bmatrix} g_1 & g_1 \\ g_2 & g_2 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \qquad Eq.\ 6$$

For both cases $|G| = 0$.

General coded Aperture design

Now the general coded aperture imaging device can be described with the following building blocks.

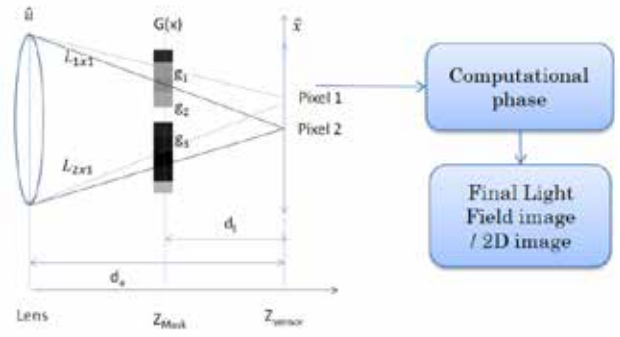


Fig. 3 – Generic coded aperture imaging system

The system comprised of 4 basic elements: lens, Attenuation masked placed at the optical path at distance *dl* from the sensor, sensor and Integrated Signal Processing unit.

Compressed Sensing / Compressed Signals (CS)

In The context of this manuscript the terminology "compressed sensing" deals with Light Field Imaging capabilities and not hyper spectral imaging.

Compressed sensing deals with the reconstruction of a high dimensional signal while sensing only the projection to lower dimension. The element which makes this magic possible is the property of many real life signals that can be well represented using only k sparse elements. Mathematically given some lower dimension signal i : $i = \Phi x$, $i \in \mathbb{R}^m, x \in \mathbb{R}^n$ $\quad m \ll n$ we can recover the signal α

$$i = \Phi D\alpha, \quad \|\alpha\|_0 \leq k \qquad Eq.\ 7$$

Aside from being sparse, a second condition is required for signal recovery. The sensing Matrix which projects x onto *i*, must satisfy some rules. It can be shown that satisfying the Restricted Isometry Property Condition (RIP) [18] will guarantee a good reconstruction. Since verifying that a general matrix satisfies the RIP condition may be of combinatorial computational complexity, In many cases it is enough to use properties of the coherence matrix [18] [19] [20]

$$\mu(A) = \max_{1 \leq i < j \leq n} \frac{|\langle a_i, a_j \rangle|}{\|a_i\|_2 \|a_j\|_2} \qquad Eq.\ 8$$

Minimizing $\mu(A)$, $A = \Phi D$ (D being the reconstruction basis / atoms and ϕ being the sensing matrix) will ensure quality of reconstruction. Additional overview of compressed sensing can be found at [18].

3. Optimizing system performances

Optimizing the Light efficiency & sensor resolution

When approaching the problems of resolution and light efficiency we need to start by analysis of light and resolution loss on previous designs. The introduction of attenuation mask is expensive in terms of light loss (about 50% according to [1], in other papers the penalty was even higher and reached as high as 95% in [14]). The light loss is somewhat mitigated with the fact that wider apertures can be used with higher DOF and less lens aberration due to the aberration restoration capabilities of light field images. Unfortunately without designing a mask which only affects the phase of the light (such work was done and patented at [21]) the light loss at the mask is un-avoidable. Additional light loss takes place on the Sensor itself by the color filter Array (CFA). The CFA – usually implemented in Bayer configuration where the red and blue are only sampled over 25% of the pixels and the green is only sampled by 50% of the pixels (see Figure 5). In Order to comply with Nyquist sampling theorem the maximal spatial resolution arriving at the sensor must be blurred optically to avoid anti-aliasing. This is achieved by insertion of additional Anti-aliasing filter to the optical path.

(*) The Nyquist sampling theorem requires that, the maximal spatial frequency must be sampled with 2 pixels per axis.

The same color filter also reduces the total amount of light arriving to the sensor by more than 60%. This again seems unavoidable as the demand to color cameras is incredible higher than the demand for "black & white" monochromatic cameras.

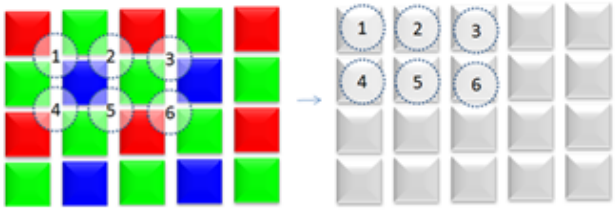


Fig. 4 - : Bayer CFA vs. monochromatic CFA

Fortunately the two problems of CFA and mask light absorption can be merged into a single multispectral mask design and the CFA itself can be removed. This Novel design will lead to improvement of the light efficiency – reverting light absorption to similar normal camera while at the same time increasing the sampling of the sensor density per color. The antialiasing induced blur can also be avoided as we can now comply with Nyquist when using the original sensors resolution.

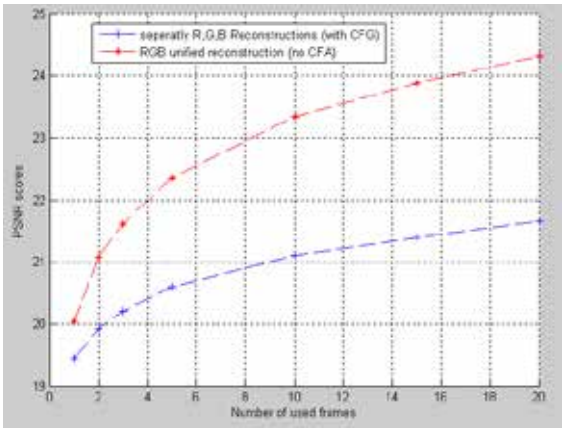


Fig. 5 - Simulation results for our configuration vs. Bayer based

The above simulation shows the increase in reconstructed image PSNR vs. number of used exposures in each method. The results ignore the light efficiency effect and only counts for the sensors improved sampling. The significant improvement in reconstruction quality is visible from the first reconstructed frame and grows. In fact these results are quite to be expected as the induced blur by the anti-aliasing limits the optimal performance of Bayer based camera.

Optimizing mask design

The first step of designing the new camera is to analyze the different properties required of the coded aperture mask. Such are the required contrast, resolution, pattern and the optimal physical location of the mask.

Optimizing the mask placement

A good intuition for this optimization can be drawn out of Eq. 4 & Fig. 2 In order to recover different rays we need to make them pass through different attenuation power of the mask. Let us explore this requirement a little.

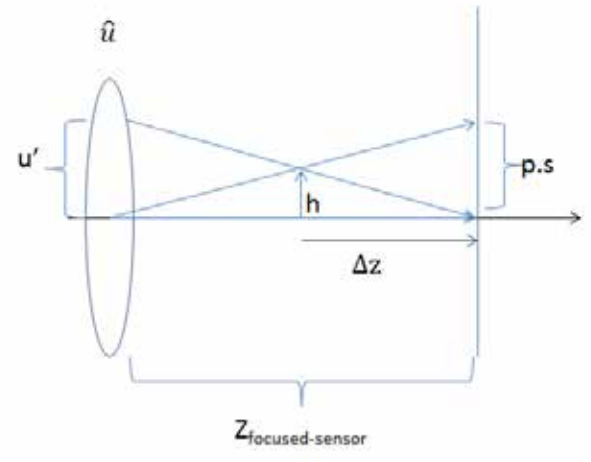


Fig. 6 - Light rays partitioning by the mask

This drawing shows 3 light rays. Two light rays are emerging from the center of the lens and traveling to adjacent pixels. The last light ray emerges from off-axis location on the lens with height u' and travels to the central pixel. For the two rays emerging from the same location we would like to increase the contrast on the mask which they cross. Let's denote the distance on the mask of the two rays by h_1.

$$h_1 = ps.\frac{(Z_{fs} - \Delta z)}{Z_{fs}} \qquad Eq.\ 9$$

Z_{fs} is the equivalent distance of the sensor from the lens, ps is the pixel size.

We also need to increase the contrast on the mask for two rays traveling from different location on the lens to the same pixel. Let us denote by h_2 this distance.

$$h_2 = u' - u'.\frac{(Z_{fs} - \Delta z)}{Z_{fs}} = u'\frac{\Delta z}{Z_{fs}} \qquad Eq.\ 10$$

Both linear with Δz balanced maximization for both mean h1=h2.

$$\Delta z = \frac{ps}{u' + ps} Z_{fs} \qquad Eq.\ 11$$

And since the pixel size is extremely smaller than required angular resolution (lens diameter divided by number of required angles). We can approximate the above equation as follows:

$$\Delta z \approx \frac{ps}{u'} \cdot Z_{fs} = \frac{ps}{D} N \cdot Z_{fs} \qquad Eq.\ 12$$

N being the number of viewed angles per axis, D is the lens Diameter. By applying the imaging equation

$$\frac{1}{Z_{obj}} + \frac{1}{Z_{focused}} = \frac{1}{f} \qquad Eq.\ 13$$

We can also write this in terms of focal length

$$\Delta z \approx \frac{ps}{D} N \frac{Z_{obj} \cdot f}{Z_{obj} - f} \qquad Eq.\ 14$$

In terms of magnitude it means the compressive element should be places a few microns away from the sensor.

Optimizing the mask resolution

The above equation predicts where the mask resolution would be most effective to maximize contrast of different rays. The same geometry also leads to estimation of what would be a good resolution on the mask plane. By combining equations: Eq. 10 & Eq. 14 we get

$$h = u'\frac{\Delta z}{Z_{fs}} \approx \frac{D}{N}\frac{\frac{ps}{D} N \cdot Z_{fs}}{Z_{fs}} = \frac{ps \cdot Z_{fs}}{Z_{fs}} \rightarrow h \approx ps \qquad Eq.\ 15$$

To Test the above result we used the simulator to generate masks with 2D resolution as a factor of the simulated sensors resolution (0.5 being that each pixel in the mask is twice as big as

a pixel in the sensor). The masks were placed close to the sensor according to equation Eq. 14.

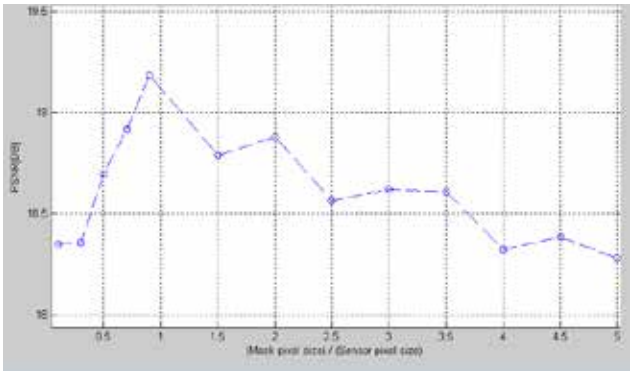


Fig. 7 - Simulation results of mask resolution effect

The plot shows that the optimal reconstruction quality is indeed reached around pixel size = mask pixel size. After this optimal ratio the reconstruction starts dropping, we assume this is due to the pixel size being so small that the equivalent mask value per light ray is no longer binary but rather a convolution of many binary values with Gaussian which yield less contrast and less robustness to noises.

Optimizing the mask dynamic range

As discussed in [18] it seems that scaling the sensing matrix should increase the immunity to noise when reconstructing a signal. The intuition of this is that the Contrast of the mask vs. the values calibrated acts as SNR or Contrast to noise ratio (CNR). Scaling the mask increases the Contrast of the sensing matrix while having no effect over the noise. That being said – we are still limited by maximal gain value $\in [0,1]$ =1 (full transmittance) ,0 being completely opaque. This means best noise immunity should be a binary mask per color spectrum. In order to test this notion we adjusted our simulator to scan trough different values of bits depth in the mask design. Additionally we applied some Gaussian additive noise to the mask and sensor and analyzed in terms of PSNR the reconstruction quality.

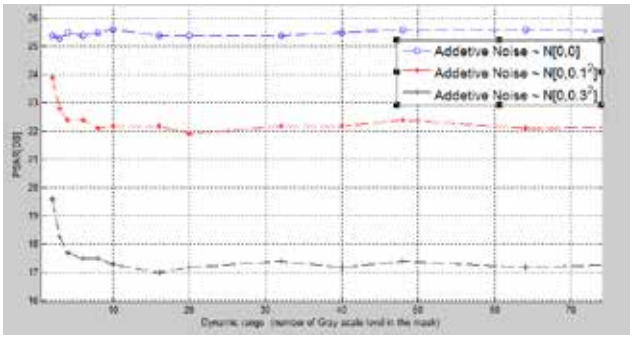


Fig. 8 - : Mask dynamic range vs. reconstruction quality

The above plot shows PSNR value with ranging masks dynamic range (number of unique gain values) with 3 additives noise amplitudes (High PSNR score = better reconstruction). The first plot shows very high PSNR regardless of mask dynamic range – this is the case of random mask without any noise. The second and third demonstrate the drop in reconstruction quality as a function of used Dynamic Range. In all scenarios the best reconstruction is produced when using binary mask. The PSNR drops quickly and stabilizes when using more than 10 different gain values in the mask design.

Optimizing the masks pattern

The pattern selected should optimize the Matrix Coherence Criteria (Eq. 8) This could be achieved using Matlab's "fmincon" or "fminsearch". The constraint of having a binary mask also reduces the number of possible solutions we need to scan through. Another constraint which needs to be taken into account is the light efficiency. The two requirements can be described as a mutual minimization problem balanced by some parameter λ.

$$e = \mu(\phi D) + \lambda T(\phi D) \qquad Eq.\ 16$$

To simplify the Numerical solution process we can utilize the result obtained earlier values can binary. Another simplification that can be applied is to solve for small mask area but forcing cyclic solution to enable the scaling of this mask for any mask size we desire. This kind of simplification is valid since the solution calculation is done on small independent patches. The optimization of such mask should minimize the error value for the "worse" patch within the mask.

The Reconstruction algorithm

Modification to the algorithm to support our configuration

The restoration of the image using a camera with monochromatic attenuation mask and sensor with CFA requires interpolating the sensors reading into Red, Green, Blue full resolution, Then solving three Compressive sensing separate problems. In our modification of the system design the sensing matrix has to be unified for all 3 spectrums (could be generalized to more colors if needed). As follows:

$$\left.\begin{array}{l} I^R = \Phi^R L^R \\ I^G = \Phi^G L^G \\ I^B = \Phi^B L^B \end{array}\right\} \rightarrow \begin{array}{l} L^{RGB} = [L^R, L^G, L^B]^T \\ \Phi^{RGB} = [\Phi^R, \Phi^G, \Phi^B] \end{array} \qquad Eq.\ 17$$

The new restoration problem is then formulated as:

$$I^{Gray} = \phi^{RGB}(L^{RGB})^T \qquad Eq.\ 18$$

The reconstruction algorithm

In the supplementary materials Marwah & Gordon [1] provided both the analyses of which CS algorithm performed well along with the code source. We saw no reason to re-do this exact work – but merely to select optimal parameters for the reconstruction. The code minimizes the following energy function:

$$\min_{\alpha} \frac{1}{2}\|i - \Phi D\alpha\|_2^2 + \lambda\|\alpha\|_1 \qquad Eq.\ 19$$

The only parameter we need to select is the Sparsity Lagrange weight-λ. As mentioned in [18] –Page 41, under some choice of λ this optimization problem will yield the same result as solving

$$\hat{x} = \underset{z}{argmin}\|z\|_1 \qquad Eq.\ 20$$
$$subject\ to\ \|Az - y\|_2 \leq \epsilon$$

If we were to solve this equation we could benefit from selecting our parameter ϵ according to the noise estimation. However we chose to solve Eq. 19 And the λ parameter is unknown a priori. The intuition is that using very small value will yield minimization of the L2 norm (which is equivalent to

maximizing the PSNR) and any high value will enforce sparser solution. We selected λ which aims to minimize k while maintaining the high level of PSNR for the reconstruction. Much like the de-noising problem [22] the joint compressive channel solution outperforms the trivial plane-to-plane reconstruction results.

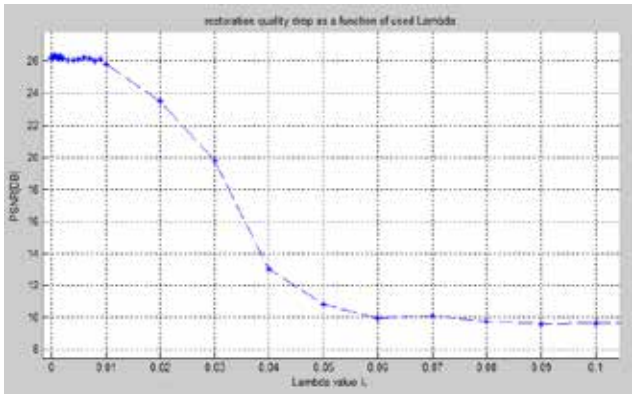


Fig. 9 - : PSNR of the reconstruction vs Lambda parameter

The reconstruction basis

The general problem of learning dictionary / overcomplete dictionary **D** that sparsify a set of signal can be formulated as follows:

$$\underset{\{D,A\}}{argmin}\|T - DA\|_F \ , \quad subject\ to\ \forall_i, \|A_i\|_0 \le k \qquad Eq.\ 21$$

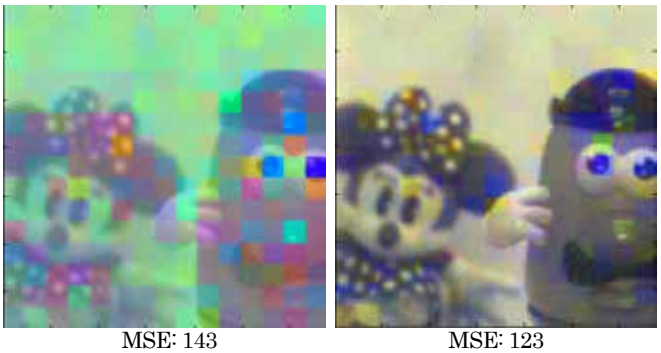
Here $\boldsymbol{T} \in R^{nxd}$ is the training set, D is the required basis & A represents the coefficients corresponding to the training set. The Frobenius norm is the square root of Sum of squared elements in that matrix. There is much literature about Finding a solution for this problem some examples can be found at [19] [20] [22] [23] [24]. Based on Recent trends in Information Theory [25] we used true light fields atoms to construct our dictionary with over-complete basis. We selected over-complete value of 20% (=1.2) to balance the size and computational effort with the image quality (According to Marwah & Gordon [1] the over- completeness factor should range $\in$ [1,2]) These atoms can be generated by the camera itself assuming there is access to the exit pupil. Another method is by collecting images from commercial light field camera (such as Lythro) or by using simulated images. For our purpose we used combination of Light field images generated by our simulator along with images published by Gordon Wetzstein [1] and images taken from our camera. To generate the basis we used modified K-SVD algorithm which was extended to color images as described in [22]. The original K-SVD is a two stage process to solve Equ. 23 [22]. Given an estimate of D we fix D and estimate the coefficient vector for each column i of A. In the next step we fix A, and Update each column of D based on the remaining error to all the training sets which identified by **A** to be corresponding with that vector. So for vector "j" from **D** we Calculate the residual error of Each set in the Training Data set: $E = T_j - \widehat{D}\widehat{A}_j$. Then we use the SVD decomposition to find the strongest Eigen value and replace the basis vector with that one. The color Modification presented By *Julian Mairal* [22] address the problem of color coding at the dictionary. The modified the OMP algorithm aims to maintain bias color of the reconstruction relative to the original patch. This is achieved by replacement of the inner product - error function being minimized by the OMP algorithm.

$$< y, x >_\gamma = y^T \left(I + \frac{\gamma}{n}K\right)x \qquad Eq.\ 22$$

$$K = \begin{pmatrix} J_n & 0 & 0 \\ 0 & J_n & 0 \\ 0 & 0 & J_n \end{pmatrix}, \ J_n(i,j) = 1 : \forall i,j \in n$$

This Dictionary modification was proposed to be better suited for RGB based reconstructions. We implemented it into Color light field imaging and tested our recorded light field and mask.

Reconstruction separated R,G,B | Sparse Color Un-masking



MSE: 143 | MSE: 123

Fig. 10 - : RGB reconstructions quality comparison

4. Conclusions

Applications which utilizes a 3D reconstruction

Thinking ahead on light field efficient camera and its capabilities we can imagine applications which utilizes this embedded 3d capability, Such as robust face recognition which will improve the security of smartphones. Another application may be motion and gestures recognition with higher accuracy through the inferred 3D data compressed in the image.

Results

Recently sensing the field became a significant feature expected from imaging devices. Approaches like dual aperture or range measurement device are complex and expensive. Light field photography might be a reasonable alternative. In this manuscript we described an efficient approach that provides light field photography without the CFA attenuation and good spatial resolution. We analyzed the various parameters of the system, such as mask location, dynamic range, resolution, contrast, and optimized each. We expect that LF camera designed accordingly to significantly outperform the already proven system of CS-LF imaging system recovering Color-LF image at true sensors resolution. As a result enormous number of applications are expected based on this novel approach.

Future work

Experimental demonstration for this design on a compact imaging device is needed. Furthermore, to improve the reconstruction quality we believe that more algorithmic work can be done. When using sliding window reconstruction, each pixel is recovered ~ N2 times (81 in our case). It was shown in [1] that using the average filter produces better estimation for pixel value

and median filter showed even better results. This suggests the error function is roughly zero biased while having significant outliers (filtered by the median). Employing a more advanced filter pixel based reconstruction methods such as "RANSAC" / "Bilateral" should be able to reduce these kinds of noises even more. * The computational effort might be significant, but with dedicated implementation of Hardware this could be reduced to reasonable effort.

Acknowledgements

This research was supported by the Broadcom Foundation and Tel Aviv University Authentication Initiative. We would like to thank the reviewers for their valuable feedback and the following people who made this research possible: Mr. Ariel Raz, Mr. lev Mulokandov, Mr Tal Remez and Mr. Roee Litman.

5. Bibliography

[1] MARWAH, Kshitij, et al. Compressive light field photography using overcomplete dictionaries and optimized projections. ACM Transactions on Graphics (TOG), 2013, 32.4: 46.

[2] VEERARAGHAVAN, Ashok, et al. Dappled photography: Mask enhanced cameras for heterodyned light fields and coded aperture refocusing. ACM Transactions on Graphics, 2007, 26.3: 69.

[3] XU, Zhimin; KE, Jun; LAM, Edmund Y. High-resolution lightfield photography using two masks. Optics express, 2012, 20.10: 10971-10983.

[4] Gershun, A., "The Light Field," Moscow, 1936. Translated by P. Moon and G. Timoshenko in Journal of Mathematics and Physics, Vol. XVIII, MIT, 1939, pp. 51-151.

[5] LIPPMANN, G. La Photographie Int´egrale. Academie des Sciences 1908, 146, 446–451.

[6] ADELSON, Edward H.; WANG, John Y.. A.. . Single lens stereo with a plenoptic camera. IEEE transactions on pattern analysis and machine intelligence, 1992, 14.2: 99-106.

[7] LEVOY, Marc; HANRAHAN, Pat. Light field rendering. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM, 1996. p. 31-42.

[8] WILBURN, Bennett. *High performance imaging using arrays of inexpensive cameras*. 2004. PhD Thesis. Stanford University.

[9] BROXTON, Michael, et al. Wave optics theory and 3-d deconvolution for the light field microscope. Optics express, 2013, 21.21: 25418-25439.

[10] BISHOP, Tom E.; ZANETTI, Sara; FAVARO, Paolo. Light field superresolution. In: Computational Photography (ICCP), 2009 IEEE International Conference on. IEEE, 2009. p. 1-9.

[11] NG, Ren, et al. Light field photography with a hand-held plenoptic camera. Computer Science Technical Report CSTR, 2005, 2.11.

[12] NG, Ren. Digital light field photography. 2006. PhD Thesis. stanford university.

[13] LEVOY, Marc, et al. Light field microscopy. In: *ACM Transactions on Graphics (TOG)*. ACM, 2006. p. 924-934.

[14] XU, Zhimin; LAM, Edmund Y. A high-resolution lightfield camera with dual-mask design. In: SPIE Optical Engineering+ Applications. International Society for Optics and Photonics, 2012. p. 85000U-85000U-11.

[15] LIANG, Chia-Kai, et al. Programmable aperture photography: multiplexed light field acquisition. In: ACM Transactions on Graphics (TOG). ACM, 2008. p. 55.

[16] BANDO, Yosuke; CHEN, Bing-Yu; NISHITA, Tomoyuki. Extracting depth and matte using a color-filtered aperture. In: ACM Transactions on Graphics (TOG). ACM, 2008. p. 134.

[17] Stroebel, L., J. Compton, I. Current, and R. Zakia. "Photographic Materials and Processes" PSA Journal, August, 1986, Vol.52, p.12(2).

[18] DAVENPORT, Mark A., et al. Introduction to compressed sensing. *Preprint*, 2011, 93.

[19] ELAD, Michael. Optimized projections for compressed sensing. *Signal Processing, IEEE Transactions on*, 2007, 55.12: 5695-5702.

[20] DUARTE-CARVAJALINO, Julio Martin; SAPIRO, Guillermo. Learning to sense sparse signals: Simultaneous sensing matrix and sparsifying dictionary optimization. *Image Processing, IEEE Transactions on*, 2009, 18.7: 1395-1408.

[21] GOLUB, Michael, et al. Snapshot spectral imaging based on digital cameras. U.S. Patent Application 13/752,560, 2013.

[22] MAIRAL, Julien; ELAD, Michael; SAPIRO, Guillermo. Sparse representation for color image restoration. Image Processing, IEEE Transactions on, 2008, 17.1: 53-69.

[23] ELAD, Michael; AHARON, Michal. Image denoising via learned dictionaries and sparse representation. In: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. IEEE, 2006. p. 895-900.

[24] ELAD, Michael; AHARON, Michal. Image denoising via sparse and redundant representations over learned dictionaries. *Image Processing, IEEE Transactions on*, 2006, 15.12: 3736-3745.

[25] CANDES, Emmanuel J., et al. Compressed sensing with coherent and redundant dictionaries. *Applied and Computational Harmonic Analysis*, 2011, 31.1: 59-73.

Dr. Noam Rinetzky

The Blavatnik School of Computer Science
Raymond and Beverly Sackler Faculty of Exact Sciences

Technical Report

- **Safety of Live Transactions in Transactional Memory: TMS is Necessary and Sufficient**
- **Brief Announcement: Concurrency-Aware Linearizability**

Safety of Live Transactions in Transactional Memory: TMS is Necessary and Sufficient

Technical Report - Tel Aviv University - School of Computer Science

Hagit Attiya[1], Alexey Gotsman[2], Sandeep Hans[1], and Noam Rinetzky[3]

[1] Technion - Israel Institute of Technology, Israel
[2] IMDEA Software Institute, Spain
[3] The Blavatnik School of Computer Science
Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University, Israel

Abstract. One of the main challenges in stating the correctness of transactional memory (TM) systems is the need to provide guarantees on the system state observed by *live* transactions, i.e., those that have not yet committed or aborted. A TM correctness condition should be weak enough to allow flexibility in implementation, yet strong enough to disallow undesirable TM behavior, which can lead to run-time errors in live transactions. The latter feature is formalized by *observational refinement* between TM implementations, stating that properties of a program using a concrete TM implementation can be established by analyzing its behavior with an abstract TM, serving as a specification of the concrete one.
We show that a variant of *transactional memory specification* (*TMS*), a TM correctness condition, is equivalent to observational refinement for the common programming model in which local variables are rolled back upon a transaction abort and, hence, is the weakest acceptable condition for this case. This is challenging due to the nontrivial formulation of TMS, which allows different aborted and live transactions to have different views of the system state. Our proof reveals some natural, but subtle, assumptions on the TM required for the equivalence result.

1 Introduction

Transactional memory (TM) eases the task of writing concurrent applications by letting the programmer designate certain code blocks as *atomic*. TM allows developing a program and reasoning about its correctness as if each atomic block executes as a *transaction*—in one step and without interleaving with others—even though in reality the blocks can be executed concurrently. Figure 1 shows how atomic blocks are used to manipulate several shared *transactional objects* X, Y and Z, access to which is mediated by the TM.

```
result := abort;
while (result == abort) {
    result := atomic {
        x := X.read();
        y := Y.read();
        z := 42 / (x - y);
        Z.write(z); } }
```

Fig. 1. TM usage

The common approach to stating TM correctness is through a *consistency condition* that restricts the possible TM executions. The main subtlety of formulating such a condition is the need to provide guarantees on the state of transactional objects observed by *live* transactions, i.e., those that have not yet committed or aborted. Because live transactions can always be aborted, one might think it unnecessary to provide any guarantees for them, as done by common database consistency conditions [1]. However, in

the setting of transactional memory, this is often unsatisfactory. For example, in Figure 1 the programmer may rely on the fact that $\mathsf{X} \neq \mathsf{Y}$, and, correspondingly, make sure that every committing transaction preserves this invariant. If we allow the transaction to read values of X and Y violating the invariant (counting on it to abort later, due to inconsistency), this will lead to the program *faulting* due to a division by zero.

The question of which TM consistency condition to use is far from settled, with several candidates having been proposed [2–5]. An ideal condition should be weak enough to allow flexibility in TM implementations, yet strong enough to satisfy the intuitive expectations of the programmer and, in particular, to disallow undesirable behaviors such as the one described above. *Observational refinement* [6, 7] allows formalizing the programmer's expectations and thereby evaluating consistency conditions systematically. Consider two TM implementations—a *concrete* one, such as an efficient TM, and an *abstract* one, such as a TM executing every atomic block atomically. Informally, the concrete TM *observationally refines* the abstract one for a given programming language if every behavior a user can observe of any program P in this language using the concrete TM can also be observed when P uses the abstract TM instead. This allows the programmer to reason about the behavior of P (e.g., the preservation of the invariant $\mathsf{X} \neq \mathsf{Y}$) using the expected intuitive semantics formalized by the abstract TM; the observational refinement relation implies that the conclusions (e.g., the safety of the division in Figure 1) will carry over to the case when P uses the concrete TM.

In prior work [8] we showed that a variant of the *opacity* condition [2] is equivalent to observational refinement for a particular programming language and, hence, is the weakest acceptable consistency condition for this language. Roughly speaking, a concrete TM implementation is in the opacity relation with an abstract one if for any sequence of interactions with the concrete TM, dubbed a *history*, there exists a history of the abstract TM where: (i) the actions of every separate thread are the same as in the original history; and (ii) the order of non-overlapping transactions present in the original history is preserved. However, our result considered a programming language in which local variables modified by a transaction are not rolled back upon an abort. Although this assumption holds in some situations (e.g., Scala STM [9]), it is non-standard and most TM systems do not satisfy it. In this paper, we consider a variant of *transactional memory specification* (*TMS*) [5], a condition weaker than opacity,[4] and show that, under some natural assumptions on the TM, it is equivalent to observational refinement for a programming language in which local variables do get rolled back upon an abort.

This result is not just a straightforward adjustment of the one about opacity to a more realistic setting: TMS weakens opacity in a nontrivial way, which makes reasoning about its relationship with observational refinement much more intricate. In more detail, the key feature of opacity is that the behavior of *all* transactions in a history of the concrete TM, including aborted and live ones, has to be justified by a single history of the abstract TM. TMS relaxes this requirement by requiring only committed transactions in the concrete history to be justified by a single abstract one obeying (i)–(ii) above; every response obtained from the TM in an aborted or live transaction may be justified by a separate abstract history. The constraints on the choice of the abstract history are subtle: on one hand, somewhat counter-intuitively, TMS allows it to include transactions that aborted in the concrete history, with their status changed to committed,

[4] The condition we present here is actually called TMS1 in [5, 10]. These papers also propose another condition, TMS2, but it is stronger than opacity [10] and therefore not considered here.

and exclude some that committed; on the other hand, this is subject to certain carefully chosen constraints. The flexibility in the choice of the abstract history is meant to allow the concrete TM implementation to perform as many optimizations as possible. However, it is not straightforward to establish that this flexibility does not invalidate observational refinement (and hence, the informal guarantees that programmers expect from a TM) or that the TMS definition cannot be weakened further.

Our results ensure that this is indeed the case. Informally, if local variables are not rolled back when transactions abort, threads can communicate to each other the observations they make inside aborted transactions about the state of transactional objects. This requires the TM to provide a consistent view of this state across all transactions, as formalized by the use of a single abstract history in opacity. However, if local variables are rolled back upon an abort, no information can leak out of an uncommitted transaction, possibly apart from the fact that the code in the transaction has faulted, stopping the computation. To get observational refinement in this case we only need to make sure that a fault in the transaction occurring with the concrete TM could be reproduced with the abstract one. For this it is sufficient to require that the state of transactional objects seen by every live transaction can be justified by some abstract history; different transactions can be justified by different histories.

Technically, we prove that TMS is sufficient for observational refinement by establishing a nontrivial property of the set of computations of a program, showing that a live transaction cannot notice the changes in the committed/aborted status of transactions concurrent with it that are allowed by TMS (Lemma 1, Section 6.1). Proving that TMS is necessary for observational refinement is challenging as well, as this requires us to devise multiple programs that can observe whether the subtle constraints governing the change of transaction status in TMS are fulfilled by the TM. We have identified several closure properties on the set of histories produced by the abstract TM required for these results to hold. Although intuitive, these properties are not necessarily provided by an arbitrary TM, and our results demonstrate their importance.

To concentrate on the core goal of this paper, the programming language we consider does not allow explicit transaction aborts or transaction nesting and assumes a static separation of transactional and non-transactional shared memory. Extending our development to lift these restrictions is an interesting avenue for future work. Also, due to space constraints, we defer some of the proofs to [11, Appendix D].

2 Programming Language Syntax

We consider a language where a program $P = C_1 \parallel \cdots \parallel C_m$ is a parallel composition of ***threads*** C_t, $t \in \mathsf{ThreadID} = \{1, \ldots, m\}$. Every thread $t \in \mathsf{ThreadID}$ has a set of ***local variables*** $\mathsf{LVar}_t = \{x, y, \ldots\}$ and threads share a set of ***global variables*** $\mathsf{GVar} = \{g, \ldots\}$, all of type integer. We let $\mathsf{Var} = \mathsf{GVar} \uplus \biguplus_{t=1}^{m} \mathsf{LVar}_t$ be the set of all program variables. Threads can also access a transactional memory, which manages a fixed collection of ***transactional objects*** $\mathsf{Obj} = \{o, \ldots\}$, each with a set of ***methods*** that threads can call. For simplicity, we assume that each method takes one integer parameter and returns an integer value, and that all objects have the same set of methods $\mathsf{Method} = \{f, \ldots\}$. The syntax of commands C is standard: C can be of the forms

$$c \mid C; C \mid \texttt{while}\,(b)\,\texttt{do}\,C \mid \texttt{if}\,(b)\,\texttt{then}\,C\,\texttt{else}\,C \mid x := \texttt{atomic}\,\{C\} \mid x := o.f(e)$$

where b and e denote Boolean and integer expressions over local variables, left unspecified. The syntax includes ***primitive commands*** c from a set PComm, sequential composition, conditionals, loops, `atomic` blocks and object method invocations. Primitive commands execute atomically, and they include assignments to local and global variables and a special `fault` command, which stops the execution of the program in an error state. Thus, `fault` encodes illegal computations, such as division by zero.

An ***atomic block*** $x := \texttt{atomic}\,\{C\}$ executes C as a ***transaction***, which the TM can ***commit*** or ***abort***. The system's decision is returned in the local variable x, which gets assigned distinguished values committed or aborted. We do not allow programs in our language to abort a transaction explicitly and forbid nested atomic blocks and, hence, nested transactions. We also assume that a program can invoke methods on transactional objects only inside atomic blocks and access global variables only outside them. Local variables can be accessed in both cases; however, threads cannot access local variables of other threads. Due to space constraints, we defer the formalisation of the rules on variable accesses to [11, Appendix A]. When we later define the semantics of our programming language, we mandate that, if a transaction is aborted, local variables are rolled back to the values they had at its start, and hence, the values written to them by the transaction cannot be observed by the following non-transactional code.

3 Model of Computations

To define the notion of observational refinement for our programming language and the TMS consistency condition, we need a formal model for program computations. To this end, we introduce *traces*, which are certain finite sequences of *actions*, each describing a single computation step (we do not consider infinite computations).

Definition 1. *Let* ActionId *be a set of action identifiers. A* ***TM interface action*** ψ *has one of the following forms:*

Request actions	*Matching response actions*
$(a, t, \mathsf{txbegin})$	$(a, t, \mathsf{OK}) \mid (a, t, \mathsf{aborted})$
$(a, t, \mathsf{txcommit})$	$(a, t, \mathsf{committed}) \mid (a, t, \mathsf{aborted})$
$(a, t, \mathsf{call}\ o.f(n))$	$(a, t, \mathsf{ret}(n')\ o.f) \mid (a, t, \mathsf{aborted})$

where $a \in$ ActionId, $t \in$ ThreadID, $o \in$ Obj, $f \in$ Method *and* $n, n' \in \mathbb{Z}$. *A* ***primitive action*** χ *has the form* (a, t, c), *where* $c \in$ PComm *is a primitive command. We use* φ *to range over actions of either type.*

TM interface actions denote the control flow of a thread t crossing the boundary between the program and the TM: ***request*** actions correspond to the control being transferred from the former to the latter, and ***response*** actions, the other way around. A txbegin action is generated upon entering an `atomic` block, and a txcommit action when a transaction tries to commit upon exiting an `atomic` block. Actions call and ret denote a call to and a return from an invocation of a method on a transactional object and are annotated with the method parameter or return value. The TM may abort a transaction at any point when it is in control; this is recorded by an aborted response action.

A ***trace*** τ is a finite sequence of actions satisfying certain natural well-formedness conditions (stated informally due to space constraints; see [11, Appendix B]): every

action in τ has a unique identifier; no action follows a `fault`; request and response actions are properly matched; for every thread t, $\tau|_t$ cannot contain a request action immediately followed by a primitive action; actions denoting the beginning and end of transactions are properly matched; call and ret actions occur only inside transactions; and commands in τ do not access local variables of other threads and do not access global variables when inside a transaction. We denote the set of traces by Trace. A ***history*** is a trace containing only TM interface actions; we use H, S to range over histories. We specify the behavior of a TM implementation by the set of possible interactions it can have with programs: a ***transactional memory*** $\mathcal{T}$ is a set of histories that is prefix-closed and closed under renaming action identifiers.

We denote irrelevant expressions by _ and use the following notation: $\tau(i)$ is the i-th element of τ; $\tau|_t$ is the projection of τ onto actions of the form $(\_, t, \_)$; $|\tau|$ is the length of τ; $\tau_1\tau_2$ is the concatenation of τ_1 and τ_2. We say that an action φ is in τ, denoted by $\varphi \in \tau$, if $\tau = \_\varphi\_$. The empty sequence of actions is denoted ε.

A ***transaction*** T is a nonempty trace such that it contains actions by the same thread, begins with a txbegin action and only its last action can be a committed or an aborted action. A transaction T is: ***committed*** if it ends with a committed action, ***aborted*** if it ends with aborted, ***commit-pending*** if it ends with txcommit, and ***live***, in all other cases. We refer to this as T's ***status***. A transaction T is ***completed*** if it is either committed or aborted, and ***visible*** if it contains a txcommit action. A transaction T is ***in a trace*** τ, written $T \in \tau$, if $\tau|_t = \tau_1 T \tau_2$ for some t, τ_1 and τ_2, where either T is completed or τ_2 is empty. We denote the set of all transactions in τ by $\mathsf{tx}(\tau)$ and use self-explanatory notation for various subsets of transactions: $\mathsf{committed}(\tau)$, $\mathsf{aborted}(\tau)$, $\mathsf{pending}(\tau)$, $\mathsf{live}(\tau)$, $\mathsf{visible}(\tau)$. For $\varphi \in \tau$, the ***transaction of*** φ ***in*** τ, denoted $\mathsf{txof}(\varphi, \tau)$, is the subsequence of τ comprised of all actions that are in the same transaction in τ as φ (undefined if φ does not belong to a transaction).

4 Transactional Memory Specification (TMS)

In this section we define the TMS [5] correctness condition in our setting. TMS was originally formulated using I/O automata; here we define it in a different style appropriate for our goals (we provide further comparison in Section 7). Since threads may communicate through global variables outside of transactions, they may observe the *real-time order* between non-overlapping transactions in a history. Therefore, this order is a crucial building block in the TMS definition, as is common in consistency conditions for shared-memory concurrency, such as opacity [2] or linearizability [12].

Definition 2. *Let $\psi = (\_, t, \_)$ and $\psi' = (\_, t', \_)$ be two actions in a history H; ψ **is before** ψ' **in the real-time order** in H, denoted by $\psi \prec_H \psi'$, if $H = H\psi H_2 H_2' \psi' H_3$ and either (i) $t = t'$ or (ii) $(\_, t', \mathsf{txbegin}) \in H_2'\psi'$ and either $(\_, t, \mathsf{committed}) \in \psi H_2$ or $(\_, t, \mathsf{aborted}) \in \psi H_2$. A transaction T **is before** an action ψ' **in the real-time order** in H, denoted by $T \prec_H \psi'$, if $\psi \prec_H \psi'$ for every $\psi \in T$. A transaction T **is before** a transaction T' **in the real-time order** in H, denoted by $T \prec_H T'$, if $T \prec_H T'$(1).*

The following *opacity relation* [2, 8] $H \sqsubseteq_{\mathsf{op}} S$ ensures that S is a a permutation of H preserving the real-time order.

Definition 3. *A history H is in the **opacity relation** with a history S, denoted by $H \sqsubseteq_{\mathsf{op}} S$, if $\forall \psi, \psi'. (\psi \in S \iff \psi \in H) \wedge (\psi \prec_H \psi' \implies \psi \prec_S \psi')$.*

Given a history H of program interactions with a concrete TM, TMS requires us to justify the behavior of all committed transactions in H by a single history S of the abstract TM, and to justify each response action ψ inside a transaction in H by an abstract history S_ψ. As we show in this paper, the existence of such justifications ensures that TMS implies observational refinement between the two TMs: the behavior of a program during some transaction in the history H of the program's interactions with the concrete TM can be reproduced when the program interacts with the abstract TM according to the history S or S_ψ. Below we use this insight when explaining the rationale for key TMS features.

The history S_ψ used to justify a response action ψ includes the transaction of ψ and a subset of transactions from H whose actions justify the response ψ. The following notion of a *possible past* of a history $H = H_1\psi$ defines all sets of transactions from H that can form S_ψ. Note that, if a transaction selected by this definition is aborted or commit-pending in H, its status is changed to committed when constructing S_ψ, as formalized later in Definition 5. Informally, the response ψ is given as if all the transactions in its possible past have taken effect and all the others have not. We first give the formal definition of a possible past, and then explain it using an example.

Definition 4. *A history $H_\psi = H_1'\psi$ is a **possible past** of a history $H = H_1\psi$, where ψ is a response action that it is not a* committed *or* aborted *action, if:*

(i) H_1' is a subsequence of H_1;

(ii) H_ψ is comprised of the transaction of ψ and some of the visible transactions in H: $\mathsf{tx}(H_\psi) \subseteq \{\mathsf{txof}(\psi, H)\} \cup \mathsf{visible}(H)$.

(iii) for every transaction $T \in H_\psi$, out of all transactions preceding T in the real-time order in H, the history H_ψ includes exactly the committed ones:

$$\forall T \in \mathsf{tx}(H_\psi).\, \forall T' \in \mathsf{tx}(H).\, T' \prec_H T \implies (T' \in \mathsf{tx}(H_\psi) \iff T' \in \mathsf{committed}(H)).$$

We denote the set of possible pasts of H by $\mathsf{TMSpast}(H)$.

We explain the definition using the history H of the trace shown in Figure 2; one of its possible pasts H_ψ consists of the transactions T_1, T_4 and T_5. According to (ii), the transaction of ψ (T_5 in Figure 2) is always included into any possible past, and live transactions are excluded: since they have not made an attempt to commit, they should not have an effect on ψ. Out of the visible transactions in H, we are allowed to select which ones to include (and, hence, treat as committed), subject to (iii): if we include a transaction T then, out of all transactions preceding T in the real-time order in H, we have to include exactly the committed ones. For example, since T_4 and T_5 are included in H_ψ, T_1 must also be included and T_3 must not. This condition is necessary for TMS to imply observational refinement. Informally, T_3 cannot be included into H_ψ because, in a program producing H, in between T_3 aborting and T_5 starting, thread t_2 could have communicated to thread t_3 the fact that T_3 has aborted, e.g., using a global variable g, as illustrated in Figure 2. When executing ψ, the code in T_5 may thus expect that T_3 did not take effect; hence, the result of ψ has to reflect this, so that the code behavior is preserved when replacing the concrete TM by an abstract one in observational refinement. This is a key idea used in our proof that TMS is necessary for observational refinement (Section 6.2). In contrast to T_3, we can include T_4 into H_ψ even if it is aborted or commit-pending. Since our language does not allow accessing

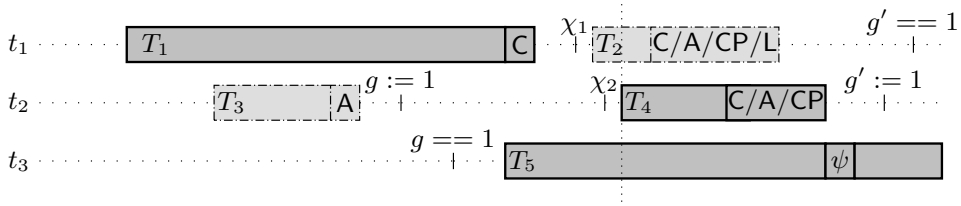


Fig. 2. Transactions T_1, T_4 and T_5 form one possible past of the history H of the trace shown. Allowed status of transactions in H is denoted as follows: committed – C, aborted – A, commit-pending – CP, live – L. The transaction T_5 executes only primitive actions after ψ in the trace.

global variables inside transactions, there is no way for the code in T_5 to find out about the status of T_4 from thread t_2, and hence, this code will not notice if the status of T_4 is changed to committed when replacing the concrete TM by an abstract one in observational refinement. For similar reasons, we can exclude T_2 from H_ψ even if it is committed. This idea is used in our proof that TMS is sufficient for observational refinement (Section 6.1).

Before giving the definition of TMS, we introduce operations used to change the status of transactions in a possible past of a history to committed. *Suffix commit completion* below converts commit-pending transactions into committed; then *completed possible past* defines a possible past with all transactions committed.

Definition 5. *A history H^c is a **suffix completion** of a history $H\psi$ if $H^c = H\psi H'$, every action in H' is either* committed *or* aborted*, and every transaction in H^c except possibly that of ψ, is completed. It is a **suffix commit completion** of H if H' consists of* committed *actions only. The sets of suffix completions and suffix commit completions of H are denoted* $\mathsf{comp}(H)$ *and* $\mathsf{ccomp}(H)$*, respectively.*

*A history H^c_ψ is a **completed possible past** of a history $H = H_1\psi$, if H^c_ψ is a suffix commit completion of a history obtained from a possible past $H'_1\psi$ of H by replacing all the* aborted *actions in H'_1 by* committed *actions. The set of completed possible pasts of H is denoted* $\mathsf{cTMSpast}(H)$*:*

$$\mathsf{cTMSpast}(H_1\psi) = \{H^c_\psi \mid \exists H'_1.\, H'_1\psi \in \mathsf{TMSpast}(H_1\psi) \wedge H^c_\psi \in \mathsf{ccomp}(\mathsf{com}(H'_1)\,\psi)\},$$

where $|\mathsf{com}(H'_1)| = |H'_1|$ *and*

$$\mathsf{com}(H'_1)(i) = (\text{if } (H'_1(i) = (a, t, \mathsf{aborted})) \text{ then } (a, t, \mathsf{committed}) \text{ else } H'_1(i)).$$

For example, one completed possible past of the history in Figure 2 consists of the transactions T_1, T_4 and T_5, with the status of the latter changed to committed if it was previously aborted or commit-pending. Note that a history H has a suffix completion only if H is of the form $H = H_1\psi$ where all the transactions in $H_1\psi$, except possibly that of ψ, are commit-pending or completed. Also, $\mathsf{cTMSpast}(H_1\psi) \neq \emptyset$ only if ψ is a response action.

The following definition of the *TMS relation* between TMs matches a history H arising from a concrete TM with a similar history S of an abstract TM. As part of this matching, we require that S preserves the real-time order of H. As in Definition 4(iii), this requirement is necessary to ensure observational refinement between the TMs: preserving the real-time order is necessary to preserve communication between threads when replacing the concrete TM with the abstract one.

Definition 6. *A history H is in the **TMS relation** with TM $\mathcal{T}$, denoted $H \sqsubseteq_{\mathsf{tms}} \mathcal{T}$, if:*

(i) $\exists H^c \in \mathsf{comp}(H|_{\neg\mathsf{live}}),\ S \in \mathcal{T}.\ H^c|_{\mathsf{com}} \sqsubseteq_{\mathsf{op}} S$*, where* $\cdot|_{\neg\mathsf{live}}$ *and* $\cdot|_{\mathsf{com}}$ *are the projections to actions by transactions that are not live and by committed transactions, respectively; and*

(ii) for every response action ψ such that it is not a committed *or* aborted *action and $H = H_1\psi H_2$, we have* $\exists H^c_\psi \in \mathsf{cTMSpast}(H_1\psi).\ \exists S_\psi \in \mathcal{T}.\ H^c_\psi \sqsubseteq_{\mathsf{op}} S_\psi$.

*A TM $\mathcal{T}_C$ is in the **TMS relation** with a TM $\mathcal{T}_A$, denoted by $\mathcal{T}_C \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A$, if* $\forall H \in \mathcal{T}_C.\ H \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A$.

5 Observational Refinement

Our main result relates TMS to *observational refinement*, which we introduce in this section. This requires defining the semantics of the programming language, i.e., the set of traces that computations of programs produce. Due to space constraints, we defer its formal definition to [11, Appendix C] and describe only its high-level structure. A ***state*** of a program records the values of all its variables: $s \in \mathsf{State} = \mathsf{Var} \to \mathbb{Z}$. The semantics of a program $P = C_1 \parallel \cdots \parallel C_m$ is given by the set of traces $[\![P, \mathcal{T}]\!](s) \subseteq \mathsf{Trace}$ it produces when executed with a TM $\mathcal{T}$ from an initial state s. To define this set, we first define the set of traces $[\![P]\!](s) \subseteq \mathsf{Trace}$ that a program can produce when executed from s with the behavior of the TM unrestricted, i.e., considering all possible values the TM can return to object method invocations and allowing transactions to commit or abort arbitrarily. We then restrict to the set of traces produced by P when executed with $\mathcal{T}$ by selecting those traces that interact with the TM in a way consistent with $\mathcal{T}$: $[\![P, \mathcal{T}]\!](s) = \{\tau \mid \tau \in [\![P]\!](s) \wedge \mathsf{history}(\tau) \in \mathcal{T}\}$, where $\mathsf{history}(\cdot)$ projects to TM interface actions. The definition of $[\![P]\!](s)$ follows the intuitive semantics of our programming language. In particular, it mandates that local variables be rolled back upon a transaction abort and includes traces corresponding to incomplete program computations into $[\![P]\!](s)$.

We can now define *observations* and *observational refinement*. Informally, given a trace τ of a client program, we consider observable: (i) the sequence of actions performed outside transactions in τ; (ii) the per-thread sequence of actions in τ excluding uncommitted transactions; and (iii) whether a τ ends with `fault` or not. Then observational refinement between a concrete TM $\mathcal{T}_C$ and an abstract one $\mathcal{T}_A$ states that every observable behavior of a program P using $\mathcal{T}_C$ can be reproduced when P uses $\mathcal{T}_A$. Hence, any conclusion about its observable behavior that a programmer makes assuming $\mathcal{T}_A$ will carry over to $\mathcal{T}_C$. Since our notion of observations excludes actions performed inside aborted or live transactions other than faulting, the programmer cannot make any conclusions about them. But, crucially, the programmer can be sure that, if a program is non-faulting under $\mathcal{T}_A$, it will stay so under $\mathcal{T}_C$. An action $\varphi \in \tau$ is ***transactional*** if $\varphi \in T$ for some $T \in \tau$, and ***non-transactional*** otherwise. We denote by $\tau|_{\mathsf{trans}}$ and $\tau|_{\neg\mathsf{trans}}$ the projections of τ to transactional and non-transactional actions.

Definition 7. *The **thread-local observable behavior** of thread t in a trace τ, denoted by* $\mathsf{observable}_t(\tau)$*, is $\lightning$ if $\tau|_t$ ends with a* `fault` *action, and $(\tau|_t)|_{\mathsf{obs}}$ otherwise, where $\cdot|_{\mathsf{obs}}$ denotes the projection to non-transactional actions and actions by committed transactions. A TM $\mathcal{T}_C$ **observationally refines** a TM $\mathcal{T}_A$, denoted by $\mathcal{T}_C \preceq \mathcal{T}_A$, if for every program P, state s and trace $\tau \in [\![P, \mathcal{T}_C]\!](s)$ we have: (i)* $\exists \tau' \in [\![P, \mathcal{T}_A]\!](s).\ \tau'|_{\neg\mathsf{trans}} = \tau|_{\neg\mathsf{trans}}$*; and (ii)* $\forall t.\ \exists \tau'_t \in [\![P, \mathcal{T}_A]\!](s).\ \mathsf{observable}_t(\tau'_t) = \mathsf{observable}_t(\tau)$.

6 Main Result

The main result of this paper is that the TMS relation is equivalent to observational refinement for abstract TMs that enjoy certain natural closure properties. Their formulation relies on the following notions.

A history H_a is an ***immediate abort extension of a history*** H if H is a subsequence of H_a, and whenever $\psi \in H_a$ and $\psi \notin H$ we have: (i) $\psi = (\_, \_, \mathsf{txbegin})$ or $\psi = (\_, \_, \mathsf{aborted})$, (ii) if $\psi = (\_, t, \mathsf{txbegin})$ then $H_a = H'_a \psi\, (\_, t, \mathsf{aborted})\, \_$, where $H'_a \in \{\varepsilon, \_\,(\_, \_, \mathsf{committed}), \_\,(\_, \_, \mathsf{aborted})\}$, and (iii) if $\psi = (\_, \_, \mathsf{aborted})$ then there exists $\psi' \notin H$ such that $H_a = \_\psi'\psi\_$. We denote by $\mathsf{addab}(H)$ the set of all immediate abort extensions of H. Informally, a history $H_a \in \mathsf{addab}(H)$ is an extension of H with transactions that abort immediately after their invocation. Note that the added transactions are placed either right before other transactions begin or right after they complete.

A history H_c is a ***non-interleaved completion*** of a history H if H is a subsequence of H_c, $\mathsf{pending}(H_c) = \emptyset$ and whenever $\psi \in H_c$ and $\psi \notin H$ we have $H_c = \_\,(\_, t, \mathsf{txcommit})\, \psi\_$ and either $\psi = (\_, t, \mathsf{committed})$ or $\psi = (\_, t, \mathsf{aborted})$. We denote the set of non-interleaved completions of H by $\mathsf{nicomp}(H)$. Informally, $H' \in \mathsf{nicomp}(H)$ completes each commit-pending transaction in H by adding a committed or aborted action at its end.

The required closure properties are formulated as follows:

CLP1 A TM $\mathcal{T}$ is ***closed under immediate aborts*** if whenever $H \in \mathcal{T}$ and $\mathsf{aborted}(H) = \emptyset$, we also have $H' \in \mathcal{T}$ for any history $H' \in \mathsf{addab}(H)$.

CLP2 A TM $\mathcal{T}$ is ***closed under removing transaction responses*** if whenever $H_1(\_, t, \mathsf{aborted})H_2 \in \mathcal{T}$ or $H_1(\_, t, \mathsf{committed})H_2 \in \mathcal{T}$ for H_2 not containing actions by t, we also have $H_1H_2 \in \mathcal{T}$.

CLP3 A TM $\mathcal{T}$ is ***closed under removing live and aborted transactions*** if whenever $H \in \mathcal{T}$, we also have $H' \in \mathcal{T}$ for any history H' which is a subsequence of H such that $\mathsf{committed}(H') = \mathsf{committed}(H)$, $\mathsf{pending}(H') = \mathsf{pending}(H)$, $\mathsf{live}(H') \subseteq \mathsf{live}(H)$ and $\mathsf{aborted}(H') \subseteq \mathsf{aborted}(H)$.

CLP4 A TM $\mathcal{T}$ is ***closed under completing commit-pending transactions*** if whenever $H \in \mathcal{T}$, we have $\mathsf{nicomp}(H) \cap \mathcal{T} \neq \emptyset$.

These properties are satisfied by the expected TM specification that executes every transaction atomically [8].

Theorem 1. *Let $\mathcal{T}_C$ and $\mathcal{T}_A$ be transactional memories.*
(i) If $\mathcal{T}_A$ satisfies CLP1 and CLP2, then $\mathcal{T}_C \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A \Longrightarrow \mathcal{T}_C \preceq \mathcal{T}_A$.
(ii) If $\mathcal{T}_A$ satisfies CLP3 and CLP4, then $\mathcal{T}_C \preceq \mathcal{T}_A \Longrightarrow \mathcal{T}_C \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A$.

6.1 Proof of Theorem 1(i) (Sufficiency)

Let us fix a program $P = C_1 \parallel \ldots \parallel C_m$ and a state s. As we have noted before, the main subtlety of TMS lies in justifying the behavior of a live transaction under $\mathcal{T}_C$ by a history of $\mathcal{T}_A$ where the committed/aborted status of some transactions is changed, as formalized by the use of $\mathsf{cTMSpast}$ in Definition 6(ii). Correspondingly, the most challenging part of the proof is to show that a trace from $[\![P, \mathcal{T}_C]\!](s)$ with a `fault` inside a live transaction can be transformed into a trace with the `fault` from $[\![P, \mathcal{T}_A]\!](s)$. The following lemma describes the first and foremost step of this transformation: given a

trace $\tau \in [\![P]\!](s)$ with a live transaction and a history $H^c_\psi \in \mathsf{cTMSpast}(\mathsf{history}(\tau))$, the lemma converts τ into another trace from $[\![P]\!](s)$ that contains the same live transaction, but whose history of non-aborted transactions is H^c_ψ. In other words, this establishes that the live transaction cannot notice changes in the committed/aborted status of other transactions done by cTMSpast. Let $\tau|_{\neg\mathsf{abortedtx}}$ be the projection of τ excluding aborted transactions.

Lemma 1 (Live transaction insensitivity). *Let $\tau = \tau_1\psi\tau_2 \in [\![P]\!](s)$ be such that ψ is a response action by thread t_0 that is not a* committed *or* aborted *action and τ_2 is a sequence of primitive actions by thread t_0. For any $H^c_\psi \in \mathsf{cTMSpast}(\mathsf{history}(\tau))$ there exists $\tau_\psi \in [\![P]\!](s)$ such that $\mathsf{history}(\tau_\psi)|_{\neg\mathsf{abortedtx}} = H^c_\psi$ and $\tau_\psi|_{t_0} = \tau|_{t_0}$.*

Proof. We first show how to construct τ_ψ and then prove that it satisfies the required properties. We illustrate the idea of its construction using the trace τ in Figure 2. Let $\mathsf{history}(\tau) = H_1\psi$. Since $H^c_\psi \in \mathsf{cTMSpast}(H)$, by Definition 5 there exist histories H'_1, H''_1, and H^{cc} such that

$$H'_1\psi \in \mathsf{TMSpast}(H_1\psi) \ \wedge\ H''_1 = \mathsf{com}(H'_1) \ \wedge\ H^c_\psi = H''_1\psi H^{cc} \in \mathsf{ccomp}(H''_1\psi).$$

Recall that, for the τ in Figure 2, $H'_1\psi$ consists of the transactions T_1, T_4 and T_5. Then H''_1 is obtained from H'_1 by changing the last action of T_4 to committed if it was aborted; H^c_ψ is obtained by completing T_4 with a committed action if it was commit-pending. The trickiness of the proof comes from the fact that just mirroring these transformations on τ may not yield a trace of the program P: for example, if T_4 aborted, the code in thread t_2 following T_4 may rely on this fact, communicated to it by the TM via a local variable. Fortunately, we show that it is possible to construct the required trace by erasing certain suffixes of every thread and therefore getting rid of the actions that could be sensitive to the changes of transaction status, such as those following T_4. This erasure has to be performed carefully, since threads can communicate via global variables: for example, the value written by the assignment to g' in the code following T_4 may later be read by t_1, and, hence, when erasing the the former, the latter action has to be erased as well. We now explain how to truncate τ consistently.

Let ψ^b be the last txbegin action in $H'_1\psi$; then for some traces τ^b_1 and τ^b_2 we have $\tau = \tau^b_1\psi^b\tau^b_2\psi\tau_2$. For the τ in Figure 2, ψ^b is the txbegin action of T_4. Our idea is, for every thread other than t_0, to erase all its actions that follow the last of its transactions included into $H'_1\psi$ or its last non-transactional action preceding ψ^b, whichever is later. Formally, for every thread t, let τ^I_t denote the prefix of $\tau|_t$ that ends with the last TM interface action of t in $H'_1\psi$, or ε if no such action exists. For example, in Figure 2, $\tau^I_{t_1}$ and $\tau^I_{t_2}$ end with the last TM interface actions of T_1 and T_4, respectively. Similarly, let τ^N_t denote the prefix of $\tau|_t$ that ends in the last non-transactional action of t in τ^b_1, or ε if no such action exists. For example, in Figure 2, $\tau^N_{t_1}$ and $\tau^N_{t_2}$ end with χ_1 and χ_2, respectively. Let $\tau_{t_0} = \tau|_{t_0}$ and for each $t \neq t_0$ let τ_t be τ^I_t, if $|\tau^N_t| < |\tau^I_t|$, and τ^N_t, otherwise. We then let the truncated trace τ' be the subsequence of τ such that $\tau'|_t = \tau_t$ for each t. Thus, for the τ in Figure 2, in the corresponding trace τ' the actions of t_1 end with χ_1 and those of t_2 with the last action of T_4; note that this erases both operations on g'. To construct τ_ψ from τ', we mirror the transformations of H'_1 into H''_1 and H^c_ψ. Let τ'' be defined by $|\tau''| = |\tau'|$ and

$$\tau''(i) = (\text{if } (\tau'(i) = (a, t, \mathsf{aborted}) \wedge \tau'(i) \in H'_1)) \text{ then } (a, t, \mathsf{committed}) \text{ else } \tau'(i)).$$

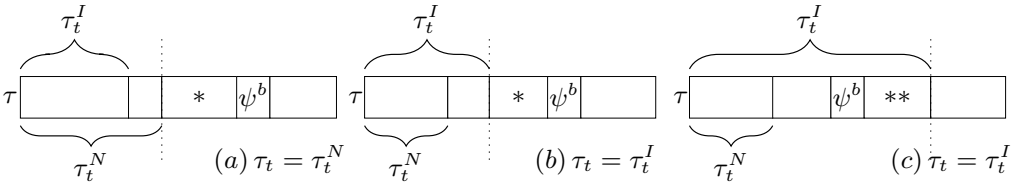


Fig. 3. Cases in the proof of Lemma 1. * all actions by t are transactional; ** all actions by t come from a single transaction, started before or by ψ^b

Then we let $\tau_\psi = \tau'' H^{cc}$.

We first prove that $\tau_\psi|_{t_0} = \tau|_{t_0}$. Let $T = \mathsf{txof}(\psi, H_1\psi)$; then by Definition 4(ii), $T \in H_1'\psi$. Hence, by Definition 4(iii) we have

$$\forall T'.\, T' \prec_{H_1'\psi} T \iff T' \prec_{H_1\psi} T \ \wedge\ T' \in \mathsf{committed}(H_1\psi), \tag{1}$$

so that $(H_1'\psi)|_{t_0}$ does not contain aborted transactions and $\tau''|_{t_0} = \tau'|_{t_0} = \tau|_{t_0}$. Besides, $H^{cc}|_{t_0} = \varepsilon$ and, hence, $\tau_\psi|_{t_0} = \tau''|_{t_0} = \tau|_{t_0}$.

We now sketch the proof that $\tau_\psi \in [\![P]\!](s)$, appealing to the intuitive understanding of the programming language semantics. To this end, we show that τ' and then τ'' belong to $[\![P]\!](s)$. We start by analyzing how the trace $\tau|_t$ is truncated to τ_t for every thread $t \neq t_0$. Let us make a case split on the relative positions of τ_t^N, τ_t^I and ψ^b in τ. There are three cases, shown in Figure 3. Either $\tau_t = \tau_t^N$ (a, thread t_1 in Figure 2) or $\tau_t = \tau_t^I$ (b, c). In the former case, ψ^b has to come after the end of τ_t^N. In the latter case, either ψ^b comes after the end of τ_t^I (b) or is its last action or precedes the latter (c, thread t_2 in Figure 2).

By the choice of τ_t^N, in (a) and (b) the fragment of τ in between the end of τ_t^N and ψ^b can contain only those actions by t that are transactional (T_2 in Figure 2). By the choice of τ_t^I and ψ^b, in (c) the fragment of τ in between ψ^b and the end of τ_t^I cannot contain a $\mathsf{txbegin}$ action by t; hence, by the choice of τ_t^N it can contain only those actions by t that are transactional. Furthermore, these have to come from a single transaction, started either by ψ^b or before it (T_4 in Figure 2). Finally, by the choice of ψ^b the actions of t_0 following ψ^b are transactional and come from the transaction of ψ, also started either by ψ^b or before it (T_5 in Figure 2). Given this analysis, the transformation from τ to τ' can be viewed as a sequence of two: (i) erase all actions following ψ^b, except those in some of transactions that were already ongoing at this time; (ii) erase some suffixes of threads containing only transactional actions. Since transactional actions do not access global variables, they are not affected by the actions of other threads. Furthermore, as we noted in Section 5, $[\![P]\!](s)$ includes incomplete program computations. This allows us to conclude that $\tau' \in [\![P]\!](s)$.

We now show that τ'' is valid, again referring to cases (a-c). Let $T = \mathsf{txof}(\psi^b, H_1\psi)$; then $T \in H_1'\psi$ by the choice of ψ^b and by Definition 4(iii) we get (1). Hence, for threads t falling into cases (a) or (b), $\tau'|_t$ does not contain aborted transactions that are also in $H_1'\psi$. For threads t falling into case (c), an aborted transaction by t included into $H_1'\psi$ can only be the last one in $\tau'|_t$. Finally, above we established that $(H_1'\psi)|_{t_0}$ does not contain aborted transactions. Hence, transactions in τ' whose status is changed from aborted to committed when switching to τ'' do not have any actions following them in τ'. Furthermore, $[\![P]\!](s)$ allows committing or aborting transactions

arbitrarily. This allows us to conclude that $\tau'' \in [\![P]\!](s)$. For the same reason, we get $\tau_\psi \in [\![P]\!](s)$.

Finally, we show that $\mathsf{history}(\tau_\psi)|_{\neg\mathsf{abortedtx}} = H^c_\psi$. It is sufficient to show that $\mathsf{history}(\tau'')|_{\neg\mathsf{abortedtx}} = H''_1\psi$; since $\tau_\psi = \tau''H^{cc}$ and H^{cc} contains only committed actions, this would imply

$$\mathsf{history}(\tau_\psi)|_{\neg\mathsf{abortedtx}} = \mathsf{history}(\tau''H^{cc})|_{\neg\mathsf{abortedtx}} = \mathsf{history}(\tau'')|_{\neg\mathsf{abortedtx}}H^{cc} = H''_1\psi H^{cc} = H^c_\psi.$$

By the choice of τ^I_t for $t \neq t_0$, every transaction in $(H'_1\psi)|_t$ is also in τ^I_t. Hence, $H'_1\psi$ is a subsequence of $\mathsf{history}(\tau')$. By the definition of τ'' and H''_1, $H''_1\psi$ is a subsequence of $\mathsf{history}(\tau'')$. Then since $H''_1\psi$ does not contain aborted transactions, $H''_1\psi$ is a subsequence of $\mathsf{history}(\tau'')|_{\neg\mathsf{abortedtx}}$.

Thus, to prove $\mathsf{history}(\tau'')|_{\neg\mathsf{abortedtx}} = H''_1\psi$ it remains to show that every non-aborted transaction in $\mathsf{history}(\tau'')$ is in $H''_1\psi$. Since the construction of τ'' from τ' changes the status of only those transactions that belong to $H'_1\psi$, it is sufficient to show that every non-aborted transaction in $\mathsf{history}(\tau')$ is in $H'_1\psi$. Here we only consider the case when such a transaction is by a thread $t \neq t_0$ and $\tau'|_t = \tau^N_t \neq \varepsilon$; we cover the other cases in [11, Appendix D]. Let χ^N_t be the last action in τ^N_t and $T = \mathsf{txof}(\psi^b, H_1\psi) \in H'_1\psi$. Then by Definition 4(iii) we get (1). Since χ^N_t comes before ψ^b in $H_1\psi$, any transaction T' in $\tau'|_t$ is such that $T' \prec_{H_1\psi} T$, which together with (1) implies the required. This concludes the proof that $\mathsf{history}(\tau'')|_{\neg\mathsf{abortedtx}} = H''_1\psi$. □

We now give the other lemmas necessary for the proof. Definition 6 matches a history of $\mathcal{T}_C$ with one of $\mathcal{T}_A$ using the opacity relation, possibly after transforming the former with cTMSpast. The following lemma is used to transform a trace of P accordingly. The lemma shows that, if we consider only traces where aborted transactions abort immediately (i.e., are of the form $(\_, \_, \mathsf{txbegin})\ (\_, \_, \mathsf{aborted})$), then the opacity relation implies observational refinement with respect to observing non-transactional actions and thread-local trace projections. This result is a simple adjustment of the one about the sufficiency of opacity for observational refinement to our setting [8, Theorem 16] (it was proved in [8] for a language where local variables are *not* rolled back upon a transaction abort; this difference, however, does not matter if aborted transactions abort immediately).

Lemma 2. *Consider $\tau \in [\![P]\!](s)$ such that all the aborted transactions in τ abort immediately. Let S be such that $\mathsf{history}(\tau) \sqsubseteq_{\mathsf{op}} S$. Then there exists $\tau' \in [\![P]\!](s)$ such that $\mathsf{history}(\tau') = S$, $\tau|_{\neg\mathsf{trans}} = \tau'|_{\neg\mathsf{trans}}$ and $\forall t.\, \tau'|_t = \tau|_t$.*

Let $\tau|_{\neg\mathsf{abortact}}$ be the trace obtained from τ by removing all actions inside aborted transactions, so that every such transaction aborts immediately. We can benefit from Lemma 2 because local variables are rolled back if a transaction aborts, and, hence, applying $\cdot|_{\neg\mathsf{abortact}}$ to a trace preserves its validity.

Proposition 1. $\forall\tau.\, \tau \in [\![P]\!](s) \implies \tau|_{\neg\mathsf{abortact}} \in [\![P]\!](s)$.

Finally, Definition 6 matches only histories of committed transactions, but the histories of the traces in Lemma 2 also contain aborted transactions. Fortunately, the following lemma allows us to add empty aborted transactions into the abstract history while preserving the opacity relation.

Lemma 3. *Let H be a history where all aborted transactions abort immediately and S be such that $H|_{\neg\mathsf{abortedtx}} \sqsubseteq_{\mathsf{op}} S$. There exists a history $S' \in \mathsf{addab}(S)$ such that $H \sqsubseteq_{\mathsf{op}} S'$.*

Definition 6(i), Proposition 1 and Lemmas 2 and 3 can be used to prove that the TMS relation preserves non-transactional actions and thread-local observable behavior of threads whose last action is not a `fault`.

Lemma 4. *If $\mathcal{T}_C \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A$ and $\mathcal{T}_A$ satisfies CLP1 and CLP2, then*

$$\forall \tau \in [\![P, \mathcal{T}_C]\!](s).\, \exists \tau' \in [\![P, \mathcal{T}_A]\!](s).\, (\tau'|_{\neg\mathsf{trans}} = \tau|_{\neg\mathsf{trans}}) \wedge (\forall t.\, (\tau'|_t)|_{\mathsf{obs}} = (\tau|_t)|_{\mathsf{obs}}).$$

Proof of Theorem 1(i). Given Lemma 4, we only need to establish the preservation of faults inside transactions. Consider $\tau_0 \in [\![P, \mathcal{T}_C]\!](s)$ such that $\tau_0 = \tau_1 \psi \tau_2 \chi$, where $\chi = (\_, t_0, \mathtt{fault})$ is transactional and ψ is the last TM interface action by thread t_0. Then $\tau_2|_{t_0}$ consists of transactional actions and thus does not contain accesses to global variables. Hence, $\tau = \tau_1 \psi (\tau_2|_{t_0}) \chi \in [\![P, \mathcal{T}_C]\!](s)$. By our assumption, $\mathcal{T}_C \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A$. Then there exists $H^c_\psi \in \mathsf{cTMSpast}(\mathsf{history}(\tau))$ and $S \in \mathcal{T}_A$ such that $H^c_\psi \sqsubseteq_{\mathsf{op}} S$. By Lemma 1, for some trace τ_ψ we have $\tau_\psi \in [\![P]\!](s)$, $\mathsf{history}(\tau_\psi)|_{\neg\mathsf{abortedtx}} = H^c_\psi$ and $\tau_\psi|_{t_0} = \tau|_{t_0}$. By Proposition 1, $\tau_\psi|_{\neg\mathsf{abortact}} \in [\![P]\!](s)$. Using Lemma 3, we get a history S' such that $\mathsf{history}(\tau_\psi|_{\neg\mathsf{abortact}}) \sqsubseteq_{\mathsf{op}} S'$ and $S' \in \mathsf{addab}(S)$. Since $S \in \mathcal{T}_A$ and $\mathcal{T}_A$ is closed under immediate aborts (CLP1), we get $S' \in \mathcal{T}_A$. Hence, by Lemma 2, for some $\tau' \in [\![P, \mathcal{T}_A]\!](s)$ we have $\tau'|_{t_0} = \tau_\psi|_{t_0} = \tau|_{t_0} = \_\chi$, as required. □

6.2 Proof Sketch for Theorem 1(ii) (Necessity)

Consider $\mathcal{T}_C$ and $\mathcal{T}_A$ such that $\mathcal{T}_C \preceq \mathcal{T}_A$ and $\mathcal{T}_A$ satisfies the closure conditions stated in the theorem. To show that for any $H_0 \in \mathcal{T}_C$ we have $H_0 \sqsubseteq_{\mathsf{tms}} \mathcal{T}_A$, we have to establish conditions (i) and (ii) from Definition 6. We sketch the more interesting case of (ii), in which $H_0 = H_1 \psi H_2 = H H_2 \in \mathcal{T}_C$, where ψ is a response action by a thread t_0 that is not a committed or aborted action. We need to find $H^c \in \mathsf{cTMSpast}(H)$ and $S \in \mathcal{T}_A$ such that $H^c \sqsubseteq_{\mathsf{op}} S$.

To this end, we construct a program P_H (as we explain further below) where every thread t performs the sequence of transactions specified in $H|_t$. The program monitors certain properties of the TM behavior, e.g., checking that the return values obtained from methods of transactional objects in committed transactions correspond to those in H and that the real-time order between actions includes that in H. If these properties hold, thread t_0 ends by executing the `fault` command. Let s be a state with all variables set to distinguished values. We next construct a trace $\tau \in [\![P_H, \mathcal{T}_C]\!](s)$ such that $\mathsf{history}(\tau) = H$ and t_0 faults in τ. By Definition 7, there exists $\tau' \in [\![P_H, \mathcal{T}_A]\!](s)$ such that t_0 faults in τ'. However, the program P_H is constructed so that t_0 can fault in τ' only if the properties of the TM behaviour the program monitors hold, and thus H is related to $\mathsf{history}(\tau')$ in a certain way. This relationship allows us to construct $H^c \in \mathsf{cTMSpast}(H)$ from H and $S \in \mathcal{T}_A$ from $\mathsf{history}(\tau')$ such that $H^c \sqsubseteq_{\mathsf{op}} S$.

In more detail, thread t_0 in P_H monitors the return status of every transaction and the return values obtained inside the atomic blocks corresponding to transactions committed in $H|_{t_0}$ and the (live) transaction of ψ. If there is a mismatch with $H|_{t_0}$, this is recorded in a special local variable. At the end of the transaction of ψ, t_0 checks the variable and faults if the TM behavior matched $H|_{t_0}$. This construction is motivated by

the fact that faulting is the only observation Definition 7 allows us to make about the behavior of the live transaction of ψ. Since the definition does not correlate actions by threads t other than t_0 between τ and τ', such threads monitor TM behavior differently: if there is a mismatch with $H|_t$, a thread t faults immediately. Since a trace can have at most one `fault` and t_0 faults in τ', this ensures that any committed transaction in τ' behaves as in H.

To check whether an execution of P_H complies with the real-time order in H, for each transaction in H, we introduce a global variable g, which is initially 0 and is set to 1 by the thread executing the transaction right after the transaction completes, by a command following the corresponding atomic block. Before starting a transaction, each thread checks whether all transactions preceding this one in the real-time order in H have finished by reading the corresponding g variables. Thread t_0 records the outcome in the special local variable checked at the end; all other threads fault upon detecting a mismatch.

Let $H' = \mathsf{history}(\tau')$. This construction of P_H allows us to infer that: (i) the projection of $H'|_{t_0}$ to committed transactions and $\mathsf{txof}(\psi, H')$ is equal to the corresponding projection of $H|_{t_0}$; (ii) for all other threads t a similar relationship holds for the prefix of $H'|_t$ ending with the last transaction preceding $\mathsf{txof}(\psi, H')$ in the real-time order; (iii) the real-time order in H' includes that in H. Transactions concurrent with $\mathsf{txof}(\psi, H')$ in H' may behave differently from H. However, checks done by P_H inside these transactions ensure that, if such a transaction T is visible in H', then the return values inside T match those in H. The checks on the global variables g done right before T also ensure that all transactions preceding T in the real-time order in H commit or abort in H' as prescribed by H. This relationship between H and H' allows us to establish the requirements of Definition 6(ii). □

7 Related Work

When presenting TMS [5], Doherty et al. discuss why it allows programmers to think only of serial executions of their programs, in which the actions of a transaction appear consecutively. This discussion—corresponding to our sufficiency result—is informal, since the paper lacks a formal model for programs and their semantics. Most of it explains how Definition 6(i) ensures the correctness of committed transactions. The discussion of the most challenging case of live transactions—corresponding to Definition 6(ii) and our Lemma 1—is one paragraph long. It only roughly sketches the construction of a trace with an abstract history allowed by TMS and does not give any reasoning for why this trace is a valid one, but only claims that constraints in Definition 6(ii) ensure this. This reasoning is very delicate, as indicated by our proof of Lemma 1, which carefully selects which actions to erase when transforming the trace. Moreover, Doherty et al. do not try to argue that TMS is the weakest condition possible, as we established by our necessity result.

Another TM consistency condition, weaker than opacity but incomparable to TMS, is *virtual world consistency* (*VWC*) [3]. Like TMS, VWC allows every operation in a live or aborted transaction to be justified by a separate abstract history. However, it places different constraints on the choice of abstract histories, which do not take into account the real-time order between actions. Because of this, VWC does not imply observational refinement for our programming language: taking into account the real-

time order is necessary when threads can communicate via global variables outside transactions.

Our earlier paper [8] laid the groundwork for relating TM consistency and observational refinement, and it includes a detailed comparison with related work on opacity and observational refinement. The present paper considers a much more challenging case of a language where local variables are rolled back upon an abort. To handle this case, we developed new techniques, such as establishing the live transaction insensitivity property (Lemma 1) to prove sufficiency and proposing monitor programs for the nontrivial constraints used in the TMS definition to prove necessity. Similarly to [8] and other papers using observational refinement to study consistency conditions [13, 14], we reformulate TMS so that it is not restricted to a particular abstract TM $\mathcal{T}_A$. This generality, not allowed by the original TMS definition, has two benefits. First, our reformulation can be used to compare two TM implementations, e.g., an optimized and an unoptimized one. Second, dealing with the general definition forces us to explicitly state the closure properties required from the abstract TM, rather than having them follow implicitly from its atomic behavior.

Acknowledgements. We thank Mohsen Lesani, Victor Luchangco and the anonymous reviewers for comments that helped us improve the paper. This work was supported by EU FP7 project ADVENT (308830) and by the Broadcom Foundation and Tel Aviv University Authentication Initiative.

References

1. Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM **26** (1979) 631–653
2. Guerraoui, R., Kapalka, M.: On the correctness of transactional memory. In: PPOPP. (2008) 175–184
3. Imbs, D., Raynal, M.: Virtual world consistency: A condition for STM systems (with a versatile protocol with invisible read operations). Theor. Comput. Sci. **444** (2012) 113–127
4. Attiya, H., Hans, S., Kuznetsov, P., Ravi, S.: Safety of deferred update in transactional memory. In: ICDCS. (2013)
5. Doherty, S., Groves, L., Luchangco, V., Moir, M.: Towards formally specifying and verifying transactional memory. Formal Aspects of Computing **25** (2013) 769–799
6. He, J., Hoare, C., Sanders, J.: Prespecification in data refinement. Information Processing Letters **25** (1987) 71 – 76
7. He, J., Hoare, C., Sanders, J.: Data refinement refined. In: ESOP. (1986) 187–196
8. Attiya, H., Gotsman, A., Hans, S., Rinetzky, N.: A programming language perspective on transactional memory consistency. In: PODC. (2013) 309–318
9. Scala STM Expert Group: Scala STM quick start guide (2012) http://nbronson.github.io/scala-stm/quick_start.html.
10. Lesani, M., Luchangco, V., Moir, M.: Putting opacity in its place. In: WTTM. (2012)
11. Attiya, H., Gotsman, A., Hans, S., Rinetzky, N.: Safety of live transactions in transactional memory: Tms is necessary and sufficient. Technical Report CS-2014-02, Technion (2014)
12. Herlihy, M., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems **12** (1990) 463–492
13. Gotsman, A., Yang, H.: Liveness-preserving atomicity abstraction. In: ICALP (2). (2011) 453–465
14. Filipovic, I., O'Hearn, P.W., Rinetzky, N., Yang, H.: Abstraction for concurrent objects. In: ESOP. (2009) 252–266

Brief Announcement: Concurrency-Aware Linearizability

Technical Report - Tel Aviv University - School of Computer Science

Nir Hemed
The Blavatnik School of Computer Science
Raymond and Beverly Sackler Faculty
of Exact Sciences
Tel Aviv University, Israel
nirh@mail.tau.ac.il

Noam Rinetzky
The Blavatnik School of Computer Science
Raymond and Beverly Sackler Faculty
of Exact Sciences
Tel Aviv University, Israel
maon@cs.tau.ac.il

ABSTRACT

Linearizabilty allows to describe the behaviour of concurrent objects using sequential specifications. Unfortunately, as we show in this paper, sequential specifications cannot be used for concurrent objects whose *observable behaviour* in the presence of concurrent operations *should* be different than their behaviour in the sequential setting. As a result, such *concurrency-aware objects* do not have formal specifications, which, in turn, precludes formal verification.

In this paper we present *Concurrency Aware Linearizability* (CAL), a new correctness condition which allows to formally specify the behaviour of a certain class of concurrency-aware objects. Technically, CAL is formalized as a strict extension of linearizability, where *concurrency-aware specifications* are used instead of sequential ones. We believe that CAL can be used as a basis for modular formal verification techniques for concurrency-aware objects.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming; D.2.4 [**Software Engineering**]: Software/Program Verification

Keywords

Linearizability; sequential specification; concurrent specification

1. INTRODUCTION

Linearizability [4] is a property of the externally-observable behaviour of concurrent objects [4]. Intuitively, a concurrent object is linearizable if in every execution each operation seems to take effect instantaneously between its invocation and response, and the resulting sequence of (seemingly instantaneous) operations respects a given sequential specification. Unfortunately, as we show below, for some concurrent objects it is *impossible* to provide a sequential specification: their behaviour in the presence of concurrent (overlapping) operations is, and should be, *observably different* from their behaviour in the sequential setting. For these objects, which we refer to as *Concurrency-Aware Concurrent Objects (CA-objects)*, the traditional notion of linearizability is simply not expressive enough to allow for describing all desired behaviours without introducing undesired ones. As a result, CA-objects are not given a formal specification. The lack of formal specifications is problematic as it precludes formal proofs.

Concurrency-Aware Linearizabilty (CAL) is a correctness condition which addresses the aforementioned problem. CAL enables programmers to provide natural and intuitive specifications for an important class of CA-objects. Technically, CAL is an extension of linearizabilty where *Concurrency-Aware specifications* are used to describe concurrency-dependent behaviours. Sequential specifications are a special case of concurrency-aware specifications in which concurrent behaviours can be explained by sequential ones.

Running Example. `Exchanger` objects (as found, e.g., in `java.util.concurrent.Exchanger`) serve as a synchronization point at which threads can pair up and *atomically swap* elements. `Exchangers` are useful in applications such as genetic algorithms and pipeline designs, and are embedded in practice in thread-pool implementations as well as other higher-level data structures [5, 6].

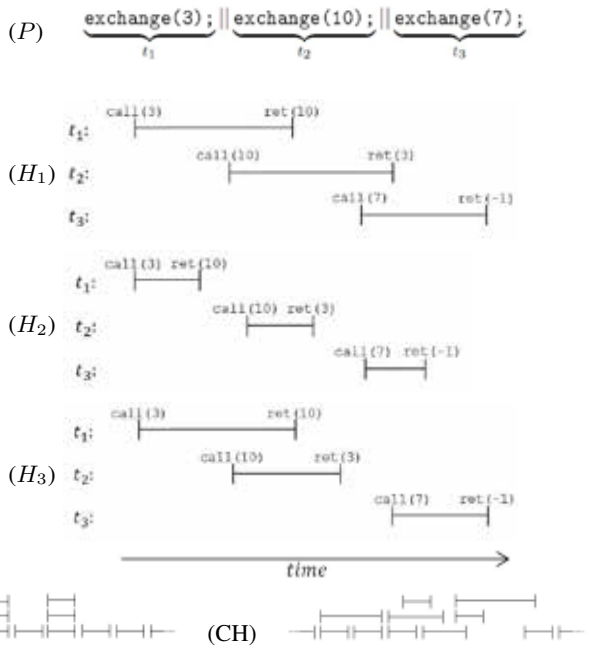
Figure 1(`E`) shows a simplified version of the wait-free exchanger of [5] in which retires are omitted. Intuitively, a client thread uses the `Exchanger` by invoking the `exchange()` method with a value that it offers to swap (in our case a positive `int`). `exchange()` attempts to find a partner thread and, if successful, instantaneously exchanges the offered value with the one offered by the partner. If a partner thread is not found, `exchange()` returns `-1`, indicating that the operation has failed. More technically, the exchange is performed by using `Offer` objects, consisting of the `data` offered for exchange and a `hole` pointer. A successful swap occurs when the `hole` pointer in the `Offer` of one thread points to the `Offer` of another thread. This can be achieved in two ways: A thread that finds that the value of `g` is null can set it to its `Offer` (line 10) and wait for a partner thread to match with (`sleep` in line 11). Upon awakening, it checks whether it was paired with another thread by executing a CAS on its own `hole` (line 12). If the CAS succeeds, then a match did not occur, and setting the `hole` pointer to point to the `fail` sentinel signals that the thread is no longer interested in the exchange. If the CAS fails then some other thread has already matched the `Offer` and the exchange can complete successfully. If `g` is not null, then the thread attempts to update the `hole` field of the `Offer` pointed to by `g` from its initial null value to its own `Offer` (line 16). An additional CAS (line 17) sets `g` back to null. By doing so, it helps to remove already-matched offers from the global pointer; hence, the CAS in line 17 is unconditional.

`Exchanger` objects do not have a formal specification. This is not surprising; describing the concurrent behaviour that requires that `exchange()` succeeds only if *two* threads invoke the method simultaneously is, as we show below, impossible using the form

PODC'14, July 15–18, 2014, Paris, France.
ACM 978-1-4503-2944-6/14/07.
http://dx.doi.org/10.1145/2611462.2611513.

(E)

```
1 class Offer {
2   int data;
3   Offer hole;
4   Offer(int d){data = d; hole = null;}
5 }
6 Offer g = null;
7 Offer fail = new Offer(-1);
8 int exchange(int d){ // we assume 0 ≤ d
9   Offer n = new Offer(d);
10  if (CAS(g, null, n)){
11    sleep(50);
12    CAS(n.hole, null, fail)
13    return n.hole.data;
14  }
15  Offer cur = g;
16  bool success = CAS(cur.hole, null, n);
17  CAS(g, cur, null);
18  if (success)
19    return cur.data;
20  else return -1;
21 }
```



(SH) (CAH) (CH)

Figure 1: (E) a simplified `Exchanger`, (P) a client program, (H_1) a concurrent history, (H_2) an undesired sequential history, (H_3) a CA-history, a graphical depiction of a (SH) sequential history, a (CAH) CA-history, and a (CH) concurrent history.

of sequential specifications suggested in [4]. As a result, correctness proofs of concurrent objects that utilize `Exchanger`-like objects are not modular. For example, the proof of the HSY-stack [3] mixes reasoning about the implementation of an (`Exchanger`-like) elimination array with its particular usage by the stack.

2. CONCURRENCY-AWARE LINEARIZABILITY (CAL)

Linearizability relates an implementation of a concurrent object with a sequential specification. Both the implementation and the specification are formalized as *prefix-closed sets of histories*. A history $H = \psi_1\psi_2\ldots$ is a sequence of methods invocations and responses. Specifications are given using *sequential histories* in which every response is immediately preceded by its matching invocation. Implementations, on the other hand, allow for arbitrary interleaving of actions by different threads, as long as the subsequence of actions of every thread is sequential. Informally, a concurrent object OS_C is linearizable with respect to a specification OS_A if every history H in OS_C can be *explained* by a history S in OS_A that "looks similar" to H. The similarity is formalized by a real-time relation $H \sqsubseteq_{RT} S$, which requires S to be a permutation of H preserving the per-thread order of actions and the order of non-overlapping operations.[1]

Why it is *impossible* to provide a sequential specification for `Exchangers`? Consider the client program P shown in Figure 1(P) which uses an `Exchanger` object. Figures 1(H_1-H_3) show three histories, where an `exchange(n)` operation returning value `n'` is depicted using an interval bounded by a "`call(n)`" and a "`ret(n')`" actions. Note that histories H_1 and H_3 might occur when P executes, but H_2 cannot.

[1]For brevity, formal details, e.g., the treatment of *history completions*, are deferred to the Appendix.

History H_1 corresponds to the case where threads t_1 and t_2 exchange items 3 and 10 respectively and t_3 fails to pair-up. History H_2 is one possible sequential explanation of H_1. Using H_2 to explain H_1 raises the following problem: if H_2 is allowed by the specification then every prefix of H_2 must be allowed as-well. In particular, history H_2' in which only t_1 performs its operation should be allowed. Note that in H_2' a thread exchanges an item without finding a partner. Clearly, H_2' is an *undesired* behaviour. In fact, any sequential history that attempts to explain H_1 would allow for similar undesired behaviours. (In general, only executions in which all `exchange()` operations fail can be explained by sequential histories.) We conclude that any sequential specification of the `Exchanger` is either too restrictive or too loose.

We now turn to the definition of *concurrency-aware linearizability*. A key notion here is that of *concurrency-aware histories*. A history H is **concurrency-aware** (**CA-History**) if for any history H_1 such that $H = H_1\psi'\psi H_2$ if ψ' is a response and ψ is an invocation then the matching response of any invocation in $H_1\psi'$ is also in $H_1\psi'$. Note that a CA-history may contain concurrent operations. However, it ensures that such operations overlap pairwise. This provides the illusion that *all* concurrent operations are performed instantaneously at the same point in time. Figure 1 illustrates a sequential history (SH), a CA-history (CAH), and a concurrent history (CH). Note that every sequential history is a CA-history and every CA-history is a concurrent history, but not vice-versa. CAL extends linearizability by allowing specifications to be a (prefix-closed) set of CA-histories: A concurrent object OS_C is *CA-linearizable* with respect to a specification OS_A, if every history H in OS_C has a "similar-looking" *CA-history* S in OS_A. The "similar-looking" relation used in CAL is the same real-time order relation used to define linearizability [4]; the term *Concurrency-Aware Linearizability* emphasizes that the specification is comprised of **concurrency-aware** histories rather than a sequential ones.

Note that history H_3, depicted in Figure 1, is a *concurrency-aware history*. It describes the observable behavior as in H_1 while maintaining the same real-time order of operations and requiring that `exchange(3)` and `exchange(10)` execute concurrently and, seemingly, at the same point in time. Also note that every prefix of H_3 describes a behavior which is allowed by the implementation. Indeed, the behaviour of `Exchanger` objects can be specified precisely using CA-histories.

3. CONCLUSIONS AND FUTURE WORK

We present Concurrency-Aware Linearizabilty (CAL), a new correctness condition for an important class of CA-objects, concurrent objects whose behaviour does not have a sequential explanation. CA-objects exist in practice but currently do not have formal specifications. CAL allows providing accurate formal specifications for CA-objects using CA-histories, a restricted generalization of sequential histories. We believe that CAL can form the semantical basis for modular and reusable correctness proofs for CA-objects.

Acknowledgements. This research was supported by the EU project ADVENT and by the Broadcom Foundation and Tel Aviv University Authentication Initiative.

References

[1] Abstraction for concurrent objects. *TCS*, 411(51-52), 2010.

[2] Y. Afek, M. Hakimi, and A. Morrison. Fast and scalable rendezvousing. In *Distributed Computing*. 2011.

[3] D. Hendler, N. Shavit, and L. Yerushalmi. A scalable lock-free stack algorithm. In *SPAA*, 2004.

[4] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *TOPLAS*, 1990.

[5] W. N. Scherer III, D. Lea, and M. L. Scott. A scalable elimination-based exchange channel. *SCOOL*, 2005.

[6] W. N. Scherer III, D. Lea, and M. L. Scott. Scalable synchronous queues. In *PPoPP*, 2006.

[7] W. N. Scherer III and M. L. Scott. Nonblocking concurrent data structures with condition synchronization. In *Distributed Computing*. 2004.

APPENDIX

In this section we formalize the notion of *contentaion-aware linearizability* (CAL). We assume infinite sets of object names $o \in \mathcal{O}$, method names $f \in \mathcal{F}$, and threads identifiers $t \in \mathcal{T}$.

DEFINITION 1. *An **object action** is either an **inocation** $\psi = (t, \text{inv}\, o.f(n))$ or a **response** $\psi' = (t', \text{res}(n')o'.f')$.*

Intuitively, an invocation $\psi = (t, \text{inv}\, o.f(n))$ means transfer of control from the client to the library, and response $\psi' = (t', \text{res}(n')o'.f')$ means the return of control to the invoking client. As in [4], the observable behaviour of a concurrent object is represented by a set of histories, which are sequences of *invocations* and *responses* of methods calls.

DEFINITION 2. *A **history** H is a finite sequence of invocations and responses. We use H_i to denote the ith action of H and $H|_t$ to denote the projection of H onto actions of thread t. We denote by $\_$ an expression that is irrelevant and implicitly existentially quantified. A history is **sequential** if every response action is immediately preceded by a matching invocation. A history H is **well-formed** if invocations and responses are properly matched: for every thread t, $H|_t$ is sequential. A history is **complete** if it is well-formed and every invocation has a matching response (i.e, for every thread t, if $H|_t = \_\psi$ then ψ is a response). History H^c is a **completion** of a well-formed history H if it is complete and can be obtained from H by (possibly) extending H with some response actions and (possibly) removing some invocation actions. We denote by **complete(H)** the set of all completions of H.*

Linearizability is a relation between **object systems**, prefix-closed sets of well-formed histories. Following [4], we define it using the notion of real-time order.

DEFINITION 3. *The **real-time order** between actions of a well-formed history H is an irreflexive partial order $\prec_H$ on (indices of) object actions:*

$$H_i \prec_H H_j \iff \exists i \le i' < j' \le j.\, tid(H_i) = tid(H_{i'}) \wedge tid(H_j) = tid(H_{j'}) \wedge (tid(H_i) = tid(H_j) \vee H_{i'} = (\_, \text{res}\_) \wedge H_{j'} = (\_, \text{inv}\_))$$

*A history H **agrees** with the real-time order of a history S, denoted by $H \sqsubseteq_{RT} S$, if (i) for every thread t, $H|_t = S|_t$ and (ii) there is a bijection $\pi : \{1, \cdots, |H|\} \to \{1, \cdots, |S|\}$ such that*

$$\forall i.(H_i = S_{\pi(i)}) \wedge (\forall i, j.H_i \prec_H H_j \implies S_{\pi(i)} \prec_S S_{\pi(j)})\,.$$

Intuitively, history S "agrees" with H if in both histories every thread performs the same sequence of actions and the real-time order induced by H is a subset of that of S, i.e. $\prec_H \subseteq \prec_S$.

DEFINITION 4 (LINEARIZABILITY [4]). *Let OS_C and OS_A be object systems. We say that OS_C is **linearizable** with respect to OS_A if every history $H \in OS$ is* sequential *and*

$$\forall H \in OS_C.\, \exists H^c \in complete(H).\, \exists S \in OS_A.\, H^c \sqsubseteq_{RT} S\,.$$

We now turn on to formally define *CAL*. The key aspect is the notion of *CA-History*, which is the building block of new class of specifications which strictly extend sequential specifications. We use the notion of complete histories to provide an alternative definition for CA-histories.

DEFINITION 5 (CONCURRENCY-AWARE HISTORY). *A history H is **concurrency-aware** (**CA-History**) if for any history H_1 such that $H = H_1\psi'\psi H_2$ if ψ' is a response and ψ is an invocation then $H_1\psi'$ is a complete history. An object system is OS **concurrency-aware** if each $H \in OS$ is a concurrency-aware history.*

A concurrency-aware history allows for some operations to be executed concurrently by multiple threads. Moreover, it ensures that out of the set of threads that are operating concurrently, no thread will return before all other threads have invoked the operation (i.e. all operations must overlap pairwise). Figure 1 illustrates sequential history (SH), concurrency-aware history (CAH) and concurrent history (CH). Note that while every sequential history is CA, the opposite does not hold.

Extending Definition 4, Concurrency-aware linearizabilty of an object system is described using the $\sqsubseteq_{RT}$ relation to a concurrency-aware object system:

DEFINITION 6 (CONCURRENCY AWARE LINEARIZABILITY). *Let OS_C and OS_A be object systems. We say that OS_C is **concurrency-aware linearizable (CAL)** with respect to OS_A if*

$$\forall H \in OS_C.\, \exists H^c \in complete(H).\, \exists S \in OS_A.\, H^c \sqsubseteq_{RT} S$$

and every history $H \in OS_A$ is concurrency-aware.

Thus, CA-linearizable object is such that every interaction with it can be "explained" by a CA-history of some concurrency-aware object system OS_A.

Note that the same real-time order $\sqsubseteq_{RT}$ and notion of completions are used in the definitions of linearizability and concurrency-aware linearizability; the term concurrency-aware linearizability emphasizes that the specification is comprised of *concurrency-aware* histories, rather than a *sequential* ones.

Dr. Eran Tromer

The Blavatnik School of Computer Science
Raymond and Beverly Sackler Faculty of Exact Sciences

Paper

- **Scalable Zero Knowledge via Cycles of Elliptic Curves**

Scalable Zero Knowledge via Cycles of Elliptic Curves

(extended version)

Eli Ben-Sasson
Technion

Alessandro Chiesa
MIT

Eran Tromer
Tel Aviv University*

Madars Virza
MIT

Abstract

Non-interactive zero-knowledge proofs of knowledge for general NP statements are a powerful cryptographic primitive, both in theory and in practical applications. Recently, much research has focused on achieving an additional property, *succinctness*, requiring the proof to be very short and easy to verify. Such proof systems are known as *zero-knowledge succinct non-interactive arguments of knowledge* (zk-SNARKs), and are desired when communication is expensive, or the verifier is computationally weak.

Existing zk-SNARK implementations have severe scalability limitations, in terms of space complexity as a function of the size of the computation being proved (e.g., running time of the NP statement's decision program). First, the size of the proving key is quasilinear in the upper bound on the computation size. Second, producing a proof requires "writing down" all intermediate values of the entire computation, and then conducting global operations such as FFTs.

The bootstrapping technique of Bitansky et al. (STOC '13), following Valiant (TCC '08), offers an approach to scalability, by recursively composing proofs: proving statements about acceptance of the proof system's own verifier (and correctness of the program's latest step). Alas, recursive composition of known zk-SNARKs has never been realized in practice, due to enormous computational cost.

Using new elliptic-curve cryptographic techniques, and methods for exploiting the proof systems' field structure and nondeterminism, we achieve the first zk-SNARK implementation that practically achieves recursive proof composition. Our zk-SNARK implementation runs random-access machine programs and produces proofs of their correct execution, on today's hardware, for any program running time. It takes constant time to generate the keys that support *all* computation sizes. Subsequently, the proving process only incurs a constant multiplicative overhead compared to the original computation's time, and an essentially-constant additive overhead in memory. Thus, our zk-SNARK implementation is the first to have a well-defined, albeit low, clock rate of "verified instructions per second".

Keywords: computationally-sound proofs, proof-carrying data, zero knowledge, elliptic curves

*The Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences

Contents

1 Introduction

Non-interactive zero-knowledge proofs of knowledge [BFM88, NY90, BDSMP91] are a powerful tool, studied extensively both in theoretical and applied cryptography. Recently, much research has focused on achieving an additional property, *succinctness*, that requires the proof to be very short and easy to verify. A proof system with this additional property is called a *zero-knowledge Succinct Non-interactive ARgument of Knowledge* (zk-SNARK). Because succinctness is a desirable, sometimes critical, property in numerous security applications, prior work has investigated zk-SNARK implementations. Unfortunately, all implementations to date suffer from severe scalability limitations, due to high space complexity, as we now explain.

1.1 Scalability limitations of prior zk-SNARK implementations

Expensive preprocessing. As in any non-interactive zero-knowledge proof, a zk-SNARK requires a one-time trusted setup of public parameters: a *key generator* samples a proving key (used to generate proofs) and a verification key (used to check proofs); the key pair is then published as the proof system's parameters.

Most zk-SNARK constructions [Gro10, Lip12, BCIOP13, GGPR13, PGHR13, BCGTV13a, Lip13, BCTV14], including all published implementations [PGHR13, BCGTV13a, BCTV14], require *expensive preprocessing* during key generation. Namely, the key generator takes as input an upper bound on the computation size, e.g., in the form of an explicit NP decision circuit C output by a *circuit generator*; then, the key generator's space complexity, as well as the size of the output proving key, depends at least linearly on this upper bound. Essentially, the circuit C is explicitly laid out and encoded so as to produce the proof system's parameters.

One way to mitigate the costs of expensive preprocessing is to make C universal, i.e., design C so that it can handle more than one choice of program [BCTV14]. Yet, C *still* depends on upper bounds on the program size and number of execution steps. Moreover, even if key generation is carried out only once per circuit C, the resulting large proving key must be stored, and accessed, *each time a proof is generated*. Prior implementations of zk-SNARKs quickly become space-bound already for modest computation sizes, e.g., with proving keys of over 4 GB for circuits of only 16 million gates [BCTV14].[1]

Thus, expensive preprocessing severely limits scalability of a zk-SNARK.

Space-intensive proof generation. Related in part to the aforementioned expensive preprocessing, the prover in all published zk-SNARK implementations has large space complexity. Essentially, the proving process requires writing down the *entire* computation (e.g., the evaluation of the circuit C) all at once, and then conduct a global computation (such as Fast Fourier transforms, or multi-exponentiations) based on it. In particular, if C expresses the execution of a program, then proving requires writing down the full trace of intermediate states throughout the program execution.

Tradeoffs are possible, using block-wise versions of the global algorithms, and repeating the computation to reproduce segments of the trace. These decrease the prover's space complexity but significantly increase its time complexity, and thus do not adequately address scalability.

Remark 1.1. Even when relaxing the goal (by allowing interaction, "theorem batching", or non-zero-knowledge proofs), all published implementations of proof systems for outsourcing NP computations [SBW11, SMBW12, SVPB$^+$12, SBVB$^+$13, BFRS$^+$13] also suffer from both of the above scalability limitations.[2]

[1] Even worse, the reported numbers are for "data at rest": the proving key consists of a list of elliptic-curve points, which are *compressed* when not in use. However, when the prover uses the proving key to produce a proof, the points are uncompressed (and represented via projective or Jacobian coordinates), and take about three times as much space in memory.

[2] In contrast, when outsourcing P computations, there are implementations without expensive preprocessing: [CMT12, TRMP12, Tha13] consider low-depth circuits, and [CRR11] consider outsourcing to multiple provers at least one of which is honest.

1.2 What we know from theory

Ideally, we would like to implement a zk-SNARK that does not suffer from either of the scalability limitations mentioned in the previous section, i.e., a zk-SNARK where:

- Key generation is *cheap* (i.e., its running time only depends on the security parameter) and *suffices for all computations* (of polynomial size). Such a zk-SNARK is called **fully succinct**.
- Proof generation is carried out *incrementally*, alongside the original computation, by updating, at each step, a proof of correctness of the computation so far. Such a zk-SNARK is called **incrementally computable**.

Work in cryptography tells us that the above properties can be achieved in theoretical zk-SNARK constructions. Namely, building on the work of Valiant on incrementally-verifiable computation [Val08] and the work of Chiesa and Tromer on proof-carrying data [CT10, CT12], Bitansky et al. [BCCT13] showed how to construct zk-SNARKs that are fully-succinct and incrementally-computable.

Concretely, the approach of [BCCT13] consists of a transformation that takes as input a *preprocessing* zk-SNARK (such as one from existing implementations), and *bootstraps* it, via recursive proof composition, into a new zk-SNARK that is fully-succinct and incrementally-computable. In recursive proof composition, a prover produces a proof about an NP statement that, among other checks, also ensures the accepting computation of the proof system's own verifier. In a zk-SNARK, proof verification is asymptotically cheaper than merely verifying the corresponding NP statement; so recursive proof composition is viable, in theory. In practice, however, this step introduces concretely enormous costs: even if zk-SNARK verifiers can be executed in just a few milliseconds on a modern desktop [PGHR13, BCTV14], zk-SNARK verifiers still take millions of machine cycles to execute. Hence, known zk-SNARK implementations cannot achieve *even one step* of recursive proof composition in practical time. Thus, whether recursive proof composition can be realized in practice, with any reasonable efficiency, has so far remained an intriguing open question.

Remark 1.2 (PCPs)**.** Suitably instantiating Micali's "computationally-sound proofs" [Mic00] yields fully-succinct zk-SNARKs. However, it is not known how to also achieve incremental computation with this approach (without also invoking the aforementioned approach of Bitansky et al. [BCCT13]). Indeed, [Mic00] requires probabilistically-checkable proofs (PCPs) [BFLS91], where one can achieve a prover that runs in quasilinear-time [BCGT13b], but only by requiring space-intensive computations — again due to the need to write down the entire computation and conducting global operations on it.

1.3 Contributions

We present the first prototype implementation that practically achieves recursive composition of zk-SNARKs. This enables us to achieve the following results:

(i) Scalable zk-SNARKs. We present the first implementation of a zk-SNARK that is fully succinct and incrementally computable. Our implementation follows the approach of Bitansky et al. [BCCT13].

Our zk-SNARK works for proving/verifying computations on a general notion of random-access machine. The key generator takes as input a *machine specification*, consisting of settings for random-access memory (number of addresses and number of bits at each address) and a CPU circuit, defining the machine's behavior. The keys sampled by the key generator support proving/verifying computations, of any polynomial length, on this machine. Thus, our zk-SNARK implementation directly supports many architectures (e.g., floating-point processors, SIMD-based processors, etc.) — one only needs to specify memory settings and a CPU circuit.

Compared to the original machine computation, our zk-SNARK only imposes a constant multiplicative overhead in time and an essentially-constant additive overhead in space. Indeed, the proving process steps through the machine's computation, each time producing a new proof that the computation is correct so far, by relying on the prior proof; each proof asserts the satisfiability of a constant-size circuit, and requires few resources in time and space to produce. Our zk-SNARK scales, on today's hardware, to any computation size.

(ii) Proof-carrying data. The main tool in [BCCT13]'s approach is *proof-carrying data* (PCD) [CT10, CT12], a cryptographic primitive that encapsulates the security guarantees provided by recursive proof composition. Thus, as a stepping stone towards the aforementioned zk-SNARK implementation, we also achieve the first implementation of PCD, for arithmetic circuits.

(iii) Evaluation on vnTinyRAM. We evaluate our zk-SNARK on a specific choice of random-access machine: vnTinyRAM, a simple RISC von Neumann architecture that is supported by the most recent preprocessing zk-SNARK implementation [BCTV14]. The evaluation confirms our expectations that our approach is slower for small computations but achieves scalability to large computations.

We evaluated our prototype on 16-bit and 32-bit vnTinyRAM with 16 registers (as in [BCTV14]). For instance, for 32-bit vnTinyRAM, our prototype incrementally proves correct program execution at the cost of 26.2 seconds per program step, using a 55 MB proving key and 993 MB of additional memory. In contrast, for a T-step program, the system of [BCTV14] requires roughly $0.05 \cdot T$ seconds, *provided* that roughly $3.1 \cdot T$ MB of main memory are available. Thus for $T > 321$ our system is more space-efficient, and the savings in space continue to grow as T increases. (These numbers are for an 80-bit security level.)

The road ahead. Obtaining scalable zk-SNARKs is but one application of PCD. More generally, PCD enables efficient "distributed theorem proving", which has applications ranging from securing the IT supply chain, to information flow control, and to distributed programming-language semantics [CT10, CT12, CTV13]. Now that a first prototype of PCD has been achieved, these applications are waiting to be explored in practice.

Remark 1.3 (parametrization)**.** In this work we describe a concrete implementation of a cryptographic system, whose efficiency scales with the security parameter and other quantities (e.g., wordsize of a machine, size of random-access memory, and so on). Since we make several concrete choices (e.g., fixing the security level at 80 bits, fixing vnTinyRAM's wordsize to 16 or 32 bits as in [BCTV14]) many asymptotic dependencies "collapse" to constants. We focus on scalability as a function of the computation size, i.e., the number of steps and amount of memory in the original program's execution on the concrete random-access machine.

1.4 Summary of challenges and techniques

As we recall in Section 2, bootstrapping zk-SNARKs involves two main ingredients: a collision-resistant hash function and a preprocessing zk-SNARK. Practical implementations of both ingredients exist. So one may conclude that "practical bootstrapping" is merely a matter of stitching together implementations of these two ingredients. As we now explain, this conclusion is mistaken, because bootstrapping a zk-SNARK in practice poses several challenges that must be tackled in order to obtain any reasonable efficiency.

Common theme: leverage field structure. The techniques that we employ to overcome efficiency barriers leverage the fact that the "native" NP language whose membership is proved/verified by the zk-SNARK is the satisfiability of $\mathbb{F}$-arithmetic circuits, for a certain finite field $\mathbb{F}$. While any NP statement can be reduced to $\mathbb{F}$-arithmetic circuits, the proof system is most efficient for statements expressible as $\mathbb{F}$-arithmetic circuits of small size. Prior work only partially leveraged this fact, by using circuits that conduct large-integer arithmetic or "pack" bits into field elements for non-bitwise checks (e.g., equality) [PGHR13, BCGTV13a, BFRS$^+$13, BCTV14]. In this paper, we go further and, for improved efficiency, use circuits that conduct *field operations*.

1.4.1 Challenge: how to efficiently "close the loop"?

By far the most prominent challenge is efficiently "closing the loop". In the bootstrapping approach, each step requires proving a statement that (i) verifies the validity of previous zk-SNARK proofs; and (ii) checks another execution step. For recursive composition, this statement needs to be expressed as an $\mathbb{F}$-arithmetic circuit C_{pcd}, so that it can be proved using the very same zk-SNARK. In particular, we need to *implement the verifier V as an $\mathbb{F}$-arithmetic circuit C_V* (a subcircuit of C_{pcd}).

In principle, constructing C_V is possible, because circuits are a universal model of computation. And not just in principle: much research has been devoted to improve the efficiency and functionality of circuit generators in practice [SVPB+12, BCGT13a, SBVB+13, PGHR13, BCGTV13a, BCTV14]. Hence, a reasonable approach to construct C_V is to apply a suitable circuit generator to a suitable software implementation of V.

However, such an approach is likely to be inefficient. Circuit generators strive to support complex program computations, by providing ways to efficiently handle data-dependent control flow, memory accesses, and so on. Instead, verifiers in preprocessing zk-SNARK constructions are "circuit-like" programs, consisting of few pairing-based arithmetic checks that do not use complex data-dependent control flow or memory accesses.

Thus, we want to avoid circuit generators, and somehow directly construct C_V so that its size is not huge. As we shall explain (see Section 3), this is not merely a programmatic difficulty, but there are *mathematical obstructions* to constructing C_V efficiently.

Main technique: PCD-friendly cycles of elliptic curves. In our underlying preprocessing zk-SNARK, the verifier V consists mainly of operations in an elliptic curve over a field $\mathbb{F}'$, and is thus expressed, most efficiently, as a $\mathbb{F}'$-arithmetic circuit. We observe that if this field $\mathbb{F}'$ is the same as the aforementioned native field $\mathbb{F}$ of the zk-SNARK's statement, then recursive composition can be orders of magnitude more efficient than otherwise. Unfortunately, as we shall explain, the "field matching" $\mathbb{F} = \mathbb{F}'$ is mathematically impossible.

In contrast, we show how to circumvent this obstruction by using multiple, suitably-chosen elliptic curves, that lie on a *PCD-friendly cycle*. For example, a PCD-friendly 2-cycle consists of two curves such that the (prime) size of the base field of one curve equals the group order of the other curve, and vice versa. Our implementation uses a PCD-friendly cycle of elliptic curves (found at a great computational expense) to attain zk-SNARKs that are *tailored* for recursive proof composition.

Additional technique: nondeterministic verification of pairings. The zk-SNARK verifier involves, more specifically, several pairing-based checks over its elliptic curve. Yet, each pairing evaluation is very expensive, if not carefully performed. To further improve efficiency, we exploit the fact that the zk-SNARK supports NP statements, and provide a hand-optimized circuit implementation of the zk-SNARK verifier that leverages nondeterminism for improved efficiency. For instance, in our construction, we make heavy use of *affine* coordinates for both curve arithmetic and divisor evaluations [LMN10], because these are particularly efficient to *verify* (as opposed to *computing*, for which projective or Jacobian coordinates are known to be faster).

1.4.2 Challenge: how to efficiently verify collision-resistant hashing?

Bootstrapping zk-SNARKs uses, at multiple places, a collision-resistant hash function H and an arithmetic circuit C_H for verifying computations of H. If not performed efficiently, this would be another bottleneck.

For instance, the aforementioned circuit C_{pcd}, besides verifying prior zk-SNARK proofs, is also tasked with verifying one step of machine execution. This involves not only checking the CPU execution but also the validity of loads and stores to random-access memory, done via memory-checking techniques based on Merkle trees [BEGKN91, BCGT13a]. Thus C_{pcd} also needs to have a subcircuit to check Merkle-tree authentication paths. Constructing such circuits is straightforward, given a circuit C_H for verifying computations of H. But the main question here is how to pick H so that C_H can be small. Indeed, if random-access memory consists of A addresses, then checking an authentication path requires at least $\lceil \log A \rceil \cdot |C_H|$ gates. If C_H is large, this subcircuit *dwarfs* the CPU, and "wastes" most of the size of C_{pcd} for a single load/store.

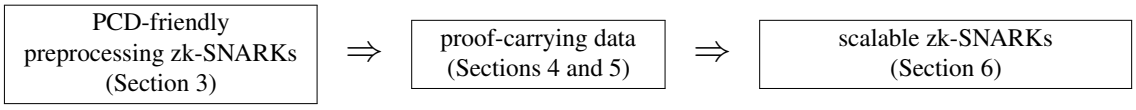
Merely picking some standard choice of hash function H (e.g., SHA-256 or Keccak) yields C_H with tens of thousands of gates [PGHR13, BCGG+14], making hash verifications very expensive. Is this inherent?

Additional technique: field-specific hashes. We select a hash H that is tailored to efficient verification in the field $\mathbb{F}$. In our setting, $\mathbb{F}$ has prime order p, so its additive group is isomorphic to $\mathbb{Z}_p$. Thus, a natural approach is to let H be a *modular subset-sum* function over $\mathbb{Z}_p$. For suitable parameter choices and for random coefficients, subset-sum functions are collision-resistant [Ajt96, GGH96]. In this paper we base all

of our collision-resistant hashing on suitable subset sums, and thereby greatly reduce the burden of hashing.[3]

1.5 Roadmap

The rest of this paper is organized as follows. In Section 2 we recall the main ideas of [BCCT13]'s approach. Then we discuss our construction in more detail, in the following three steps:



In Section 7, we evaluate our system on the machine vnTinyRAM. In Section 8, we discuss open problems.

2 Preliminaries

We give here the essential definitions needed for the technical discussions in the body of the paper; more detailed definitions can be found in the appendices (where some definitions are taken verbatim from [BCTV14]).

We denote by $\mathbb{F}$ a field, and by $\mathbb{F}_n$ the field of size n. Throughout, we assume familiarity with finite fields; for background on these, see the book of Lidl and Niederreiter [LN97].

2.1 Preprocessing zk-SNARKs for arithmetic circuits

Given a field $\mathbb{F}$, the *circuit satisfaction problem* of an $\mathbb{F}$-arithmetic circuit $C\colon \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ is defined by the relation $\mathcal{R}_C = \{(x,a) \in \mathbb{F}^n \times \mathbb{F}^h : C(x,a) = 0^l\}$; its language is $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists\, a \in \mathbb{F}^h,\ C(x,a) = 0^l\}$.

A **preprocessing zk-SNARK** for $\mathbb{F}$-arithmetic circuit satisfiability (see, e.g., [BCIOP13]) is a triple of polynomial-time algorithms (G, P, V), called *key generator*, *prover*, and *verifier*. The key generator G, given a security parameter λ and an $\mathbb{F}$-arithmetic circuit $C\colon \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$, samples a *proving key* pk and a *verification key* vk; these are the proof system's public parameters, which need to be generated only once per circuit. After that, anyone can use pk to generate non-interactive proofs for the language $\mathcal{L}_C$, and anyone can use the vk to check these proofs. Namely, given pk and any $(x,a) \in \mathcal{R}_C$, the honest prover $P(\mathsf{pk}, x, a)$ produces a proof π attesting that $x \in \mathcal{L}_C$; the verifier $V(\mathsf{vk}, x, \pi)$ checks that π is a valid proof for $x \in \mathcal{L}_C$. A proof π is a proof of knowledge, as well as a (statistical) zero-knowledge proof. The succinctness property requires that π has length $O_\lambda(1)$ and V runs in time $O_\lambda(|x|)$, where O_λ hides a (fixed) polynomial in λ.

See Appendix C for details.

2.2 Proof-carrying data

Proof-carrying data (PCD) [CT10, CT12] is a cryptographic primitive that encapsulates the security guarantees obtainable via recursive composition of proofs. Since recursive proof composition naturally involves multiple (physical or virtual) parties, PCD is phrased in the language of a dynamically-evolving *distributed computation* among mutually-untrusting computing nodes, who perform local computations, based on local data and previous messages, and then produce output messages. Given a *compliance predicate* Π to express local checks, the goal of PCD is to ensure that any given message z in the distributed computation is Π-*compliant*, i.e., is consistent with a history in which each node's local computation satisfies Π. This formulation includes as special cases incrementally-verifiable computation [Val08] and targeted malleability [BSW12].

Concretely, a proof-carrying data (PCD) system is a triple of polynomial-time algorithms $(\mathbb{G}, \mathbb{P}, \mathbb{V})$, called *key generator*, *prover*, and *verifier*. The key generator $\mathbb{G}$ is given as input a predicate Π (specified as an arithmetic circuit), and outputs a proving key pk and a verification key vk; these keys allow anyone to

[3] We note that subset-sum functions were also used in [BFRS+13], but, crucially, they were *not* tailored to the field. This is a key difference in usage and efficiency. (E.g., our hash function can be verified in ≤ 300 gates, while [BFRS+13] report 13,000.)

prove/verify that a piece of data z is Π-compliant. This is achieved by attaching a short and easy-to-verify proof to each piece of data. Namely, given pk, received messages $\vec{z}_{\mathsf{in}}$ with proofs $\vec{\pi}_{\mathsf{in}}$, local data z_{loc}, and a claimed outgoing message z, $\mathbb{P}$ computes a new proof π to attach to z, which attests that z is Π-compliant; the verifier $\mathbb{V}(\mathsf{vk}, z, \pi)$ verifies that z is Π-compliant. A proof π is a proof of knowledge, as well as a (statistical) zero-knowledge proof; succinctness requires that π has length $O_\lambda(1)$ and $\mathbb{V}$ runs in time $O_\lambda(|z|)$.

Finally, note that since Π is expressed as an $\mathbb{F}$-arithmetic circuit for a given field $\mathbb{F}$, the size of messages and local data are fixed; we denote these sizes by $n_{\mathsf{msg}}, n_{\mathsf{loc}} \in \mathbb{N}$. Similarly, the number of input messages is also fixed; we call this the *arity*, and denote it by $s \in \mathbb{N}$. Moreover, for convenience, Π also takes as input a flag $b_{\mathsf{base}} \in \{0, 1\}$ denoting whether the node has no predecessors (i.e., b_{base} is a "base-case" flag). Overall, Π takes an input $(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, b_{\mathsf{base}}) \in \mathbb{F}^{n_{\mathsf{msg}}} \times \mathbb{F}^{n_{\mathsf{loc}}} \times \mathbb{F}^{s \cdot n_{\mathsf{msg}}} \times \mathbb{F}$.

See Appendix D for details.

2.3 The bootstrapping approach

Our implementation follows [BCCT13], which we now review. The approach consists of a transformation that, on input a preprocessing zk-SNARK and a collision-resistant hash function, outputs a scalable zk-SNARK. Thus, the input zk-SNARK is *bootstrapped* into one with improved scalability properties.

So fix a preprocessing zk-SNARK (G, P, V) and collision-resistant function H. The goal is to construct a fully-succinct incrementally-computable zk-SNARK $(G^\star, P^\star, V^\star)$ for proving/verifying the correct execution on a given random-access machine $\mathbf{M}$. Informally, we describe the transformation in four steps.

Step 1: from zk-SNARKs to PCD. The first step, independent of $\mathbf{M}$, is to construct a PCD system $(\mathbb{G}, \mathbb{P}, \mathbb{V})$, by using the zk-SNARK (G, P, V). This step involves recursive composition of zk-SNARK proofs.

Step 2: delegate the machine's memory. The second step is to reduce the footprint of the machine $\mathbf{M}$, by delegating its random-access memory to an untrusted storage, via standard memory-checking techniques based on Merkle trees [BEGKN91, BCGT13a]. We thus modify $\mathbf{M}$ so that its "CPU" receives values loaded from memory as nondeterministic guesses, along with corresponding authentication paths that are checked against the root of a Merkle tree based on the hash function H. Thus, the entire state of $\mathbf{M}$ only consists of a (short) CPU state, and a (short) root of the Merkle tree that "summarizes" memory.[4]

Step 3: design a predicate $\Pi_{\mathbf{M},H}$ for step-wise verification. The third step is to design a compliance predicate $\Pi_{\mathbf{M},H}$ that ensures that the only $\Pi_{\mathbf{M},H}$-compliant messages z are the ones that result from the correct execution of the (modified) machine $\mathbf{M}$, one step at a time; this is analogous to the notion of incremental computation [Val08]. Crucially, because $\Pi_{\mathbf{M},H}$ is only asked to verify one step of execution at a time, we can implement $\Pi_{\mathbf{M},H}$'s requisite checks with a circuit of merely constant size.

Step 4: construct new proof system. The new zk-SNARK $(G^\star, P^\star, V^\star)$ is constructed as follows. The new key generator $G^\star$ is set to the PCD generator $\mathbb{G}$ invoked on $\Pi_{\mathbf{M},H}$. The new prover $P^\star$ uses the PCD prover $\mathbb{P}$ to prove correct execution of $\mathbf{M}$, one step at a time and conducting the incremental distributed computation "in his head". The new verifier $V^\star$ simply uses the PCD verifier $\mathbb{V}$ to verify $\Pi_{\mathbf{M},H}$-compliance. In sum, since $\Pi_{\mathbf{M},H}$ is small and suffices for all computations, the new zk-SNARK is scalable: it is fully succinct; moreover, because the new prover computes a proof for each new step based on the previous one, it is also incrementally computable. (See Appendix E for definitions of these properties.)

Our goal is to realize the above approach in a practical implementation.

Security of recursive proof composition. Security in [BCCT13] is proved by using the *proof-of-knowledge property* of zk-SNARKs; we refer the interested reader to [BCCT13] for details. One aspect that must be addressed from a theoretical standpoint is the *depth* of composition. Depending on assumption strength, one

[4] Similarly to [BCCT13] and our realization thereof, Braun et al. [BFRS+13] leverage memory-checking techniques based on Merkle trees [BEGKN91] for the purpose of enabling a circuit to "securely" load from and store to an untrusted storage. However, the systems' goals (delegation of MapReduce computations via a 2-move protocol) and techniques are different (cf. Footnote 3).

may have to recursively compose proofs in "proof trees above the message chain", rather than along the chain. From a practical perspective we make the heuristic assumption that depth of composition does not affect security of the zk-SNARK, because no evidence suggests otherwise for the constructions that we use.

3 PCD-friendly preprocessing zk-SNARKs

We first construct preprocessing zk-SNARKs that are tailored for efficient recursive composition of proofs. Later, in Section 5, we discuss how we use such zk-SNARKs to construct a PCD system.

3.1 PCD-friendly cycles of elliptic curves

Let $\mathbb{F}$ be a finite field, and (G, P, V) a preprocessing zk-SNARK for $\mathbb{F}$-arithmetic satisfiability. The idea of recursive proof composition is to prove/verify satisfiability of an $\mathbb{F}$-arithmetic circuit C_{pcd} that checks the validity of previous proofs (among other things). Thus, we need to implement the verifier V as an $\mathbb{F}$-arithmetic circuit C_V, to be used as a sub-circuit of C_{pcd}.

How to write C_V depends on the algorithm of V, which in turn depends on which elliptic curve is used to instantiate the pairing-based zk-SNARK. For prime r, in order to prove statements about $\mathbb{F}_r$-arithmetic circuit satisfiability, one instantiates (G, P, V) using an elliptic curve E defined over some finite field $\mathbb{F}_q$, where the group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points has order $r = \#E(\mathbb{F}_q)$ (or, more generally, r divides $\#E(\mathbb{F}_q)$). Then, all of V's arithmetic computations are over $\mathbb{F}_q$, or extensions of $\mathbb{F}_q$ up to degree k, where k is the embedding degree of E with respect to r (i.e., the smallest integer k such that r divides $q^k - 1$). (See Appendix C.2.)

We motivate our approach by first describing two "failed attempts".

Attempt #1: pick curve with $q = r$. Ideally, we would like to select a curve E with $q = r$, so that V's arithmetic is over the *same field* for which V's native NP language is defined. Unfortunately, this cannot happen: the condition that E has embedding degree k with respect to r implies that r divides $q^k - 1$, which implies that $q \neq r$. The same implication holds even if $E(\mathbb{F}_q)$ has a non-prime order n and the prime r (with respect to which k is defined) only divides n. So, while appealing, this idea cannot even be instantiated.[5]

Attempt #2: long arithmetic. Since we are stuck with $q \neq r$, we may consider doing "long arithmetic": simulating $\mathbb{F}_q$ operations via $\mathbb{F}_r$ operations, by working with bit chunks to perform integer arithmetic, and modding out by q when needed. Alas, having to work at the "bit level" implies a blowup on the order of $\log q$ compared to native arithmetic. So, while this approach can at least be instantiated, it is very expensive.[6]

Our approach: cycle through multiple curves. We formulate, and instantiate, a new property for elliptic curves that enables us to completely circumvent long arithmetic, even with $q \neq r$. In short, our idea is to base recursive proof composition, not on a single zk-SNARK, but on *multiple* zk-SNARKs, each instantiated on a different elliptic curve, that *jointly* satisfy a special property.

For the simplest case, suppose we have two primes q_α and q_β, and elliptic curves $E_\alpha/\mathbb{F}_{q_\alpha}$ and $E_\beta/\mathbb{F}_{q_\beta}$ such that $q_\alpha = \#E_\beta(\mathbb{F}_{q_\beta})$ and $q_\beta = \#E_\alpha(\mathbb{F}_{q_\alpha})$, i.e., the size of the base field of one curve equals the group order of the other curve, and vice versa. We then construct two preprocessing zk-SNARKs $(G_\alpha, P_\alpha, V_\alpha)$ and $(G_\beta, P_\beta, V_\beta)$, respectively instantiated on the two curves $E_\alpha/\mathbb{F}_{q_\alpha}$ and $E_\beta/\mathbb{F}_{q_\beta}$.

Now note that $(G_\alpha, P_\alpha, V_\alpha)$ works for $\mathbb{F}_{q_\beta}$-arithmetic circuit satisfiability, but all of V_α's arithmetic computations are over $\mathbb{F}_{q_\alpha}$ (or extensions thereof); while $(G_\beta, P_\beta, V_\beta)$ works for $\mathbb{F}_{q_\alpha}$-arithmetic circuits, but V_β's arithmetic computations are over $\mathbb{F}_{q_\beta}$ (or extensions thereof). Instead of having each zk-SNARK handle statements about its *own* verifier, as in the prior attempts (i.e., writing V_α as a $\mathbb{F}_{q_\beta}$-arithmetic circuit, or V_β as

[5] Besides, the condition $q = \#E(\mathbb{F}_q)$ is undesirable even when not ruled out (e.g., when $k = \infty$): on such curves, known as *anomalous*, discrete logarithms can be computed in polynomial time via the SSSA attack [Sem98, Sma99, SA98].

[6] Showing that it is *provably* expensive requires stronger circuit lower bounds than currently known [Raz87, Smo87, GLS09].

a $\mathbb{F}_{q_\alpha}$-arithmetic circuit), we instead let each zk-SNARK handle statements about the verifier of the *other* zk-SNARK. That is, we write V_α as a $\mathbb{F}_{q_\alpha}$-arithmetic circuit C_{V_α}, and V_β as a $\mathbb{F}_{q_\beta}$-arithmetic circuit C_{V_β}.

We can then perform recursive proof composition by *alternating* between the two proof systems. Roughly, one can use P_α to prove successful verification of a proof by C_{V_β} and, conversely, P_β to prove successful verification of a proof by C_{V_α}. Doing so in alternation ensures that fields "match up", and no long arithmetic is needed. (This sketch omits key technical details; see Section 4.)

Since E_α and E_β facilitate constructing PCD, we say that (E_α, E_β) is a *PCD-friendly 2-cycle of elliptic curves*. More generally, the idea extends to cycling through ℓ curves satisfying this definition:

Definition 3.1. *Let $E_0, \ldots, E_{\ell-1}$ be elliptic curves, respectively defined over finite fields $\mathbb{F}_{q_0}, \ldots, \mathbb{F}_{q_{\ell-1}}$, with each q_i a prime. We say that $(E_0, \ldots, E_{\ell-1})$ is a* **PCD-friendly cycle** *of length ℓ if each E_i is pairing friendly and, moreover, $\forall\, i \in \{0, \ldots, \ell-1\},\ q_i = \#E_{i+1 \bmod \ell}(\mathbb{F}_{q_{i+1 \bmod \ell}})$.*

To our knowledge this notion has not been explicitly sought before.[7] Though, fortunately, a family that satisfies this notion is already known, as discussed in the next subsection.

Remark 3.2 (relaxation)**.** One can relax Definition 3.1 to require a weaker, but still useful, condition: for each $i \in \{0, \ldots, \ell-1\}$, q_i divides $\#E(\mathbb{F}_{i+1 \bmod \ell})$. Even if weaker, this condition is *still* very strong. For instance, it implies that each curve E_i has ρ-value ≈ 1, i.e., that each E_i has near-prime order.[8] Constructing pairing-friendly curves with such good ρ-values is challenging even without the cycle condition!

Hence, generic methods such as Cocks–Pinch [CP01] and Dupont–Enge–Morain [DEM05], which yield (with high probability) curves with ρ-values > 1 (specifically, ≈ 2), cannot be used to construct PCD-friendly cycles.[9] This also applies to generalizations of the Cocks–Pinch method [BLS03, BW05, SB06] that improve the ρ-value to be $1 < \rho < 2$. In this work we do not investigate the above relaxation because, we can fulfill Definition 3.1 with $\ell = 2$, the minimal length possible.

3.2 Two-cycles based on MNT curves

We construct pairs of elliptic curves, E_4 and E_6, that form PCD-friendly 2-cycles (E_4, E_6). These are MNT curves [MNT01] of embedding degrees 4 and 6. Our construction also ensures that E_4 and E_6 are sufficiently *2-adic* (see below), a desirable property for efficient implementations of preprocessing zk-SNARKs.

MNT curves and the KT correspondence. Miyaji, Nakabayashi, and Takano [MNT01] characterized prime-order elliptic curves with embedding degrees $k = 3, 4, 6$; such curves are now known as *MNT curves*. Given an elliptic curve E defined over a prime field $\mathbb{F}_q$, they gave necessary and sufficient conditions on the pair (q, t), where t is the *trace* of E over $\mathbb{F}_q$, for E to have embedding degree $k = 3, 4, 6$. We refer to an MNT curve with embedding degree k as an MNTk curve. Karabina and Teske [KT08] proved an explicit 1-to-1 correspondence between MNT4 and MNT6 curves:

Theorem 3.3 ([KT08])**.** *Let $r, q > 64$ be primes. Then the following two conditions are equivalent:*

1. *r and q represent an elliptic curve $E_4/\mathbb{F}_q$ with embedding degree $k = 4$ and $r = \#E(\mathbb{F}_q)$;*
2. *r and q represent an elliptic curve $E_6/\mathbb{F}_r$ with embedding degree $k = 6$ and $q = \#E(\mathbb{F}_r)$.*

[7] Definition 3.1 is reminiscent, but different from, the notion of an *aliquot cycle* of elliptic curves by Silverman and Stange [SS11]. An aliquot cycle considers a *single* curve (over $\mathbb{Q}$) reduced at ℓ primes, rather than ℓ curves, and does not require pairing-friendliness.

[8] For each $i \in \{0, \ldots, \ell-1\}$, the condition that q_{i-1} divides $\#E(\mathbb{F}_i)$ implies, via the Hasse bound, that $h_i q_{i-1} \leq q_i \cdot (1 + 2/\sqrt{q_i})$ for some cofactor $h_i \in \mathbb{N}$; hence $\log q_{i-1} \leq \log q_i + \log(1 + 2/\sqrt{q_i}) - \log h_i$, and thus $\frac{\log q_{i-1}}{\log q_i} \leq 1 + \frac{\log(1+2/\sqrt{q_i})}{\log q_i} \leq 1 + \frac{2}{\sqrt{q_i} \log q_i} = 1 + a_i$ for $a_i := \frac{2}{\sqrt{q_i} \log q_i}$. Note that each a_i is exponentially small. Therefore, for each $i \in \{0, \ldots, \ell-1\}$, we can upper bound the ρ-value of E_i, equal to $\frac{\log q_i}{\log q_{i-1}}$, as follows: $\frac{\log q_i}{\log q_{i-1}} = \prod_{j \neq i} \frac{\log q_{j-1}}{\log q_j} \leq \prod_{j \neq i} (1 + a_j) = 1 + \sum_{j \neq i} a_j + \prod_{j \neq i} a_j$. In sum, the ρ-value of E_i is upper bounded by $1 + \epsilon_i$, where the quantity $\epsilon_i := \sum_{j \neq i} a_j + \prod_{j \neq i} a_j$ is exponentially small.

[9] At best, such methods can be used to construct PCD-friendly "chains", which can be used to reduce the space complexity of preprocessing zk-SNARKs via a limited application of recursive proof composition. But the large ρ-values would imply that each recursive composition roughly doubles the cost of the zk-SNARK so that long chains do not seem to be advantageous.

PCD-friendly 2-cycles on MNT curves. The above theorem implies that:

Each MNT6 curve lies on a PCD-friendly 2-cycle with the corresponding MNT4 curve (and vice versa).

Thus, a PCD-friendly 2-cycle can be obtained by constructing an MNT4 curve and its corresponding MNT6 curve. Next, we explain at high level how this can be done.

Constructing PCD-friendly 2-cycles. First, we recall the only known method to construct MNTk curves [MNT01]. It consists of two steps:

- Step I: *curve discovery*. Find suitable $(q,t) \in \mathbb{N}^2$ such that there exists an ordinary elliptic curve $E/\mathbb{F}_q$ of prime order $r := q+1-t$ and embedding degree k.
- Step II: *curve construction*. Starting from (q,t), use the *Complex-Multiplication method* (CM method) [AM93] to compute the equation of E over $\mathbb{F}_q$.

The complexity of Step II depends on the *discriminant* D of E, which is the square-free part of $4q-t^2$. At present, the CM method is feasible for discriminants D up to size 10^{16} [Sut12]. Thus, Step I is conducted in a way that results in candidate parameters (q,t) inducing relatively-small discriminants, to aid Step II. (Instead, "most" (q,t) induce a discriminant D of size $\sqrt{q}$, which is too large to handle.) Concretely, [MNT01] derived, for $k \in \{3,4,6\}$ and discriminant D, Pell-type equations whose solutions yield candidate parameters (q,t) for MNTk curves $E/\mathbb{F}_q$ of trace t and discriminant D. So Step I can be performed by iteratively solving the *MNTk Pell-type equation*, for increasing discriminant size, until a suitable (q,t) is found.

The above strategy can be extended, in a straightforward way, to construct PCD-friendly 2-cycles. First perform Step I to obtain suitable parameters (q_4,t_4) for an MNT4 curve $E_4/\mathbb{F}_{q_4}$; the parameters (q_6,t_6) for the corresponding MNT6 curve $E_6/\mathbb{F}_{q_6}$ are $q_6 := q_4+1-t_4$ and $t_6 := 2-t_4$. Then perform Step II for (q_4,t_4) to compute the equation of E_4, and then also for (q_6,t_6) to compute that of E_6. The complexity in both cases is the same: one can verify that E_4 and E_6 have the same discriminant. The two curves E_4 and E_6 form a PCD-friendly 2-cycle (E_4,E_6).

Suitable cycle parameters. We now explain what "suitable (q_4,t_4)" means in our context, by specifying a list of additional properties that we wish a PCD-friendly cycle to satisfy.

- *Bit lengths.* In a 2-cycle (E_4,E_6), the curve E_4 is "less secure" than E_6, because E_4 has embedding degree 4 while E_6 has embedding degree 6. Thus, we use E_4 to set lower bounds on bit lengths. Since we aim at a security level of 80 bits, we need $r_4 \geq 2^{160}$ and $q_4 \geq 2^{240}$ (so that $\sqrt{r_4} \geq 2^{80}$ and $q_4^4 \geq 2^{960}$ [FST10]). Since $\log r_4 \approx \log q_4$ for MNT4 curves, we only need to ensure that q_4 has at least 240 bits.[10]

- *Towering friendliness.* We restrict our focus to moduli q_4 and q_6 that are *towering friendly* (i.e., congruent to 1 modulo 6) [BS10]; this improves the efficiency of arithmetic in $\mathbb{F}_{q_4}^4$ and $\mathbb{F}_{q_6}^6$ (and their subfields).

- *2-adicity.* As discussed in [BCGTV13a, BCTV14], if a pairing-based preprocessing zk-SNARK (G,P,V) is instantiated with an elliptic curve $E/\mathbb{F}_q$ of prime order r (or with $\#E(\mathbb{F}_q)$ divisible by a prime r), it is important, for efficiency reasons, that $r-1$ is divisible by a large power of 2, i.e., $\nu_2(r-1)$ is large. (Recall that $\nu_2(n)$, the 2-adic order of n, is the largest power of 2 dividing n.) Concretely, if G is invoked on an $\mathbb{F}_r$-arithmetic circuit C, it is important that $\nu_2(r-1) \geq \lceil \log |C| \rceil$. We call $\nu_2(r-1)$ the *2-adic order of E*, or the *2-adicity of E*. (See Appendix C.2 for more details.)

  So let ℓ_4 and ℓ_6 be the target values for $\nu_2(r_4-1)$ and $\nu_2(r_6-1)$. One can verify that, for any MNT-based PCD-friendly 2-cycle (E_4,E_6), it holds that $\nu_2(r_4-1) = 2 \cdot \nu_2(r_6-1)$; in other words, E_4 is always "twice as 2-adic" as E_6. Thus, to achieve the target 2-adic orders, it suffices to ensure that $\nu_2(r_4-1) \geq \max\{\ell_4, 2\ell_6\}$ (where, as before, $r_4 := q_4+1-t_4$). As we shall see (in Section 5), in this paper it will suffice to take $\nu_2(r_4-1) \geq 34$.

[10] Alas, since E_4 has a low embedding degree, the ECDLP in $E(\mathbb{F}_{q_4})$ and DLP in $\mathbb{F}_{q_4}^4$ are "unbalanced": the former provides 120 bits of security, while the latter only 80. Moreover, the same is true for E_6: the ECDLP in $E(\mathbb{F}_{q_6})$ provides 120 bits of security, while the DLP in $\mathbb{F}_{q_4}^6$ only 80. Finding PCD-friendly cycles without these inefficiencies is an open problem (see Section 8).

Of the above properties, the most restrictive one is 2-adicity, because it requires seeing enough curves until, "by sheer statistics", one finds (q_4, t_4) with a high-enough value for $\nu_2(r_4 - 1)$. Collecting enough samples is costly because, as discriminant size increases, the density of MNT curves decreases: empirically, one finds that the number MNT curves with discriminant $D \leq N$ is (approximately) less than $\sqrt{N}$ [KT08].

An extensive computation for a suitable cycle. Overall, finding and constructing a suitable cycle required a substantial computational effort.

- *Cycle discovery.* In order to find suitable parameters for a cycle, we explored a large space: all discriminants up to $1.1 \cdot 10^{15}$, requiring about 610,000 core-hours on a large cluster of modern x86 servers. Our search algorithm is a modification of [KT08, Algorithm 3]. Among all the 2-cycles that we found, we selected parameters (q_4, t_4) and (q_6, t_6) for a 2-cycle (E_4, E_6) of curves such that: (i) q_4, q_6 each have 298 bits; (ii) q_4, q_6 are towering friendly; and (iii) $\nu_2(r_4 - 1) = 34$ and $\nu_2(r_6 - 1) = 17$. The bit length of q_4, q_6 is higher than the lower bound of 240; we entail this cost so to pick a rare cycle with high 2-adicity, which helps the zk-SNARK's efficiency more than the slowdown incurred by the higher bit length.
- *Cycle construction.* Both E_4 and E_6 have discriminant 614144978799019, whose size requires state-of-the-art techniques in the CM method [Sut11, ES10, Sut12] in order to explicitly construct the curves.[11]

Below, we report the parameters and equations for the 2-cycle (E_4, E_6) that we selected.

$E_4/\mathbb{F}_{q_4} : y^2 = x^3 + A_4 x + B_4$ where
$A_4 = 2$,
$B_4 = 42389453652668417828941601153388824002931810367389600280334154412405474501934079536084168 5$,
$q_4 = 47592228616926132575334924965304845154512487924269472539555512857621026281795580048375808 1$.

$E_6/\mathbb{F}_{q_6} : y^2 = x^3 + A_6 x + B_6$ where
$A_6 = 11$,
$B_6 = 10670008051085173567796731963258535225645425120136758789018598936293600026260666846952307 4$,
$q_6 = 47592228616926132575334924965304845154512487855282351555326773573916464730740849055996313 7$.

Security. One may wonder if curves lying on PCD-friendly cycles are weak (e.g., in terms of DL hardness). Yet, MNT4 and MNT6 curves of suitable parameters are widely believed to be secure, and they *all* fall in PCD-friendly 2-cycles. The additional requirement of high 2-adicity is not known to cause weakness either.

3.3 A matched pair of preprocessing zk-SNARKs

Based on the PCD-friendly cycle (E_4, E_6), we designed and constructed two preprocessing zk-SNARKs for arithmetic circuit satisfiability: (G_4, P_4, V_4) based on the curve E_4, and (G_6, P_6, V_6) on E_6. The software implementation follows [BCTV14], the fastest preprocessing zk-SNARK implementation for circuits at the time of writing. We thus adapt the techniques in [BCTV14] to our algebraic setting, which consists of the two MNT curves E_4 and E_6, and achieve efficient implementations of (G_4, P_4, V_4) and (G_6, P_6, V_6).

The implementation itself entails many algorithmic and engineering details, and we refer the reader to [BCTV14] for a discussion of these techniques. We only provide a high-level efficiency comparison between the preprocessing zk-SNARK of [BCTV14] based on Edwards curves (also at 80-bit security), and our implementations of (G_4, P_4, V_4) and (G_6, P_6, V_6); see Figure 1. Our implementation is slower, because of two main reasons: (i) MNT curves do not enjoy advantageous properties that Edwards curves do; and (ii) the modulus sizes are larger (298 bits in our case vs. 180 bits in [BCTV14]). On the other hand, the fact that MNT curves lie on a PCD-friendly 2-cycle is crucial for the PCD construction described next.

[11] The authors are grateful to Andrew V. Sutherland for generous help in running the CM method on such a large discriminant.

	80 bits of security		
	$(G_{\mathsf{Ed}}, P_{\mathsf{Ed}}, V_{\mathsf{Ed}})$	(G_4, P_4, V_4)	(G_6, P_6, V_6)
Key generator	12.4 s	33.9 s	48.8 s
Prover	13.0 s	36.5 s	49.4 s
Verifier	4.4 ms	9.1 ms	16.7 ms
Proof size	230 B	337 B	374 B

Figure 1: Comparison of $(G_{\mathsf{Ed}}, P_{\mathsf{Ed}}, V_{\mathsf{Ed}})$, (G_4, P_4, V_4), and (G_6, P_6, V_6), on a circuit C with 2^{17} gates and inputs of 10 field elements. The size of C was chosen so that the 2-adicity of each zk-SNARK's curve is high enough (i.e., $\nu_2(r_i - 1) \geq 17$ for $i = \mathsf{Ed}, 4, 6$). The experiment was conducted on our benchmarking machine (described in Section 7), running in single-thread mode. (The reported times are the average of 10 experiments, with standard deviation less than 1%.)

4 Proof-carrying data from PCD-friendly zk-SNARKs

In Section 3 we formulated, and instantiated, PCD-friendly cycles of elliptic curves (see Definition 3.1); this notion was motivated by efficiency considerations arising when recursively composing zk-SNARK proofs. Roughly, given two zk-SNARKs based on elliptic curves forming a PCD-friendly 2-cycle, one can alternate between the two proof systems, and the 2-cycle property ensures that fields "match up" at each recursive verification, allowing for an efficient circuit implementation of the verifier of both proof systems.

The discussion so far, however, is only a sketch of the approach and omits key technical details. We now spell out these by describing how to construct a PCD system, given the two zk-SNARKs. So let (E_α, E_β) be a PCD-friendly 2-cycle of elliptic curves, and let $(G_\alpha, P_\alpha, V_\alpha)$ and $(G_\beta, P_\beta, V_\beta)$ be two preprocessing zk-SNARKs respectively instantiated with the two elliptic curves $E_\alpha/\mathbb{F}_{q_\alpha}$ and $E_\beta/\mathbb{F}_{q_\beta}$. Note that:

- $(G_\alpha, P_\alpha, V_\alpha)$ works for $\mathbb{F}_{r_\alpha}$-arithmetic circuit satisfiability, while V_α's computations are over $\mathbb{F}_{q_\alpha}$; and
- $(G_\beta, P_\beta, V_\beta)$ works for $\mathbb{F}_{r_\beta}$-arithmetic circuit satisfiability, while V_β's computations are over $\mathbb{F}_{q_\beta}$.

Due to the 2-cycle property, $\mathbb{F}_{r_\alpha}$ equals $\mathbb{F}_{q_\beta}$, and $\mathbb{F}_{r_\beta}$ equals $\mathbb{F}_{q_\alpha}$. Our goal is to use $(G_\alpha, P_\alpha, V_\alpha)$ and $(G_\beta, P_\beta, V_\beta)$, along with other ingredients, to construct a PCD system $(\mathbb{G}, \mathbb{P}, \mathbb{V})$.

Remark 4.1 (longer cycles)**.** As we have PCD-friendly cycles of length $\ell = 2$, the PCD construction described in this section (including our code) is specialized to this case. One can extend the construction to work with (preprocessing zk-SNARKs based on) PCD-friendly cycles of length $\ell > 2$.

4.1 Intuition

We begin by giving the intuition behind our construction of the PCD generator $\mathbb{G}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$. For simplicity, for now, we focus on the case where each node receives a single input message (i.e., the special case of "message chains" having arity $s = 1$).

Starting point. A natural first attempt is to construct two arithmetic circuits, $C_{\mathsf{pcd},\alpha}$ over $\mathbb{F}_{r_\alpha}$ and $C_{\mathsf{pcd},\beta}$ over $\mathbb{F}_{r_\beta}$, that, for a given compliance predicate Π, work as follows.

$\underline{C_{\mathsf{pcd},\alpha}(x, a)}$
1. Parse the input x as $(\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z)$.
2. Parse the witness a as $(z_{\mathsf{loc}}, z_{\mathsf{in}}, \pi_{\mathsf{in}})$.
3. If $\pi_{\mathsf{in}} = 0$ (base case), set $b_{\mathsf{base}} := 1$.
4. If $\pi_{\mathsf{in}} \neq 0$ (not base case), set $b_{\mathsf{base}} := 0$ and check that $V_\beta\big(\mathsf{vk}_\beta, (\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z_{\mathsf{in}}), \pi_{\mathsf{in}}\big)$ accepts.
5. Check that $\Pi(z, z_{\mathsf{loc}}, z_{\mathsf{in}}, b_{\mathsf{base}}) = 0$.

$\underline{C_{\mathsf{pcd},\beta}(x, a)}$
1. Parse the input x as $(\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z)$.
2. Parse the witness a as $(z_{\mathsf{loc}}, z_{\mathsf{in}}, \pi_{\mathsf{in}})$.
3. If $\pi_{\mathsf{in}} = 0$ (base case), set $b_{\mathsf{base}} := 1$.
4. If $\pi_{\mathsf{in}} \neq 0$ (not base case), set $b_{\mathsf{base}} := 0$ and check that $V_\alpha\big(\mathsf{vk}_\alpha, (\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z_{\mathsf{in}}), \pi_{\mathsf{in}}\big)$ accepts.
5. Check that $\Pi(z, z_{\mathsf{loc}}, z_{\mathsf{in}}, b_{\mathsf{base}}) = 0$.

In other words, $C_{\mathsf{pcd},\alpha}$ checks Π-compliance at a node and also verifies a previous proof, relative to V_β; while $C_{\mathsf{pcd},\beta}$ does the same, but verifies a previous proof relative to V_α. Also note that the input x, but not the witness a (over which we have no control), specifies the choice of verification keys.

More precisely, on input Π, the PCD generator $\mathbb{G}$ would work as follows: (i) construct $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$ from Π; (ii) sample two key pairs, $(\mathsf{pk}_\alpha, \mathsf{vk}_\alpha) \leftarrow G_4(C_{\mathsf{pcd},\alpha})$ and $(\mathsf{pk}_\beta, \mathsf{vk}_\beta) \leftarrow G_6(C_{\mathsf{pcd},\beta})$; and (iii) output $\mathsf{pk} := (\mathsf{pk}_\alpha, \mathsf{pk}_\beta, \mathsf{vk}_\alpha, \mathsf{vk}_\beta)$ and $\mathsf{vk} := (\mathsf{vk}_\alpha, \mathsf{vk}_\beta)$. On input proving key pk, outgoing message z, local data z_{loc}, and incoming message z_{in}, the PCD prover $\mathbb{P}$ would invoke $P_\alpha(\mathsf{pk}_\alpha, x, a)$ if π_{in} is relative to V_β or $P_\beta(\mathsf{pk}_\beta, x, a)$ if π_{in} is relative to V_α, where $x := (\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z)$ and $a := (z_{\mathsf{loc}}, z_{\mathsf{in}}, \pi_{\mathsf{in}})$. Finally, on input verification key vk, message z, and proof π, the PCD verifier $\mathbb{V}$ would either invoke $V_\alpha(\mathsf{vk}_\alpha, x, \pi)$ or $V_\beta(\mathsf{vk}_\beta, x, \pi)$, where $x := (\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z)$.

However, the above simple sketch suffers from two main problems, which we now describe.

Problem #1. The compliance predicate Π is an arithmetic circuit. However, should Π be defined over $\mathbb{F}_{r_\alpha}$ or $\mathbb{F}_{r_\beta}$? If Π is defined over $\mathbb{F}_{r_\alpha}$, then the $\mathbb{F}_{r_\beta}$-arithmetic circuit $C_{\mathsf{pcd},\beta}$ will be very inefficient, because it has to evaluate Π over the "wrong" field; conversely, if Π is defined over $\mathbb{F}_{r_\beta}$, then $C_{\mathsf{pcd},\alpha}$ will be very inefficient.

Problem #2. In known preprocessing zk-SNARK constructions, including the one underlying $(G_\alpha, P_\alpha, V_\alpha)$ and $(G_\beta, P_\beta, V_\beta)$, a verification key has length $\ell(n) > n$, where n is the size of the input to the circuit with respect to which the key was created. Thus, it is not possible to obtain either vk_α or vk_β that works for inputs of the form $x = (\mathsf{vk}_\alpha, \mathsf{vk}_\beta, z)$.

Our solution (at high level). To address the first problem, we simply "pick one side": only one of $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$ evaluates Π, while the other circuit merely enables the PCD prover to translate a proof relative to one zk-SNARK verifier to one relative to the other zk-SNARK verifier. Arbitrarily, we pick $C_{\mathsf{pcd},\alpha}$ to be the one that evaluates Π; in particular, Π will be an $\mathbb{F}_{r_\alpha}$-arithmetic circuit.[12] (The choice of $C_{\mathsf{pcd},\alpha}$ is without loss of generality, since we can always relabel: if (E_α, E_β) is PCD-friendly 2-cycle, so is (E_β, E_α).)

To address the second problem, the ideal solution is to simply hardcode vk_β in $C_{\mathsf{pcd},\alpha}$ and vk_α in $C_{\mathsf{pcd},\beta}$ (and let an input x consist only of a message z). However, this is not possible: vk_β depends on $C_{\mathsf{pcd},\beta}$, while vk_α depends on $C_{\mathsf{pcd},\alpha}$ (i.e., there is a circular dependency). We thus proceed as follows. We hardcode vk_α in $C_{\mathsf{pcd},\beta}$. Then, for vk_β, we rely on collision-resistant hashing. Namely, inputs x have the form (χ_β, z) where, allegedly, χ_β is the hash of vk_β. We modify $C_{\mathsf{pcd},\alpha}$ to check that this holds: $C_{\mathsf{pcd},\alpha}$'s witness is extended to (allegedly) contain vk_β and then $C_{\mathsf{pcd},\alpha}$ checks that $H_\alpha(\mathsf{vk}_\beta) = \chi_\beta$, where $H_\alpha \colon \{0,1\}^{m_{H,\alpha}} \to \mathbb{F}_{r_\alpha}^{d_{H,\alpha}}$ is a suitable collision-resistant hash function.

The above modifications to $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$ yield the following construction.

$\underline{C_{\mathsf{pcd},\alpha}(x, a)}$
1. Parse the input x as (χ_β, z).
2. Parse the witness a as $(\mathsf{vk}_\beta, z_{\mathsf{loc}}, z_{\mathsf{in}}, \pi_{\mathsf{in}})$.
3. Check that $H_\alpha(\mathsf{vk}_\beta) = \chi_\beta$.
4. If $\pi_{\mathsf{in}} = 0$ (base case), set $b_{\mathsf{base}} := 1$.
5. If $\pi_{\mathsf{in}} \neq 0$ (not base case), set $b_{\mathsf{base}} := 0$ and check that $V_\beta\big(\mathsf{vk}_\beta, (\chi_\beta, z_{\mathsf{in}}), \pi_{\mathsf{in}}\big)$ accepts.
6. Check that $\Pi(z, z_{\mathsf{loc}}, z_{\mathsf{in}}, b_{\mathsf{base}}) = 0$.

$\underline{C_{\mathsf{pcd},\beta}(x, a)}$
1. Parse the input x as (χ_β, z).
2. Parse the witness a as (π_α).
3. Check that $V_\alpha\big(\mathsf{vk}_\alpha, (\chi_\beta, z), \pi_\alpha\big)$ accepts.

Unfortunately, while the above two fixes make $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$ well-defined, the use of hashing makes $C_{\mathsf{pcd},\alpha}$ large, as we now explain. The verification key vk_β consists of various points on the elliptic curve $E_\beta/\mathbb{F}_{q_\beta}$ (or a twist of it). Recalling that $\mathbb{F}_{q_\beta}$ equals $\mathbb{F}_{r_\alpha}$ (the field of definition of $C_{\mathsf{pcd},\alpha}$), we represent vk_β as a list of elements of $\mathbb{F}_{q_\beta}$, because the (circuit implementing the) verifier V_β uses vk_β for computations in $\mathbb{F}_{q_\beta}$. Unfortunately, H_α only accepts binary strings as input, and the translation from vk_β to its binary representation incurs a $\log q_\beta$ blow up — a nontrivial cost. We do not know how to eliminate this cost.[13]

Thus, instead, we further modify $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$ so to shorten vk_β. Indeed, if $C_{\mathsf{pcd},\beta}$ accepts inputs of n elements in $\mathbb{F}_{r_\beta}$, then vk_β consists of $\ell_{\mathsf{vk},\beta}(n)$ elements in $\mathbb{F}_{q_\beta}$; hence, we seek to reduce n. To do so, instead of working with messages of the form $x = (\chi_\beta, z)$, we work with messages of the form $x = (\chi_\beta)$, by also hashing z along with vk_β (both of which are now supplied in the witness), as follows.

$\underline{C_{\mathsf{pcd},\alpha}(x, a)}$
1. Parse the input x as (χ_β).
2. Parse the witness a as $(\mathsf{vk}_\beta, z, z_{\mathsf{loc}}, z_{\mathsf{in}}, \pi_{\mathsf{in}})$.
3. Check that $H_\alpha(\mathsf{vk}_\beta \| z) = \chi_\beta$.
4. If $\pi_{\mathsf{in}} = 0$ (base case), set $b_{\mathsf{base}} := 1$.
5. If $\pi_{\mathsf{in}} \neq 0$ (not base case), set $b_{\mathsf{base}} := 0$ and check that $V_\beta\big(\mathsf{vk}_\beta, H_\alpha(\mathsf{vk}_\beta \| z_{\mathsf{in}}), \pi_{\mathsf{in}}\big)$ accepts.
6. Check that $\Pi(z, z_{\mathsf{loc}}, z_{\mathsf{in}}, b_{\mathsf{base}}) = 0$.

$\underline{C_{\mathsf{pcd},\beta}(x, a)}$
1. Parse the input x as (χ_β).
2. Parse the witness a as a zk-SNARK proof π_α.
3. Check that $V_\alpha(\mathsf{vk}_\alpha, \chi_\beta, \pi_\alpha)$ accepts.

[12] Alternatively, we could restrict Π to be a boolean circuit, so that it can be easily evaluated by $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$. But this foregoes Π's ability to conduct field operations in large prime fields. Thus, it is more efficient to "give up" on $C_{\mathsf{pcd},\beta}$, and only let $C_{\mathsf{pcd},\alpha}$ evaluate Π, and retain the expressive power of arithmetic circuits. (Other alternatives are possible, but we do not explore them.)

[13] As explained in Section 5.2, we are not aware of a collision-resistant hash function H_α, which can be easily verified over $\mathbb{F}_{r_\alpha}$, that accepts inputs represented as strings of elements in $\mathbb{F}_{r_\alpha}$. So, we are "stuck" with binary inputs.

Further details. The above discussion omits various technical details and optimizations.

For instance, thus far we have ignored the fact that, while $C_{\mathsf{pcd},\alpha}$ expects inputs over $\mathbb{F}_{r_\alpha}$, $C_{\mathsf{pcd},\beta}$ expects inputs over $\mathbb{F}_{r_\beta}$. Since $x = \chi_\beta$ lies in $\mathbb{F}_{r_\alpha}^{d_{H,\alpha}}$ (as it is the output of H_α), we cannot use the same representation of it for both $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$; instead, we need two representations: $x_\alpha \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ for $C_{\mathsf{pcd},\alpha}$, and $x_\beta \in \mathbb{F}_{r_\beta}^{n_\beta}$ for $C_{\mathsf{pcd},\beta}$. Naturally, for the first, we can set $n_\alpha := d_{H,\alpha}$, and let $x_\alpha := \chi_\beta$. For the second, merely letting x_β be the list of $n_\alpha \cdot \lceil \log r_\alpha \rceil$ bits in χ_β is *not* efficient: it would cause vk_β to have length $\ell_{\mathsf{vk},\beta}(n_\alpha \cdot \lceil \log r_\alpha \rceil)$. Instead, we let x_β store these bits into as few elements of $\mathbb{F}_{r_\beta}$ as possible; specifically, $n_\beta := \lceil \frac{n_\alpha \cdot \lceil \log r_\alpha \rceil}{\lfloor \log r_\beta \rfloor} \rceil$ of them. So let:

- $S_{\alpha\to\beta} \colon \mathbb{F}_{r_\alpha}^{n_\alpha} \to \mathbb{F}_{r_\alpha}^{n_\beta \cdot \lceil \log r_\beta \rceil}$ denote the function that maps x_α to (the binary representation of) x_β; and
- $S_{\alpha\leftarrow\beta} \colon \mathbb{F}_{r_\beta}^{n_\beta} \to \mathbb{F}_{r_\beta}^{n_\alpha \cdot \lceil \log r_\alpha \rceil}$ denote the function that maps x_β back to (the binary representation of) x_α.

The above implies that we need to further modify $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$, and include explicit subcircuits $C_{S,\alpha\to\beta}$ and $C_{S,\alpha\leftarrow\beta}$ to carry out these "type conversions"; both of these circuits are simple to construct, and have size $|C_{S,\alpha\to\beta}| = |C_{S,\alpha\leftarrow\beta}| = n_\alpha \cdot \lceil \log r_\alpha \rceil$.

Moreover, we leverage precomputation techniques [BCTV14]. A zk-SNARK verifier V can be viewed as two functions: an "offline" function V^{offline} that, given the verification key vk, computes a *processed verification key* pvk; and an "online" function V^{online} that, given pvk, an input x, and proof π, computes the decision bit. (I.e., $V(\mathsf{vk}, x, \pi) := V^{\text{online}}\big(V^{\text{offline}}(\mathsf{vk}), x, \pi\big)$.) Precomputation offers a tradeoff: while V^{online} is cheaper to compute than V, pvk is larger than vk (in each case, the difference is an additive constant). In our setting, it turns out that it pays off to use precomputation techniques only in $C_{\mathsf{pcd},\beta}$ but not in $C_{\mathsf{pcd},\alpha}$.

We address all the details in the next subsection, where we give the construction of the PCD system.

4.2 Construction

We now describe in more detail our construction of the PCD generator $\mathbb{G}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$. Throughout, we fix a message size $n_{\mathsf{msg}} \in \mathbb{N}$, local-data size $n_{\mathsf{loc}} \in \mathbb{N}$, and arity $s \in \mathbb{N}$. The construction will then work for $\mathbb{F}_{r_\alpha}$-arithmetic compliance predicates $\Pi \colon \mathbb{F}_{r_\alpha}^{n_{\mathsf{msg}}} \times \mathbb{F}_{r_\alpha}^{n_{\mathsf{loc}}} \times \mathbb{F}_{r_\alpha}^{s \cdot n_{\mathsf{msg}}} \times \mathbb{F}_{r_\alpha} \to \mathbb{F}_{r_\alpha}^{l}$ (i.e., for message size n_{msg}, local-data size n_{loc}, arity s, and some output size $l \in \mathbb{N}$). In terms of ingredients, we make use of the following arithmetic circuits:

- An $\mathbb{F}_{r_\alpha}$-arithmetic circuit $C_{H,\alpha}$, implementing a collision-resistant function $H_\alpha \colon \{0,1\}^{m_{H,\alpha}} \to \mathbb{F}_{r_\alpha}^{d_{H,\alpha}}$ such that $m_{H,\alpha} \geq \big(\ell_{\mathsf{vk},\beta}(n_\beta) + n_{\mathsf{msg}}\big) \cdot \lceil \log r_\alpha \rceil$, where $n_\alpha := d_{H,\alpha}$ and $n_\beta := \lceil \frac{n_\alpha \cdot \lceil \log r_\alpha \rceil}{\lfloor \log r_\beta \rfloor} \rceil$.
- An $\mathbb{F}_{r_\alpha}$-arithmetic circuit $C_{S,\alpha\to\beta}$, implementing $S_{\alpha\to\beta} \colon \mathbb{F}_{r_\alpha}^{n_\alpha} \to \mathbb{F}_{r_\alpha}^{n_\beta \cdot \lceil \log r_\beta \rceil}$.
- An $\mathbb{F}_{r_\beta}$-arithmetic circuit $C_{S,\alpha\leftarrow\beta}$, implementing $S_{\alpha\leftarrow\beta} \colon \mathbb{F}_{r_\beta}^{n_\beta} \to \mathbb{F}_{r_\beta}^{n_\alpha \cdot \lceil \log r_\alpha \rceil}$.
- An $\mathbb{F}_{r_\beta}$-arithmetic circuit $C_{V,\alpha}^{\text{online}}$, implementing V_α^{online} for inputs of n_α elements in $\mathbb{F}_{r_\alpha}$; an input $x_\alpha \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ is given to $C_{V,\alpha}^{\text{online}}$ as a string of $n_\alpha \cdot \lceil \log r_\alpha \rceil$ elements in $\mathbb{F}_{r_\beta}$, each carrying a bit of x_α.
- An $\mathbb{F}_{r_\alpha}$-arithmetic circuit $C_{V,\beta}$, implementing V_β for inputs of n_β elements in $\mathbb{F}_{r_\beta}$; an input $x_\beta \in \mathbb{F}_{r_\beta}^{n_\beta}$ is given to $C_{V,\beta}$ as a string of $n_\beta \cdot \lceil \log r_\beta \rceil$ elements in $\mathbb{F}_{r_\alpha}$, each carrying a bit of x_β.

For now we take the above circuits as given; later, in Section 5 we discuss our concrete instantiations of them. Also, we generically denote by bits_α a function that, given an input y in $\mathbb{F}_{r_\alpha}^{\ell}$ (for some ℓ), outputs y's binary representation; the corresponding $\mathbb{F}_{r_\alpha}$-arithmetic circuit is denoted $C_{\mathsf{bits},\alpha}$, and has $\ell \cdot \lceil \log r_\alpha \rceil$ gates.[14]

For reference, pseudocode for the triple $(\mathbb{G}, \mathbb{P}, \mathbb{V})$ is given in Figure 2.

[14] More precisely, for each $\mathbb{F}_{r_\alpha}$-element y_i in the vector y, bits_α outputs bits $b_1, \ldots, b_{\lceil \log r_\alpha \rceil}$ such that $\sum_{j=0}^{\lceil \log r_\alpha \rceil - 1} b_j 2^j = y_i$, where arithmetic is conducted over $\mathbb{F}_{r_\alpha}$. Due to wrap around, some elements in $\mathbb{F}_{r_\alpha}$ have *two* such representations; if so, bits_α outputs the lexicographically-first one. None the less, we construct $C_{\mathsf{bits},\alpha}$ to only check for either of these two representations, because: (i) discriminating between representations costs an additional $\ell \cdot \lceil \log r_\alpha \rceil$ gates; and (ii) doing so does not affect completeness or soundness of our construction.

The PCD generator. The PCD generator $\mathbb{G}$ takes as input an $\mathbb{F}_{r_\alpha}$-arithmetic compliance predicate Π, and outputs a key pair $(\mathsf{pk}, \mathsf{vk})$ for proving/verifying Π-compliance. The PCD generator works as follows: (i) it uses $C_{H,\alpha}, C_{S,\alpha\to\beta}, C_{V,\beta}, \Pi$ to construct the circuit $C_{\mathsf{pcd},\alpha}$; (ii) it samples a key pair, $(\mathsf{pk}_\alpha, \mathsf{vk}_\alpha) \leftarrow G_4(C_{\mathsf{pcd},\alpha})$; (iii) it uses $\mathsf{vk}_\alpha, C_{S,\alpha\leftarrow\beta}, C^{\text{online}}_{V,\alpha}$ to construct the other circuit $C_{\mathsf{pcd},\beta}$; (iv) it samples another key pair, $(\mathsf{pk}_\beta, \mathsf{vk}_\beta) \leftarrow G_6(C_{\mathsf{pcd},\beta})$; and (v) it outputs $\mathsf{pk} := (\mathsf{pk}_\alpha, \mathsf{pk}_\beta, \mathsf{vk}_\alpha, \mathsf{vk}_\beta)$ and $\mathsf{vk} := (\mathsf{vk}_\alpha, \mathsf{vk}_\beta)$.

We now describe $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$. The circuit $C_{\mathsf{pcd},\beta}$ acts as a "proof converter": it takes an input $x_\beta \in \mathbb{F}_{r_\beta}^{n_\beta}$ and a witness $a_\beta \in \mathbb{F}_{r_\beta}^{h_\beta}$, parses a_β as a zk-SNARK proof π_α for V_α, and simply checks that $C^{\text{online}}_{V,\alpha}\big(\mathsf{vk}_\alpha, C_{S,\alpha\leftarrow\beta}(x_\beta), \pi_\alpha\big) = 1$. (The verification key vk_α is hardcoded in $C_{\mathsf{pcd},\beta}$.)

In contrast, the circuit $C_{\mathsf{pcd},\alpha}$ verifies Π-compliance: it takes an input $x_\alpha \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ and a witness $a_\alpha \in \mathbb{F}_{r_\alpha}^{h_\alpha}$, parses a_α as $(\mathsf{vk}_\beta, z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, b_{\mathsf{base}}, \vec{\pi}_{\mathsf{in}}, b_{\mathsf{res}})$, and verifies that $x_\alpha = C_{H,\alpha}(C_{\mathsf{bits},\alpha}(\mathsf{vk}_\beta) \| C_{\mathsf{bits},\alpha}(z))$ and that $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, b_{\mathsf{base}}) = 0$. Moreover, if $b_{\mathsf{base}} = 0$ (not the base case), $C_{\mathsf{pcd},\alpha}$ also recursively verifies Π-compliance of previous messages: for each corresponding pair $(z_{\mathsf{in}}, \pi_{\mathsf{in}})$ in $(\vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})$, it verifies that $C_{V,\beta}\big(\mathsf{vk}_\beta, x_{\mathsf{in},\beta}, \pi_{\mathsf{in}}\big) = 1$ where $x_{\mathsf{in},\beta} = C_{S,\alpha\to\beta}(C_{H,\alpha}(C_{\mathsf{bits},\alpha}(\mathsf{vk}_\beta) \| C_{\mathsf{bits},\alpha}(z_{\mathsf{in}})))$.

See Figure 2 for details. Overall, the two circuits have the following sizes:

$$\begin{aligned} |C_{\mathsf{pcd},\alpha}| &\approx \big(\ell_{\mathsf{vk},\beta}(n_\beta) + (1+s)n_{\mathsf{msg}}\big) \cdot \lceil \log r_\alpha \rceil + |C_{H,\alpha}| + |\Pi| + s \cdot \big(|C_{H,\alpha}| + |C_{S,\alpha\to\beta}| + |C_{V,\beta}|\big) \ , \\ |C_{\mathsf{pcd},\beta}| &\approx |C_{S,\alpha\leftarrow\beta}| + |C^{\text{online}}_{V,\alpha}| \ . \end{aligned} \tag{1}$$

The PCD prover. The PCD prover $\mathbb{P}$ takes as input a proving key pk, outgoing message z, local data z_{loc}, and incoming messages $\vec{z}_{\mathsf{in}}$; when not in the base case, it also takes as input proofs $\vec{\pi}_{\mathsf{in}}$, each attesting that a message in $\vec{z}_{\mathsf{in}}$ is Π-compliant. The PCD prover outputs a proof π attesting to the fact that z is Π-compliant.

At high level, the PCD prover performs not one, but two, steps of recursive composition, "going around the PCD-friendly 2-cycle". The first step is relative to $C_{\mathsf{pcd},\alpha}$ and checks Π-compliance; the second step is relative to $C_{\mathsf{pcd},\beta}$ and merely converts the proof produced by the first step to the other verifier. More precisely, the PCD prover constructs $x_\alpha := H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z)) \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ and then uses P_α to produce a proof π_α attesting that $x_\alpha \in \mathcal{L}_{C_{\mathsf{pcd},\alpha}}$. In the base case, $C_{\mathsf{pcd},\alpha}$ only verifies that $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 1) = 0$; but, when previous proofs $\vec{\pi}_{\mathsf{in}}$ are supplied, $C_{\mathsf{pcd},\alpha}$ verifies instead that $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 0) = 0$ and, for each pair $(z_{\mathsf{in}}, \pi_{\mathsf{in}})$, that $V_\beta\big(\mathsf{vk}_\beta, x_{\beta,\mathsf{in}}, \pi_{\mathsf{in}}\big) = 1$ where $x_{\beta,\mathsf{in}} := S_{\alpha\to\beta}(H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z_{\mathsf{in}}))) \in \mathbb{F}_{r_\beta}^{n_\beta}$. Next, the PCD prover uses P_β to convert π_α into a proof π attesting that $x_\beta \in \mathcal{L}_{C_{\mathsf{pcd},\beta}}$, where $x_\beta := S_{\alpha\to\beta}(x_\alpha) \in \mathbb{F}_{r_\beta}^{n_\beta}$; this is merely a translation because $C_{\mathsf{pcd},\beta}$ only verifies that π_α is valid. The proof π is $\mathbb{P}$'s output.

The PCD verifier. The PCD verifier $\mathbb{V}$ takes as input a verification key vk, message z, and proof π. Proofs relative to $(G_\alpha, P_\alpha, V_\alpha)$ are never "seen" outside the PCD prover, because the prover converts them to proofs relative to $(G_\beta, P_\beta, V_\beta)$. Hence, the proof π is relative to $(G_\beta, P_\beta, V_\beta)$, and the PCD verifier checks that z is Π-compliant by checking that $V_\beta(\mathsf{vk}_\beta, x_\beta, \pi) = 1$, where $x_\beta := S_{\alpha\to\beta}(H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z))) \in \mathbb{F}_{r_\beta}^{n_\beta}$.

4.3 Security

The intuition for the security of $(\mathbb{G}, \mathbb{P}, \mathbb{V})$ is straightforward. Suppose that a malicious polynomial-size prover $\tilde{P}$ outputs a message z and proof π that are accepted by the PCD verifier $\mathbb{V}$. Our goal is to deduce that z is Π-compliant. By construction of $\mathbb{V}$, we deduce that $S_{\alpha\to\beta}(H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z))) \in \mathcal{L}_{C_{\mathsf{pcd},\beta}}$. In turn, by construction of $C_{\mathsf{pcd},\beta}$, we deduce that $H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z)) \in \mathcal{L}_{C_{\mathsf{pcd},\alpha}}$. In turn, by construction of $C_{\mathsf{pcd},\alpha}$, we deduce that there is local data z_{loc} and previous messages $\vec{z}_{\mathsf{in}}$ such that one of the following holds: (i) $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 1) = 0$, which is the base case; or (ii) $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 0) = 0$ and, for each incoming message z_{in}, $S_{\alpha\to\beta}(H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z_{\mathsf{in}}))) \in \mathcal{L}_{C_{\mathsf{pcd},\beta}}$ (and thus, by induction, that each z_{in} is Π-compliant). In either case, we conclude that z is Π-compliant.

The above argument can be formalized by using the *proof-of-knowledge property* of zk-SNARKs. Yet, as explained in Section 2.3, a formal argument lies beyond the scope of this paper, which instead focuses on practical aspects of PCD systems; see [BCCT13] for more details.

$\mathsf{MakePCDCircuitA}(C_{H,\alpha}, C_{S,\alpha\to\beta}, C_{V,\beta}, \Pi)$

Output the $\mathbb{F}_{r_\alpha}$-arithmetic circuit $C_{\mathsf{pcd},\alpha}$ that, given input $x_\alpha \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ and witness $a_\alpha \in \mathbb{F}_{r_\alpha}^{h_\alpha}$, works as follows:

1. Parse a_α as $(\mathsf{vk}_\beta, z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, b_{\mathsf{base}}, \vec{\pi}_{\mathsf{in}}, b_{\mathsf{res}})$.
2. Compute $\sigma_{\mathsf{vk},\beta} := C_{\mathsf{bits},\alpha}(\mathsf{vk}_\beta)$.
3. Check that $x_\alpha = C_{H,\alpha}(\sigma_{\mathsf{vk},\beta} \| C_{\mathsf{bits},\alpha}(z))$.
4. Check that $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, b_{\mathsf{base}}) = 0$.
5. For each $(z_{\mathsf{in}}, \pi_{\mathsf{in}}) \in (\vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})$:
   (a) Compute $x_{\mathsf{in},\alpha} := C_{H,\alpha}(\sigma_{\mathsf{vk},\beta} \| C_{\mathsf{bits},\alpha}(z_{\mathsf{in}})) \in \mathbb{F}_{r_\alpha}^{n_\alpha}$.
   (b) Compute $x_{\mathsf{in},\beta} := C_{S,\alpha\to\beta}(x_{\mathsf{in},\alpha}) \in \mathbb{F}_{r_\alpha}^{n_\beta \cdot \lceil \log r_\beta \rceil}$.
   (c) Check that $C_{V,\beta}(\mathsf{vk}_\beta, x_{\mathsf{in},\beta}, \pi_{\mathsf{in}}) = b_{\mathsf{res}}$.
6. Check that $b_{\mathsf{base}}, b_{\mathsf{res}} \in \{0,1\}$ and $(1 - b_{\mathsf{base}})(1 - b_{\mathsf{res}}) = 0$.

$\mathsf{MakePCDCircuitB}(\mathsf{pvk}_\alpha, C_{S,\alpha\leftarrow\beta}, C_{V,\alpha}^{\text{online}})$

Output the $\mathbb{F}_{r_\beta}$-arithmetic circuit $C_{\mathsf{pcd},\beta}$ that, given input $x_\beta \in \mathbb{F}_{r_\beta}^{n_\beta}$ and witness $a_\beta \in \mathbb{F}_{r_\beta}^{h_\beta}$, works as follows:

1. Parse a_β as a zk-SNARK proof π_α.
2. Compute $x_\alpha := C_{S,\alpha\leftarrow\beta}(x_\beta) \in \mathbb{F}_{r_\beta}^{n_\alpha \cdot \lceil \log r_\alpha \rceil}$.
3. Check that $C_{V,\alpha}^{\text{online}}(\mathsf{pvk}_\alpha, x_\alpha, \pi_\alpha) = 1$.

PARAMETERS. Message size $n_{\mathsf{msg}} \in \mathbb{N}$, local-data size $n_{\mathsf{loc}} \in \mathbb{N}$, and arity $s \in \mathbb{N}$.

PCD generator $\mathbb{G}$

- INPUTS: a compliance predicate $\Pi\colon \mathbb{F}_{r_\alpha}^{n_{\mathsf{msg}}} \times \mathbb{F}_{r_\alpha}^{n_{\mathsf{loc}}} \times \mathbb{F}_{r_\alpha}^{s \cdot n_{\mathsf{msg}}} \times \mathbb{F}_{r_\alpha} \to \mathbb{F}_{r_\alpha}^{l}$ (for some $l \in \mathbb{N}$)
- OUTPUTS: proving key pk and verification key vk

1. Set $n_\alpha := d_{H,\alpha}$ and $n_\beta := \lceil \frac{n_\alpha \cdot \lceil \log r_\alpha \rceil}{\lfloor \log r_\beta \rfloor} \rceil$.
2. Construct $C_{H,\alpha}$, the $\mathbb{F}_{r_\alpha}$-arithmetic circuit implementing $H_\alpha\colon \{0,1\}^{m_{H,\alpha}} \to \mathbb{F}_{r_\alpha}^{d_{H,\alpha}}$.
3. Construct $C_{S,\alpha\to\beta}$, the $\mathbb{F}_{r_\alpha}$-arithmetic circuit implementing $S_{\alpha\to\beta}\colon \mathbb{F}_{r_\alpha}^{n_\alpha} \to \mathbb{F}_{r_\alpha}^{n_\beta \cdot \lceil \log r_\beta \rceil}$.
4. Construct $C_{S,\alpha\leftarrow\beta}$, the $\mathbb{F}_{r_\beta}$-arithmetic circuit implementing $S_{\alpha\leftarrow\beta}\colon \mathbb{F}_{r_\beta}^{n_\beta} \to \mathbb{F}_{r_\beta}^{n_\alpha \cdot \lceil \log r_\alpha \rceil}$.
5. Construct $C_{V,\beta}$, the $\mathbb{F}_{r_\alpha}$-arithmetic circuit implementing V_β for inputs of n_β elements in $\mathbb{F}_{r_\beta}$.
6. Construct $C_{V,\alpha}^{\text{online}}$, the $\mathbb{F}_{r_\beta}$-arithmetic circuit implementing V_α^{online} for inputs of n_α elements in $\mathbb{F}_{r_\alpha}$.
7. Compute $C_{\mathsf{pcd},\alpha} := \mathsf{MakePCDCircuitA}(C_{H,\alpha}, C_{S,\alpha\to\beta}, C_{V,\beta}, \Pi)$.
8. Compute $(\mathsf{pk}_\alpha, \mathsf{vk}_\alpha) := G_\alpha(C_{\mathsf{pcd},\alpha})$.
9. Compute $\mathsf{pvk}_\alpha := V_\alpha^{\text{offline}}(\mathsf{vk}_\alpha)$.
10. Compute $C_{\mathsf{pcd},\beta} := \mathsf{MakePCDCircuitB}(\mathsf{pvk}_\alpha, C_{S,\alpha\leftarrow\beta}, C_{V,\alpha}^{\text{online}})$.
11. Compute $(\mathsf{pk}_\beta, \mathsf{vk}_\beta) := G_\beta(C_{\mathsf{pcd},\beta})$.
12. Set $\mathsf{pk} := (\mathsf{pk}_\alpha, \mathsf{pk}_\beta, \mathsf{vk}_\alpha, \mathsf{vk}_\beta)$ and $\mathsf{vk} := (\mathsf{vk}_\alpha, \mathsf{vk}_\beta)$.
13. Output $(\mathsf{pk}, \mathsf{vk})$.

PCD prover $\mathbb{P}$

- INPUTS:
  - proving key pk
  - outgoing message $z \in \mathbb{F}_{r_\alpha}^{n_{\mathsf{msg}}}$
  - local data $z_{\mathsf{loc}} \in \mathbb{F}_{r_\alpha}^{n_{\mathsf{loc}}}$
  - incoming messages $\vec{z}_{\mathsf{in}} \in \mathbb{F}_{r_\alpha}^{s \cdot n_{\mathsf{msg}}}$
  - previous proofs $\vec{\pi}_{\mathsf{in}}$ ($\vec{\pi}_{\mathsf{in}} = \perp$ in the base case, as there is no previous proofs)
- OUTPUTS: proof π for the outgoing message z

1. Compute $x_\alpha := H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z)) \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ and $x_\beta := S_{\alpha\to\beta}(x_\alpha) \in \mathbb{F}_{r_\alpha}^{n_\beta \cdot \lceil \log r_\beta \rceil}$, and parse x_β as lying in $\mathbb{F}_{r_\beta}^{n_\beta}$.
2. If base case (i.e., $\vec{\pi}_{\mathsf{in}} = \perp$), then set $a_\alpha := (\mathsf{vk}_\beta, z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 1, *, *)$, where $*$ is any assignment (of the correct length).
3. If not base case (i.e., $\vec{\pi}_{\mathsf{in}} \neq \perp$), then set $a_\alpha := (\mathsf{vk}_\beta, z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 0, \vec{\pi}_{\mathsf{in}}, 1)$.
4. Compute $\pi_\alpha := P_\alpha(\mathsf{pk}_\alpha, x_\alpha, a_\alpha)$.
5. Set $a_\beta := (\pi_\alpha)$.
6. Compute $\pi := P_\beta(\mathsf{pk}_\beta, x_\beta, a_\beta)$.
7. Output π.

PCD verifier $\mathbb{V}$

- INPUTS: verification key vk, message $z \in \mathbb{F}_{r_\alpha}^{n_{\mathsf{msg}}}$, and proof π
- OUTPUTS: decision bit

1. Compute $x_\alpha := H_\alpha(\mathsf{bits}_\alpha(\mathsf{vk}_\beta) \| \mathsf{bits}_\alpha(z)) \in \mathbb{F}_{r_\alpha}^{n_\alpha}$ and $x_\beta := S_{\alpha\to\beta}(x_\alpha) \in \mathbb{F}_{r_\alpha}^{n_\beta \cdot \lceil \log r_\beta \rceil}$, and parse x_β as lying in $\mathbb{F}_{r_\beta}^{n_\beta}$.
2. Compute $b := V_\beta(\mathsf{vk}_\beta, x_\beta, \pi)$ and output b.

Figure 2: Construction of a PCD system from two PCD-friendly preprocessing zk-SNARKs, along with other arithmetic circuits.

5 Constructions of arithmetic circuits

In Section 4 we gave a construction of a PCD system $(\mathbb{G}, \mathbb{P}, \mathbb{V})$ in terms of two preprocessing zk-SNARKs, based on a PCD-friendly 2-cycle (E_α, E_β), and various arithmetic circuits. We now discuss concrete implementations of these arithmetic circuits, which determine the sizes of $C_{\mathsf{pcd},\alpha}$ and $C_{\mathsf{pcd},\beta}$ (see Equation 1).

In our code implementation, (E_α, E_β) equals (E_4, E_6), a specific 2-cycle based on MNT curves of embedding degree 4 and 6, selected to have high 2-adicity (see Section 3.2). Thus, in the text below, "$\alpha = 4$ and $\beta = 6$".[15] We obtain the following efficiency for the two circuits $C_{\mathsf{pcd},4}$ and $C_{\mathsf{pcd},6}$.

Lemma 5.1 (informal)**.** *Let $\Pi\colon \mathbb{F}_{r_4}^{n_{\mathsf{msg}}} \times \mathbb{F}_{r_4}^{n_{\mathsf{loc}}} \times \mathbb{F}_{r_4}^{s \cdot n_{\mathsf{msg}}} \times \mathbb{F}_{r_4} \to \mathbb{F}_{r_4}^{l}$ be an $\mathbb{F}_{r_4}$-arithmetic compliance predicate for message size n_{msg}, local-data size n_{loc}, arity s (and some output size l). The $\mathbb{F}_{r_4}$-arithmetic circuit $C_{\mathsf{pcd},4}$ and the $\mathbb{F}_{r_6}$-arithmetic circuit $C_{\mathsf{pcd},6}$ have the following number of gates.*

Gate count for $C_{\mathsf{pcd},4}$	
booleanity checks	$11920 + (1+s) \cdot n_{\mathsf{msg}} \cdot 298$
$(1+s)$ copies of $C_{H,6}$	$(1+s) \times 1$
s copies of $C_{S,4\leftarrow 6}$	$s \times 298$
s copies of $C_{V,6}$ for $n_6 = 2$	$s \times 89113$
Π	$\lvert\Pi\rvert$
misc.	4
Total	$\lvert\Pi\rvert + s \cdot 89412$ $+(1+s) \cdot n_{\mathsf{msg}} \cdot 298 + 11925$

Gate count for $C_{\mathsf{pcd},6}$	
$C_{S,4\leftarrow 6}$	298
$C_{V,4}^{\text{online}}$ for $n_4 = 1$	31729
Total	32027

Next, we discuss the various subcircuits: for the zk-SNARK verifiers and for collision-resistant hashing.

Remark 5.2. We selected the PCD-friendly 2-cycle (E_4, E_6) to have high 2-adicity: it has $\nu_2(r_4 - 1) = 34$ and $\nu_2(r_6 - 1) = 17$. These values are not accidental, but were chosen so that the 2-cycle (E_4, E_6) suffices for "essentially all practical uses" of our PCD system. Specifically, recall that we would like, for efficiency reasons, that (i) $\nu_2(r_4 - 1) \geq \lceil \log |C_{\mathsf{pcd},4}| \rceil$ and (ii) $\nu_2(r_6 - 1) \geq \lceil \log |C_{\mathsf{pcd},6}| \rceil$ (see Section 3.2 and Appendix C.2). First, since $\lceil \log |C_{\mathsf{pcd},6}| \rceil = 15$, Condition (ii) holds always. As for Condition (i), it depends on Π; however, since $\nu_2(r_4 - 1) = 34$ is so large, it is not a limitation for practically-feasible choices of Π.[16]

5.1 Arithmetic circuits for zk-SNARK verifiers

We seek arithmetic circuits for the two zk-SNARK verifiers: an $\mathbb{F}_{r_6}$-arithmetic circuit $C_{V,4}$ implementing V_4 and an $\mathbb{F}_{r_4}$-arithmetic circuit $C_{V,6}$ implementing V_6. Note the field characteristics: V_4's arithmetic operations are over $\mathbb{F}_{q_4}$ (which is equal to $\mathbb{F}_{r_6}$) and V_6's operations are over $\mathbb{F}_{q_6}$ (which is equal to $\mathbb{F}_{r_4}$).

We design and construct $C_{V,4}$ and $C_{V,6}$, each consisting of two subcircuits for the "offline" and "online" parts of the verifier (see Section 4.1), and achieve the following efficiency:

Lemma 5.3 (informal)**.** *Let $n, l \in \mathbb{N}$.*

- *There is an $\mathbb{F}_{q_4}$-arithmetic circuit $C_{V,4}$ with size*

$$(10 \cdot l - 4) \cdot n + 43{,}767$$

that implements V_4 for all inputs $x \in \mathbb{F}_{r_4}^n$ such that each x_i has at most l bits. (Naturally, $l \leq \lceil \log r_4 \rceil$.) Moreover, $C_{V,4}$ consists of two subcircuits, $C_{V,4}^{\text{offline}}$ and $C_{V,4}^{\text{online}}$, implementing V_4^{offline} and V_4^{online}, with sizes

$$12270 \quad \textit{and} \quad (10 \cdot l - 4) \cdot n + 31{,}497 \ .$$

[15] The choice $(E_\alpha, E_\beta) = (E_4, E_6)$, rather than $(E_\alpha, E_\beta) = (E_6, E_4)$, is intentional. We expect $C_{\mathsf{pcd},\alpha}$ to be larger than $C_{\mathsf{pcd},\beta}$ (due to a larger number of checks), so that E_α should be the curve with the higher 2-adicity. In this case, E_4 is twice as 2-adic as E_6.

[16] More precisely, if we take $|\Pi| + s \cdot 89412$ to be the leading terms in $|C_{\mathsf{pcd},4}|$ (which we expect to be the case), we obtain that $\log(|\Pi| + s \cdot 89412) \leq 1 + \max\{\log |\Pi|, 17 + \log s\}$, which is likely to be well below 34.

- *There is an $\mathbb{F}_{q_6}$-arithmetic circuit $C_{V,6}$ with size*

$$(10 \cdot l - 4) \cdot n + 83{,}181$$

that implements V_6 for all inputs $x \in \mathbb{F}_{r_6}^n$ such that each x_i has at most l bits. (Naturally, $l \leq \lceil \log r_6 \rceil$.) Moreover, $C_{V,6}$ consists of two subcircuits, $C_{V,6}^{\text{offline}}$ and $C_{V,6}^{\text{online}}$, implementing V_6^{offline} and V_6^{online}, with sizes

$$23325 \quad \textit{and} \quad (10 \cdot l - 4) \cdot n + 59{,}856 \ .$$

(In, $C_{\mathsf{pcd},4}$ and $C_{\mathsf{pcd},6}$, we set $(n, l) = (2, \lfloor \log r_6 \rfloor)$ and $(n, l) = (1, \lceil \log r_4 \rceil)$ for $C_{V,6}$ and $C_{V,4}$, respectively. Also, in Lemma 5.1, the reported size for $C_{V,4}^{\text{online}}$ is even smaller than $(10 \cdot \lceil \log r_4 \rceil - 4) \cdot 1 + 31{,}497$ because we hardcode the processed verification key pvk_4 into $C_{V,4}^{\text{online}}$, which provides additional savings.)

The zk-SNARK verifier protocol. The protocol of the zk-SNARK verifier V (recalled in Appendix C.3) consists of two parts: (a) use the verification key vk and input $\vec{x} \in \mathbb{F}_r^n$ to compute an element $\mathsf{vk}_{\vec{x}} \in \mathbb{G}_1$; and (b) use the verification key vk, element $\mathsf{vk}_{\vec{x}}$, and proof π, to compute 12 pairings for the required checks.

Prior techniques for fast program execution of V. The first part of V requires $O(n)$ scalar multiplications in $\mathbb{G}_1$, and can be efficiently performed via a suitable choice of variable-base multi-scalar multiplication techniques. The second part dominates V's efficiency for small n, and an efficient implementation is algorithmically more complex. Ben-Sasson at al. [BCTV14] address this second part by (i) obtaining optimized implementations of sub-components of a pairing, and then (ii) combining these in a way that is tailored to V's protocol. In short, after breaking a pairing into its two main parts, the *Miller loop* and the *final exponentiation*, and implementing both (using optimal pairings [Ver10] and other methods [SBCDPK09, GS10, KKC13]), they apply precomputation techniques to the verification key [GHS02, BLS03, Sco07] and share subcomputations of the Miller loop and final exponentiations across V's different pairing evaluations [Sol03, Sco05, GS06, Sco07].

Our techniques for fast circuit verification of V. The high-level structure of our construction of $C_{V,4}$ and $C_{V,6}$ mirrors that of our software implementation of V_4 and V_6, itself based on techniques from [BCTV14]. Namely, both $C_{V,4}$ and $C_{V,6}$ also break an (optimal) pairing into a Miller loop and final exponentiation, and combine these components in a way that is tailored to the verifier protocol.

However, our construction differs in how these two components are implemented, especially with regard to the Miller loop. This is because, in our setting, two main operations come "for free": (a) field operations over the circuit's field, and (b) nondeterministic guessing (i.e., auxiliary advice). In particular, *field divisions cost the same as field multiplications* (since we can guess the answer and check it).

Traditional software implemantions go to great lengths to avoid expensive field divisions (e.g., by use of projective coordinates instead of affine ones, and in "addition" and "doubling" steps in the Miller loop). By contrast, both $C_{V,4}$ and $C_{V,6}$ perform the Miller loop by using *affine* coordinates for both curve arithmetic and divisor evaluations [LMN10], which can be done very efficiently by nondeterministic arithmetic circuits.

Moreover, sharing Miller loop subcomputation traditionally only applies to products of pairings, of which there are only two in the verifier. Instead, in our setting, such techniques extend to *ratios of products* of pairings, and can thus be applied to *every* pairing check in the verifier, to further improve efficiency.

Overall, in our software implementation, the number of field multiplications used to *compute* the checks of V_4, V_6 is $3.8\times, 3.2\times$ more than the number of those used by $C_{V,4}, C_{V,6}$ to *verify* them, respectively.

5.2 Arithmetic circuits for collision-resistant hashing

We also require arithmetic circuits for hashing: an $\mathbb{F}_{r_4}$-arithmetic circuit $C_{H,4}$ for a collision-resistant function $H_4 \colon \{0,1\}^{m_{H,4}} \to \mathbb{F}_{r_4}^{d_{H,4}}$ such that $m_{H,4} \geq \left(\ell_{\mathsf{vk},6}(\lceil \frac{d_{H,4} \cdot \lceil \log r_4 \rceil}{\lfloor \log r_6 \rfloor} \rceil) + n_{\mathsf{msg}}\right) \cdot \lceil \log r_4 \rceil$; indeed, $C_{\mathsf{pcd},\alpha}$ uses H_4 to hash (the binary representation of) both the verification key vk_6 and a message z.

We base collision-resistant hashing on subset-sum functions [Ajt96, GGH96], chosen to have an especially compact representation as arithmetic circuits over the zk-SNARK's "native field".

Subset sums. For $p, d, m \in \mathbb{N}$ with p prime and $M \in \mathbb{Z}_p^{d \times m}$, the subset-sum function $H_M \colon \{0,1\}^m \to \mathbb{Z}_p^d$ maps an m-bit string x to $\sum_{i=1}^{m} x_i M(i)$, where $M(i)$ is the i-th column of M.[17] Designing and constructing an $\mathbb{F}_p$-arithmetic circuit that verifies H_M is straightforward, and only requires d gates (the j-th gate computes the j-th linear combination). The entries of M should be drawn at random. (To remove suspicion of trapdoors, M can be chosen, e.g., according to the digits of π.)

Parameters. We set $H_4 := H_{M_4}$ for a random matrix $M_4 \in \mathbb{Z}_{r_4}^{d_{H,4} \times m_{H,4}}$ and integers $d_{H,4}, m_{H,4} \in \mathbb{N}$. We have fixed the prime of the subset sum to be r_4; this ensures that $C_{H,4}$, which is defined over $\mathbb{F}_{r_4}$, works over the correct ring, and only requires $d_{H,4}$ gates. Next, for any given dimension $d_{H,4}$ and PCD message length n_{msg}, we set the input length to $m_{H,4} := \left(\ell_{\mathsf{vk},6}(\lceil \frac{d_{H,4} \cdot \lceil \log r_4 \rceil}{\lfloor \log r_6 \rfloor} \rceil) + n_{\mathsf{msg}}\right) \cdot \lceil \log r_4 \rceil$, to ensure the aforementioned condition on the input and output lengths. There remains to fix the output length $d_{H,4}$. This is delicate, because it affects security (and recall we aim at 80-bit security). Since r_4 is a 298-bit prime, it appears heuristically sufficient to fix $d_{H,4} = 1$ [JJ98].[18] In particular, this yields $|C_{H,4}| = 1$.

Remark 5.4 (boolean input)**.** Ideally, we would like a collision-resistant function whose "natural" domain is strings of $\mathbb{F}_{r_4}$-elements, rather than strings of bits (as in subset-sum functions). Indeed, converting a string $x \in \mathbb{F}_p^m$ to its binary representation $s \in \{0,1\}^{m \cdot \lceil \log p \rceil}$ (in order to "prepare" the function's input) costs $m \cdot \lceil \log p \rceil$ gates, which is a nontrivial contribution to the size of $C_{\mathsf{pcd},\alpha}$ unless one keeps m quite small (see Section 4). While a subset-sum function $H_M \colon \{0,1\}^m \to \mathbb{Z}_p^d$ continues to remain collision-resistant even for domains consisting of "small-norm" vectors (of which binary strings are a special case), H_M is *not* collision-resistant (or even one-way) when the domain is enlarged to include all elements in $\mathbb{Z}_p^m$ (simply because, being a linear function, it can be efficiently inverted). It is an open question whether the cost of converting to binary strings can be avoided, via some other choice of hash function.

6 Scalable zk-SNARKs

Having constructed a PCD system (see Section 4 and Section 5), we use it to obtain a new zk-SNARK that is *scalable* (i.e., fully succinct and incrementally computable).

6.1 Specifying a machine

A notable feature of our zk-SNARK is generality: it can prove/verify correctness of executions on any given random-access machine $\mathbf{M}$, specified by a memory configuration and a corresponding CPU circuit. For instance, $\mathbf{M}$ may encode a floating-point-arithmetic processor for running quantitative analysis programs; or, $\mathbf{M}$ may encode a SIMD-based architecture for running multimedia programs.

Parameters. More precisely, a machine $\mathbf{M}$ is specified by a tuple $(A, W, N, \mathrm{CPU}_{\mathsf{exe}}, \mathrm{CPU}_{\mathsf{ver}})$ where:

- $A, W \in \mathbb{N}$ specify that (random-access) memory contains A addresses each storing W bits, i.e., that memory is a function $\mathcal{M} \colon [A] \to \{0,1\}^W$;
- $N \in \mathbb{N}$ specifies the length, in bits, of a CPU state;
- $\mathrm{CPU}_{\mathsf{exe}}$ is a (stateful) function for *executing* the CPU;
- $\mathrm{CPU}_{\mathsf{ver}}$ is an $\mathbb{F}$-arithmetic circuit for *verifying* the CPU's execution.

We now elaborate on the above parameters. For more details, see Appendix A.2.

[17] We do not require the hash function to be universal, so we do not need to add a random vector to the subset sum.

[18] Recent works [LM06, PR06, LMPR08, ADLM+08, BLPRS13] use a small modulus and larger dimension, but our "native" modulus is already a large one.

Execution on $\mathbf{M}$. A computation on $\mathbf{M}$ proceeds in steps, as determined by $\mathrm{CPU}_{\mathsf{exe}}$, which can be thought of as $\mathbf{M}$'s "processor": step after step, $\mathrm{CPU}_{\mathsf{exe}}$ takes the previous state and instruction (and its address), executes the instruction, communicates with random-access memory, and produces the next state and instruction address. More precisely, each step consists of two phases:

- *Instruction fetch.* Given the current CPU state $s_{\mathsf{cpu}} \in \{0,1\}^N$ and address $a_{\mathsf{pc}} \in [A]$ of the instruction to be executed, the new instruction to be executed is fetched: $v_{\mathsf{pc}} := \mathcal{M}(a_{\mathsf{pc}}) \in \{0,1\}^W$.
- *Instruction execution.* For an auxiliary input $g \in \{0,1\}^W$, $\mathrm{CPU}_{\mathsf{exe}}$ receives $(s_{\mathsf{cpu}}, a_{\mathsf{pc}}, v_{\mathsf{pc}}, g)$ and outputs $(a_{\mathsf{mem}}, v_{\mathsf{st}}, f_{\mathsf{st}})$, where $a_{\mathsf{mem}} \in [A]$ is an address, $v_{\mathsf{st}} \in \{0,1\}^W$ a value, and $f_{\mathsf{st}} \in \{0,1\}$ a store flag. Afterwards, $\mathrm{CPU}_{\mathsf{exe}}$ receives $v_{\mathsf{ld}} := \mathcal{M}(a_{\mathsf{mem}}) \in \{0,1\}^W$ (i.e., the value at the address) and outputs a new CPU state $s'_{\mathsf{cpu}} \in \{0,1\}^N$, an address $a'_{\mathsf{pc}} \in [A]$ for the next instruction, and a flag $f'_{\mathsf{acc}} \in \{0,1\}$ denoting whether the machine has accepted. Meanwhile, if a store was requested, it is performed: if $f_{\mathsf{st}} = 1$ then $\mathcal{M}(a_{\mathsf{mem}}) := v_{\mathsf{st}}$. Finally, at the end of every step, $\mathrm{CPU}_{\mathsf{exe}}$'s state is reset.

See Figure 3 for a diagram of these two phases.

Verification of the CPU. The circuit $\mathrm{CPU}_{\mathsf{ver}}$ verifies the correct input/output relationship of $\mathrm{CPU}_{\mathsf{exe}}$ (but *not* memory consistency). In other words $\mathrm{CPU}_{\mathsf{exe}}$ satisfies the following property:

> Fix $s_{\mathsf{cpu}}, s'_{\mathsf{cpu}} \in \{0,1\}^N$, $a_{\mathsf{pc}}, a_{\mathsf{mem}}, a'_{\mathsf{pc}} \in [A]$, $v_{\mathsf{pc}}, v_{\mathsf{st}}, v_{\mathsf{ld}}, g \in \{0,1\}^W$, $f_{\mathsf{st}}, f'_{\mathsf{acc}} \in \{0,1\}$, and let x_{ver} be the concatenation of all these. There is a witness a_{ver} such that $\mathrm{CPU}_{\mathsf{ver}}(x_{\mathsf{ver}}, a_{\mathsf{ver}}) = 0$ iff $(a_{\mathsf{mem}}, v_{\mathsf{st}}, f_{\mathsf{st}}) \leftarrow \mathrm{CPU}_{\mathsf{exe}}(s_{\mathsf{cpu}}, a_{\mathsf{pc}}, v_{\mathsf{pc}}, g)$ and, afterwards, $(s'_{\mathsf{cpu}}, a'_{\mathsf{pc}}, f'_{\mathsf{acc}}) \leftarrow \mathrm{CPU}_{\mathsf{exe}}(v_{\mathsf{ld}})$. Moreover, a_{ver} can be efficiently computed from x_{ver}.

While we do not care about how the function $\mathrm{CPU}_{\mathsf{exe}}$ is specified (e.g., it can be a computed program), $\mathrm{CPU}_{\mathsf{ver}}$ must be an arithmetic circuit; if $\mathrm{CPU}_{\mathsf{ver}}$ is defined over $\mathbb{F}$, we say that $\mathbf{M}$ has *verification over* $\mathbb{F}$.

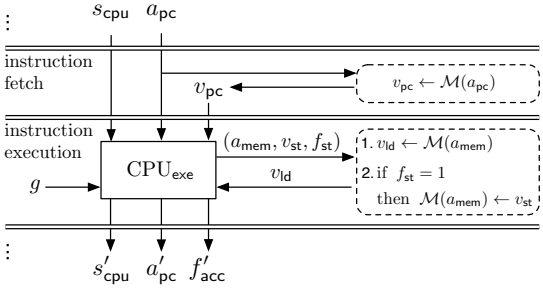
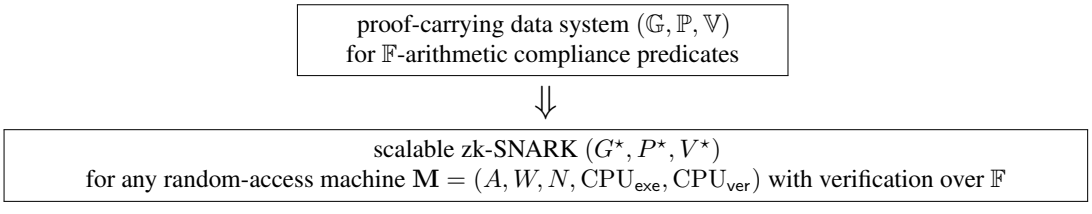


Figure 3: The two phases in one step of execution of a random-access machine $\mathbf{M}$.

6.2 Construction summary

The construction of the new zk-SNARK consists of the following transformation:



The transformation's outline is as follows (see Section 2.3). First, given $\mathbf{M}$, we design a compliance predicate $\Pi_{\mathbf{M},H}$ for the incremental verification of $\mathbf{M}$'s execution, when its random-access memory $\mathcal{M}$ is delegated via memory-checking techniques based on a collision-resistant hash H [BEGKN91, BCGT13a]. Then, we use the PCD system $(\mathbb{G}, \mathbb{P}, \mathbb{V})$ to enforce the compliance predicate $\Pi_{\mathbf{M},H}$, and thereby construct the algorithms zk-SNARK $(G^\star, P^\star, V^\star)$ of the new zk-SNARK, which is fully-succinct and incrementally-computable.

For the first part, we again use field-specific subset-sum functions for constructing circuits that verify authentication paths (Section 6.3), and then combine these, together with $\mathbf{M}$'s CPU circuit, to construct $\Pi_{\mathbf{M},H}$ (Section 6.4). For the second part, the construction of the new zk-SNARK's three algorithms is fairly straightforward in light of previous work, and we include its details for completeness (Section 6.5).

Later, in Section 7, we evaluate our scalable zk-SNARK when the machine $\mathbf{M}$ equals vnTinyRAM.

6.3 Arithmetic circuits for secure loads and stores

We construct arithmetic circuits for checking loads/stores of an untrusted random-access memory, relative to a (trusted) root of a Merkle tree over the memory; this task is known as *memory checking* (see Remark 6.1).

Let $A, W \in \mathbb{N}$ specify that memory contains A addresses each storing W bits, i.e., that memory is a function $\mathcal{M}\colon [A] \to \{0,1\}^W$. Let $H\colon \{0,1\}^m \to \{0,1\}^\ell$ be a collision-resistant function suitable for building binary Merkle trees over $\mathcal{M}$ (i.e., $m \geq W$ and $m/\ell \geq 2$); we say that H is (A, W)**-good**. For a field $\mathbb{F}$, let C_H be an $\mathbb{F}$-arithmetic circuit that verifies H; we construct the following two $\mathbb{F}$-arithmetic circuits.

Secure load. A *secure-load* circuit C_{SecLd} that, for a given address a, checks the validity of a loaded value v against a Merkle-tree root ρ. More precisely, the circuit C_{SecLd} satisfies the following property: for any root $\rho \in \{0,1\}^\ell$, address $a \in [A]$, value $v \in \{0,1\}^W$, and authentication path $\mathbf{p} \in \{0,1\}^{W+(\lceil \log A \rceil - 1)\ell}$, $C_{\mathsf{SecLd}}(\rho, a, v, \mathbf{p}) = 0$ if and only if $\mathbf{p}$ is a valid authentication path for the value v as the a-th leaf in a Merkle tree of root ρ. One can verify that the size of such a circuit is $\lceil \log A \rceil \cdot (|C_H| + 2\ell)$, because the check can be performed via $\lceil \log A \rceil$ invocations of H plus 2ℓ gates per level.[19]

Secure load-then-store. A *secure-load-then-store* circuit C_{SecLdSt} that, for a given address a, checks: (i) the validity of a loaded value v_{ld} against a Merkle-tree root ρ; and (ii) the validity of storing v_{st}, to the same address, against a (possibly different) Merkle-tree root ρ'. More precisely, the circuit C_{SecLdSt} satisfies the following property: for any two roots $\rho, \rho' \in \{0,1\}^\ell$, address $a \in [A]$, two values $v_{\mathsf{ld}}, v_{\mathsf{st}} \in \{0,1\}^W$, and authentication path $\mathbf{p} \in \{0,1\}^{W+(\lceil \log A \rceil - 1)\ell}$, $C_{\mathsf{SecLdSt}}(\rho, \rho', a, v_{\mathsf{ld}}, v_{\mathsf{st}}, \mathbf{p}) = 0$ if and only if:

- $\mathbf{p}$ is a valid authentication path for the value v_{ld} as the a-th leaf in a Merkle tree of root ρ, AND
- $\mathbf{p}$ is a valid authentication path for the value v_{st} as the a-th leaf in a Merkle tree of root ρ'.

One can verify that the size of such a circuit is $\lceil \log A \rceil \cdot (2|C_H| + 4\ell)$.

Instantiation with subset-sum functions. We are left to choose the function H and construct C_H, required to obtain the two circuits C_{SecLd} and C_{SecLdSt}.

As in Section 5.2, subset-sum functions are a natural candidate, for efficiency considerations. Namely, since $\mathbb{F}$ has prime order p, its additive group is isomorphic to $\mathbb{Z}_p$; hence, for $M \in \mathbb{Z}_p^{d \times m}$, the subset-sum function $H_M\colon \{0,1\}^m \to \mathbb{Z}_p^d$ can be computed with only d gates over $\mathbb{F}$.

Unlike Section 5.2, however, both C_{SecLd} and C_{SecLdSt} require H's outputs to be inputs to other invocations of H. Thus, here we treat a subset-sum function as having binary output: $H_M\colon \{0,1\}^m \to \{0,1\}^\ell$ where $\ell := d \cdot \lceil \log p \rceil$. Doing so requires additional gates, summing up to a total of $d + \ell = d \cdot (1 + \lceil \log p \rceil)$ gates over $\mathbb{F}$ to compute H_M. Moreover, the condition on input length is different: here we need to ensure that the function is (A, W)-good, which requires that $m \geq \max\{W, 2\ell\} = \max\{W, 2d\lceil \log p \rceil\}$.

Overall, if we set $H := H_M$ (for a random M), we can achieve circuit sizes that are:

[19] Specifically, in the i-th level, ℓ gates ensure booleanity of the i-th chunk of the authentication path $\mathbf{p}$, while another ℓ gates prepare the correct input to the next invocation of H, depending on the i-th address bit.

$$|C_{\mathsf{SecLd}}| = \lceil \log A \rceil \cdot d \cdot \big(1 + 3\lceil \log p \rceil\big) \text{ and } |C_{\mathsf{SecLdSt}}| = \lceil \log A \rceil \cdot d \cdot \big(2 + 6\lceil \log p \rceil\big).$$

In terms of concrete numbers, recalling from Section 5.2 that $d = 1$ and $\lceil \log p \rceil = 298$, we get that

$$|C_{\mathsf{SecLd}}| = \lceil \log A \rceil \cdot 895 \text{ and } |C_{\mathsf{SecLdSt}}| = \lceil \log A \rceil \cdot 1{,}790.$$

Remark 6.1 (memory checking)**.** *Memory checking* was introduced by Blum et al. [BEGKN91]; they showed how to use Merkle hashing to delegate a machine's memory to an untrusted storage, and dynamically verify its consistency using only a small $\mathrm{poly}(\lambda)$-size "trusted" memory.

Blum et al. instantiated Merkle hashing with *universal one-way hash functions* [NY89, Rom90]. Yet, in general, a machine's computation includes a "nondeterministic component", e.g., an auxiliary input. In such a case (as in this paper), Merkle hashing must be based on hash functions that are *collision resistant*.

Memory checking techniques have found numerous practical applications for securing untrusted storages [MVS00, MS01, GSCvDD03, GSMB03, KRSWF03]. Ben-Sasson et al. [BCGT13a] suggested that verification of memory via Merkle hashing can be a useful computational alternative to the information-theoretic use of nondeterministic routing for efficient circuit generators. For instance, the recent circuit generator of Braun et al. [BFRS$^+$13] uses memory checking to verify accesses to an untrusted storage.

6.4 The RAM compliance predicate

Given a random-access machine $\mathbf{M}$ and a (suitable) collision-resistant function H, we construct a compliance predicate $\Pi_{\mathbf{M},H}$ that checks a step of execution of $\mathbf{M}$. The transformation is:

random-access machine $\mathbf{M} = (A, W, N, \mathrm{CPU}_{\mathsf{exe}}, \mathrm{CPU}_{\mathsf{ver}})$	+	(A, W)-good collision-resistant function $H\colon \{0,1\}^m \to \{0,1\}^\ell$	$\Rightarrow$	predicate $\Pi_{\mathbf{M},H}$

Briefly, a message z for $\Pi_{\mathbf{M},H}$ encodes a short representation of $\mathbf{M}$'s state at a given time step. Then, at a node with input message z_{in} and output message z_{out}, the compliance predicate $\Pi_{\mathbf{M},H}$ checks that the transition from the state in z_{in} to the state in z_{out} is a valid transition of the machine $\mathbf{M}$.

Below, we make this plan more concrete by describing the format of messages and local data for $\Pi_{\mathbf{M},H}$, and by describing the checks performed by $\Pi_{\mathbf{M},H}$. See Figure 5 for details (and Appendix A.2 as reference).

Format of messages. A message z summarizes $\mathbf{M}$'s entire state at a given time t, by storing the following:

- a timestamp t, denoting how many computation steps have occurred;
- a root ρ of a Merkle tree of random-access memory (after t computation steps);
- a CPU state s_{cpu} (after t computation steps); and
- a flag f_{acc} denoting whether the machine has accepted (after t computation steps).

Furthermore, z also stores ρ_0, the root of a Merkle tree over initial memory, so to "remember" $\mathbf{M}$'s input. Note that a message z is short because the large memory is "summarized" by the short root of a Merkle tree.

Format of local data. Now consider a node with input message z_{in} and output message z_{out}. The goal of $\Pi_{\mathbf{M},H}$ is to verify $\mathbf{M}$'s transition from z_{in} to z_{out}, using two main tools:

- $\mathrm{CPU}_{\mathsf{ver}}$ for checking CPU transitions, *given* consistent memory accesses ("what you store is what you get");
- C_{SecLd} and C_{SecLdSt} for checking memory accesses.

Thus, in the local data z_{loc} provided at a node, we store whatever auxiliary information is needed by $\Pi_{\mathbf{M},H}$ to evaluate $\mathrm{CPU}_{\mathsf{ver}}$ (e.g., requested memory addresses, memory values, flags, etc.) and C_{SecLd} and C_{SecLdSt} (e.g., addresses, values, and authentication paths). Furthermore, z_{loc} includes a flag f_{halt} specifying whether the computation should halt or not (as, in such a case, $\Pi_{\mathbf{M},H}$ will perform a different set of checks).

Construction. The compliance predicate $\Pi_{\mathbf{M},H}$ takes an input $(z_{\mathsf{out}}, z_{\mathsf{loc}}, z_{\mathsf{in}}, b_{\mathsf{base}})$, where z_{out} is the outgoing message, z_{loc} the local data, z_{in} the incoming message, and b_{base} the base-case flag, and must verify $\mathbf{M}$'s transition from z_{in} to z_{out}. (Thus, $\Pi_{\mathbf{M},H}$ has arity $s = 1$.) Our construction of $\Pi_{\mathbf{M},H}$ goes as follows.

In the base case (i.e., $b_{\mathsf{base}} = 1$), $\Pi_{\mathbf{M},H}$ ensures that z_{in} is correctly initialized: its timestamp, CPU state, instruction address, accept flag should all be set to zero; $\Pi_{\mathbf{M},H}$ also checks that the root of the Merkle tree of memory is equal to that of the Merkle tree of initial memory.

Moreover, regardless of base case or not, $\Pi_{\mathbf{M},H}$ always checks that the root of the Merkle tree of initial memory is preserved from z_{in} to z_{out}, in order to not "forget" what the initial state of the machine was.

When the computation does not halt (i.e., $f_{\mathsf{halt}} = 0$), $\Pi_{\mathbf{M},H}$ checks that the timestamp is incremented by 1 and that $\mathrm{CPU}_{\mathsf{ver}}$ (on the appropriate inputs) accepts; furthermore, it uses C_{SecLd} to check that the instruction was correctly loaded and C_{SecLdSt} to check that the memory access (a load or a store) was correctly performed.

When the computation does halt (i.e., $f_{\mathsf{halt}} = 1$), $\Pi_{\mathbf{M},H}$ first of all ensures that the computation has in fact accepted so far; then it clears out the root of the Merkle tree over memory and the CPU state (as these may leak information about the private auxiliary input) and ensures that the time step in z_{out} is at least as large as the number of steps so far. (Again for privacy reasons, $\Pi_{\mathbf{M},H}$ does not force z_{out} to carry the exact number of computation steps, but only a number that is at least that much.)

Overall, the above checks suffice for $\Pi_{\mathbf{M},H}$ to ensure that any $\Pi_{\mathbf{M},H}$-compliant distributed computation corresponds to correctly initializing, stepping through, and halting an accepting computation of $\mathbf{M}$.

Efficiency. By implementing $\Pi_{\mathbf{M},H}$ as an $\mathbb{F}$-arithmetic circuit, we obtain the following efficiency:

$$|\Pi_{\mathbf{M},H}| = |\mathrm{CPU}_{\mathsf{ver}}| + |C_{\mathsf{SecLd}}| + |C_{\mathsf{SecLdSt}}| + \varepsilon \ ,$$

where ε is a "small but ugly" term, depending on $d, N, \mathbb{F}$, that can be upper bounded as follows

$$\varepsilon \leq 2 \cdot \left(301 + 4N + 2\ell + \left\lceil \frac{301 + 4N + 2\ell}{\lfloor \log |\mathbb{F}| \rfloor} \right\rceil\right) + 24 \left\lceil \frac{N}{\lfloor \log |\mathbb{F}| \rfloor} \right\rceil + 2N + 12 \left\lceil \frac{\ell}{\lfloor \log |\mathbb{F}| \rfloor} \right\rceil + \ell + 10 \ .$$

Crucially, $|\Pi_{\mathbf{M},H}|$ only depends (nicely) on $\mathbf{M}$ and H, but is independent of the computation length on $\mathbf{M}$: the term $|\mathrm{CPU}_{\mathsf{ver}}|$ is the cost of verifying $\mathbf{M}$'s CPU (and depends on "how complex" is the CPU); while the term $|C_{\mathsf{SecLd}}| + |C_{\mathsf{SecLdSt}}| = \lceil \log A \rceil \cdot d \cdot (3 + 9\lceil \log p \rceil)$ is the per-cycle cost to ensure memory consistency via collision-resistant hashing (see Section 6.3).

In Section 7, we consider the case when $\mathbf{M}$ equals vnTinyRAM (a simple RISC von Neumann machine), with wordsizes $w \in \{16, 32\}$ and $k = 16$ registers. In Figure 4 we report, for these cases, the size of $\Pi_{\mathbf{M},H}$, its sub-circuits, and the resulting PCD circuits $C_{\mathsf{pcd},4}$ and $C_{\mathsf{pcd},6}$ (which affect the PCD system's efficiency).

	16-bit vnTinyRAM $(w,k) = (16,16)$	32-bit vnTinyRAM $(w,k) = (32,16)$
$\lvert\mathrm{CPU}_{\mathsf{ver}}\rvert$	766	1,108
$\lvert C_{\mathsf{SecLd}}\rvert$	12,530	25,955
$\lvert C_{\mathsf{SecLdSt}}\rvert$	25,060	51,910
ε	3501	4867
$\lvert\Pi_{\mathbf{M},H}\rvert$	41,857	83,840
$\lvert C_{\mathsf{pcd},4}\rvert$	146,174	189,349
$\lvert C_{\mathsf{pcd},6}\rvert$	32,027	32,027

Figure 4: Sizes of the compliance predicate $\Pi_{\mathbf{M},H}$ (and its sub-circuits) and the corresponding PCD circuits, for 16-bit and 32-bit vnTinyRAM.

Remark 6.2. The per-cycle cost of ensuring memory consistency via collision-resistant hashing (i.e., $|C_{\mathsf{SecLd}}| + |C_{\mathsf{SecLdSt}}|$) is typically much larger than that incurred when using nondeterministic routing (in [BCTV14], it is less than 1000). However, collision-resistant hashing ultimately enables scalability, whereas nondeterministic routing is not known to be useful for scalability (also see Section 8).

In particular, while in Section 7 we focus on the case $\mathbf{M} = \mathsf{vnTinyRAM}$, for which $|\mathrm{CPU}_{\mathsf{ver}}| \approx 10^3$, we could have chosen more complex machines. Indeed, even if $\mathrm{CPU}_{\mathsf{ver}}$ had a few tens of thousands of gates, the size of $\Pi_{\mathbf{M},H}$ would remain on the order of 10^5 gates. In other words, our scalable zk-SNARK can accommodate much more complex machines at a relatively small additional cost.

Format of messages for $\Pi_{\mathbf{M},H}$. A message z for the compliance predicate $\Pi_{\mathbf{M},H}$ is a tuple

$$z = (\rho_0, t, \rho, s_{\mathsf{cpu}}, f_{\mathsf{acc}})$$

where:

- $\rho_0 \in \{0,1\}^\ell$ is an output of H; allegedly, it is the root of a Merkle tree whose leaves are a program $\mathcal{P}$ (i.e., initial memory).
- $t \in \{0,1\}^{300}$ is a timestamp; allegedly, it is the number of computation steps so far.
  (For concreteness, we bound all computations to 2^{300} steps, which is good enough for a long while.)
- $\rho \in \{0,1\}^\ell$ is an output of H; allegedly, it is the root of a Merkle tree whose leaves are $\mathcal{M}_t$ (memory after t steps).
- $s_{\mathsf{cpu}} \in \{0,1\}^N$ is a CPU state; allegedly, it is the machine's CPU state after t steps of computation.
- $f_{\mathsf{acc}} \in \{0,1\}$ is a flag which denotes whether the machine has accepted so far or not.

The length n_{msg} of a message is equal to $2\ell + 300 + N + 1$.

Format of local data for $\Pi_{\mathbf{M},H}$. Local data z_{loc} for the compliance predicate $\Pi_{\mathbf{M},H}$ is a tuple

$$z_{\mathsf{loc}} = (a_{\mathsf{pc}}, a_{\mathsf{mem}}, a'_{\mathsf{pc}}, v_{\mathsf{pc}}, v_{\mathsf{st}}, v_{\mathsf{ld}}, g, f_{\mathsf{st}}, f_{\mathsf{halt}}, a_{\mathsf{ver}}, \mathbf{p}_{\mathsf{pc}}, \mathbf{p}_{\mathsf{mem}})$$

where:

- $a_{\mathsf{pc}}, a_{\mathsf{mem}}, a'_{\mathsf{pc}} \in [A]$ are memory addresses.
- $v_{\mathsf{pc}}, v_{\mathsf{st}}, v_{\mathsf{ld}} \in \{0,1\}^W$ are memory values.
- $g \in \{0,1\}^W$ is a non-deterministic guess.
- $f_{\mathsf{st}}, f_{\mathsf{halt}} \in \{0,1\}$ are flags.
- a_{ver} is a witness for the $\mathbb{F}$-arithmetic circuit $\mathrm{CPU}_{\mathsf{ver}}$.
- $\mathbf{p}_{\mathsf{pc}}, \mathbf{p}_{\mathsf{mem}} \in \{0,1\}^{W+(\lceil \log A \rceil - 1)\ell}$ are authentication paths for Merkle trees over memory.

The length n_{loc} of local data is equal to $(3+2\ell) \cdot \lceil \log A \rceil + 6W + 2 - 2\ell + |a_{\mathsf{ver}}|$.

Compliance predicate $\Pi_{\mathbf{M},H}$.

- INPUTS:
  - output PCD message $z_{\mathsf{out}} = (\rho'_0, t', \rho', s'_{\mathsf{cpu}}, f'_{\mathsf{acc}}) \in \{0,1\}^{n_{\mathsf{msg}}}$
  - local data $z_{\mathsf{loc}} = (a_{\mathsf{pc}}, a_{\mathsf{mem}}, a'_{\mathsf{pc}}, v_{\mathsf{pc}}, v_{\mathsf{st}}, v_{\mathsf{ld}}, g, f_{\mathsf{st}}, f_{\mathsf{halt}}, a_{\mathsf{ver}}, \mathbf{p}_{\mathsf{pc}}, \mathbf{p}_{\mathsf{mem}}) \in \{0,1\}^{n_{\mathsf{loc}}}$
  - input PCD message $z_{\mathsf{in}} = (\rho_0, t, \rho, s_{\mathsf{cpu}}, f_{\mathsf{acc}}) \in \{0,1\}^{n_{\mathsf{msg}}}$
  - base case flag $b_{\mathsf{base}} \in \{0,1\}$
- OUTPUTS: 0 iff all checks passed

1. If $b_{\mathsf{base}} = 1$ (i.e., base case):
   (a) Check that $t = 0$, $s_{\mathsf{cpu}} = 0^N$, $a_{\mathsf{pc}} = 0$, $f_{\mathsf{acc}} = 0$.
   (b) Check that the root of the Merkle tree of current memory is correctly initialized: $\rho' = \rho_0$.
2. Check that the root of the Merkle tree of initial memory is copied over: $\rho'_0 = \rho_0$.
3. If $f_{\mathsf{halt}} = 0$ (i.e., do not halt):
   (a) Check that the timestamp is incremented: $t' = t + 1$.
   (b) Set $x_{\mathsf{ver}} := (s_{\mathsf{cpu}}, s'_{\mathsf{cpu}}, a_{\mathsf{pc}}, a_{\mathsf{mem}}, a'_{\mathsf{pc}}, v_{\mathsf{pc}}, v_{\mathsf{st}}, v_{\mathsf{ld}}, g, f_{\mathsf{st}}, f'_{\mathsf{acc}})$ and check that $\mathrm{CPU}_{\mathsf{ver}}(x_{\mathsf{ver}}, a_{\mathsf{ver}}) = 0$.
   (c) Check that the instruction is correctly loaded: $C_{\mathsf{SecLd}}(\rho, a_{\mathsf{pc}}, v_{\mathsf{pc}}, \mathbf{p}_{\mathsf{pc}}) = 0$.
   (d) If $f_{\mathsf{st}} = 0$ (i.e., no store), then check that $v_{\mathsf{ld}} = v_{\mathsf{st}}$.
   (e) Check that the load-then-store is correct: $C_{\mathsf{SecLdSt}}(\rho, \rho', a_{\mathsf{mem}}, v_{\mathsf{ld}}, v_{\mathsf{st}}, \mathbf{p}_{\mathsf{mem}}) = 0$.
4. If $f_{\mathsf{halt}} = 1$ (i.e., do halt):
   (a) Check that the machine has accepted: $f_{\mathsf{acc}} = 1$.
   (b) Check that the root and the CPU state have been cleared: $\rho' = 0^\ell$, $s'_{\mathsf{cpu}} = 0^N$.
   (c) Check that the timestamp is not less than the computation time: $t' \geq t$.
   (d) Check that the accept flag is copied over: $f'_{\mathsf{acc}} = f_{\mathsf{acc}}$.

Figure 5: Construction of the compliance predicate $\Pi_{\mathbf{M},H}$ from $\mathbf{M}$ and H.

6.5 The new zk-SNARK construction

We now explain how to use a PCD system $(\mathbb{G}, \mathbb{P}, \mathbb{V})$, invoked on $\Pi_{\mathbf{M},H}$, to construct a scalable zk-SNARK for random-access machines; see Figure 6 for the construction's pseudocode.

Construction of $G^\star$. The key generator $G^\star$ takes a random-access machine $\mathbf{M}$ as input, and must output a key pair that enables anyone to prove/verify correctness of computations on $\mathbf{M}$. Recall that the PCD generator $\mathbb{G}$ expects as input an $\mathbb{F}$-arithmetic compliance predicate. Thus, $G^\star$ constructs the $\mathbb{F}$-arithmetic compliance predicate $\Pi_{\mathbf{M},H}$, for a suitable choice of collision-resistant function H; it invokes $\mathbb{G}$ on $\Pi_{\mathbf{M},H}$ to generate a key pair $(\mathsf{pk}_{\mathsf{pcd}}, \mathsf{vk}_{\mathsf{pcd}})$; and, finally, it outputs $(\mathsf{pk}, \mathsf{vk})$ where $\mathsf{pk} := (\mathbf{M}, H, \mathsf{pk}_{\mathsf{pcd}})$ and $\mathsf{vk} := (\mathbf{M}, H, \mathsf{vk}_{\mathsf{pcd}})$.

More precisely, the key pair $(\mathsf{pk}, \mathsf{vk})$ allows to prove/verify membership of instances in the language $\mathcal{L}_{\mathbf{M}}$ of accepting computations on $\mathbf{M}$, i.e., the language consisting of pairs $(\mathcal{P}, T)$ such that: (i) $\mathcal{P}$ is a program for $\mathbf{M}$ (a program is just an initial memory state); (ii) T is a time bound; (iii) there exists an auxiliary input $\mathcal{G}$ such that $\mathbf{M}(\mathcal{P}; \mathcal{G})$ accepts in at most T steps. (See Definition A.2 for a more formal discussion of $\mathcal{L}_{\mathbf{M}}$.)

Construction of $P^\star$. The prover $P^\star$ takes as input a proving key pk, program $\mathcal{P}$, time bound T, and auxiliary input $\mathcal{G}$, and must output a proof π for the claim "$(\mathcal{P}, T) \in \mathcal{L}_{\mathbf{M}}$". Recall that the PCD prover $\mathbb{P}$ expects as input a proving key $\mathsf{pk}_{\mathsf{pcd}}$, output message z, local data z_{loc}, input message z_{in}, and (in the non-base case) also a corresponding proof π_{in}. Thus, $P^\star$ steps through the computation of $\mathbf{M}$ on $\mathcal{P}$, at each step generating a new message and proof, by using a previous message and proof; throughout, $P^\star$ maintains a Merkle tree over random-access memory. Concretely, at each step, $P^\star$ executes the CPU of $\mathbf{M}$, handles any memory loads or stores, and prepares the necessary $\Pi_{\mathbf{M},H}$-compliant inputs for $\mathbb{P}$ in order to compute the next proof; and then it continues to the next step. After T steps of computation, $P^\star$ produces a final proof, relative to a specially-constructed message z_{fin}, which clears from a message all but essential information (so not to compromise zero knowledge), and outputs the final proof.

Construction of $V^\star$. The verifier $V^\star$ takes as input a verification key vk, program $\mathcal{P}$, time bound T, and proof π , and must output a bit indicating whether π is a convincing proof for the claim "$(\mathcal{P}, T) \in \mathcal{L}_{\mathbf{M}}$". Recall that the PCD verifier $\mathbb{V}$ expects as input a verification key $\mathsf{vk}_{\mathsf{pcd}}$, message z, and proof π. Thus, $V^\star$ constructs a message z, corresponding to a "halted and accepted" computation of $\mathbf{M}$ on $\mathcal{P}$ (cf. the construction of $\Pi_{\mathbf{M},H}$), and then accepts if and only if $\mathbb{V}(\mathsf{pk}_{\mathsf{pcd}}, z, \pi)$ does.

Preparing the message z requires computing the root ρ_0 of a Merkle tree over the program $\mathcal{P}$.[20] In fact, computing ρ_0 can be done even before receiving the proof π, as it only requires knowledge of H and $\mathcal{P}$.

Security. Suppose that $V^\star$ accepts a proof π for an instance $(\mathcal{P}, T)$. Then the PCD verifier $\mathbb{V}$ accepted π for a message z, constructed from $\mathcal{P}$ and T, that corresponds to a "halted and accepted" computation. By construction of the compliance predicate $\Pi_{\mathbf{M},H}$ and the security of $\mathbb{V}$, we educe that $(\mathcal{P}, T) \in \mathcal{L}_{\mathbf{M}}$. (And proof of knowledge is inherited from the proof of knowledge of the PCD system.)

Scalability. From the above description (and the pseudocode in Figure 6), one can verify that $(G^\star, P^\star, V^\star)$ is both fully succinct and incrementally computable.

[20] Of course, to perform this step in time $O(|\mathcal{P}|)$, one should not explicitly build a Merkle tree over all of memory, but instead build the Merkle tree by considering only the *non-zero* memory entries specified by $\mathcal{P}$.

New zk-SNARK generator $G^\star$

- INPUTS: a random-access machine $\mathbf{M} = (A, W, N, \mathrm{CPU}_{\mathsf{exe}}, \mathrm{CPU}_{\mathsf{ver}})$ with verification over $\mathbb{F}$
- OUTPUTS: proving key pk and verification key vk

1. Select any (A, W)-good collision-resistant hash function H.
2. Construct C_H, an $\mathbb{F}$-arithmetic circuit implementing H.
3. Use $\mathbf{M}$ and C_H to construct the compliance predicate $\Pi_{\mathbf{M},H}$.
4. Compute $(\mathsf{pk}_{\mathsf{pcd}}, \mathsf{vk}_{\mathsf{pcd}}) := \mathbb{G}(\Pi_{\mathbf{M},H})$.
5. Set $\mathsf{pk} := (\mathbf{M}, H, \mathsf{pk}_{\mathsf{pcd}})$ and $\mathsf{vk} := (\mathbf{M}, H, \mathsf{vk}_{\mathsf{pcd}})$.
6. Output $(\mathsf{pk}, \mathsf{vk})$.

New zk-SNARK prover $P^\star$

- INPUTS: proving key pk, program $\mathcal{P}$, time bound T, and auxiliary input $\mathcal{G} = (g_0, g_1, \ldots, g_{T-1})$
- OUTPUTS: proof π for the instance $(\mathcal{P}, T)$

1. Use H to compute ρ_0, the root of the Merkle tree over $\mathcal{P}$.
2. Initialize memory to the given program: $\mathcal{M}_0 := \mathcal{P}$.
3. Initialize the CPU state and instruction address to zero: $s_{\mathsf{cpu},0} := 0^N$ and $a_{\mathsf{pc},0} := 0$.
4. Initialize the first message: $z_{\mathsf{msg},0} := (\rho_0, 0, \rho_0, s_{\mathsf{cpu},0}, 0)$.
5. Initialize the first proof to empty: $\pi_0 := \bot$.
6. For $i = 0, \ldots, T-1$, compute the next message $z_{\mathsf{msg},i+1}$ and proof π_{i+1} as follows:
   (a) Give $\mathrm{CPU}_{\mathsf{exe}}$
   - the current CPU state ($s_{\mathsf{cpu},i} \in \{0,1\}^N$),
   - address of the instruction to be executed ($a_{\mathsf{pc},i} \in [A]$),
   - instruction to be executed ($v_{\mathsf{pc},i} := \mathcal{M}_i(a_{\mathsf{pc},i}) \in \{0,1\}^W$), and
   - guess ($g_i \in \{0,1\}^W$).

   (b) Get from $\mathrm{CPU}_{\mathsf{exe}}$
   - an address ($a_{\mathsf{mem},i} \in [A]$),
   - value ($v_{\mathsf{st},i} \in \{0,1\}^W$), and
   - store flag ($f_{\mathsf{st},i} \in \{0,1\}$).

   (c) Give $\mathrm{CPU}_{\mathsf{exe}}$ the value at the address ($v_{\mathsf{ld},i} := \mathcal{M}_i(a_{\mathsf{mem},i}) \in \{0,1\}^W$).
   (d) Set $\mathcal{M}_{i+1}$ to equal $\mathcal{M}_i$; if a store was requested (i.e., $f_{\mathsf{st},i} = 1$), do it (i.e., $\mathcal{M}_{i+1}(a_{\mathsf{mem},i}) := v_{\mathsf{st},i}$).
   (e) Compute ρ_{i+1} (the root of the Merkle tree over $\mathcal{M}_{i+1}$) from ρ_i.
   (f) Get from $\mathrm{CPU}_{\mathsf{exe}}$
   - a new CPU state ($s_{\mathsf{cpu},i+1} \in \{0,1\}^N$),
   - an address for the next instruction ($a_{\mathsf{pc},i+1} \in [A]$), and
   - a flag denoting whether the machine has accepted ($f_{\mathsf{acc},i+1} \in \{0,1\}$).

   Reset $\mathrm{CPU}_{\mathsf{exe}}$'s state.
   (g) Create the next message: $z_{\mathsf{msg},i+1} := (\rho_0, i+1, \rho_{i+1}, s_{\mathsf{cpu},i+1}, f_{\mathsf{acc},i+1})$.
   (h) Deduce a_{ver} from $x_{\mathsf{ver}} := (s_{\mathsf{cpu},i}, s_{\mathsf{cpu},i+1}, a_{\mathsf{pc},i}, a_{\mathsf{mem},i}, a_{\mathsf{pc},i+1}, v_{\mathsf{pc},i}, v_{\mathsf{st},i}, v_{\mathsf{ld},i}, g_i, f_{\mathsf{st},i}, f_{\mathsf{acc},i+1})$.
   (i) Let $\mathbf{p}_{\mathsf{pc},i}$ (resp., $\mathbf{p}_{\mathsf{mem},i}$) be the authentication path for address $a_{\mathsf{pc},i}$ (resp., $a_{\mathsf{mem},i}$) in $\mathcal{M}_i$.
   (j) Create local data: $z_{\mathsf{loc},i+1} := (a_{\mathsf{pc},i}, a_{\mathsf{mem},i}, a_{\mathsf{pc},i+1}, v_{\mathsf{pc},i}, v_{\mathsf{st},i}, v_{\mathsf{ld},i}, g_i, f_{\mathsf{acc},i+1}, 0, a_{\mathsf{ver}}, \mathbf{p}_{\mathsf{pc},i}, \mathbf{p}_{\mathsf{mem},i})$.
   (k) Compute the next proof: $\pi_{i+1} := \mathbb{P}(\mathsf{pk}_{\mathsf{pcd}}, z_{\mathsf{msg},i+1}, z_{\mathsf{loc},i+1}, z_i, \pi_i)$.
7. Prepare the final message: $z_{\mathsf{msg,fin}} := (\rho_0, T, 0^\ell, 0^N, 1)$.
8. Prepare the final local data: $z_{\mathsf{loc,fin}} := (*, *, *, *, *, *, *, *, 1, *, *, *)$, where $*$ can be set to anything of the right length.
9. Compute the final proof: $\pi := \mathbb{P}(\mathsf{pk}_{\mathsf{pcd}}, z_{\mathsf{msg,fin}}, z_{\mathsf{loc,fin}}, z_T, \pi_T)$.
10. Output π.

New zk-SNARK verifier $V^\star$

- INPUTS: verification key vk, program $\mathcal{P}$, time bound T, and proof π
- OUTPUTS: decision bit

1. Use H to compute ρ_0, the root of the Merkle tree over $\mathcal{P}$. (This can also be done beforehand.)
2. Construct the message $z := (\rho_0, T, 0^\ell, 0^N, 1)$.
3. Accept if and only if $\mathbb{V}(\mathsf{vk}_{\mathsf{pcd}}, z, \pi) = 1$.

Figure 6: Construction of a scalable zk-SNARK for random-access machines.

7 Evaluation on vnTinyRAM

We evaluate our scalable zk-SNARK when the given random-access machine $\mathbf{M}$ equals vnTinyRAM, a simple RISC von Neumann architecture [BCTV14, BCGTV13b]. For comparison, we also compare [BCTV14]'s preprocessing zk-SNARK (which also supports vnTinyRAM) with our scalable zk-SNARK.

We ran our experiments on a desktop PC with a 3.40 GHz Intel Core i7-4770 CPU and 16 GB of RAM available. Unless otherwise specified, all times are in single-thread mode; as for our multi-core experiments, we enabled one thread for each of the CPU's 4 cores (for a total of 4 threads).

Recalling vnTinyRAM. The architecture vnTinyRAM is parametrized by the *word size*, denoted w, and the *number of registers*, denoted k. In terms of instructions, vnTinyRAM includes load and store instructions for accessing random-access memory (in byte or word blocks), as well as simple integer, shift, logical, compare, move, and jump instructions. Thus, vnTinyRAM can efficiently implement control flow, loops, subroutines, recursion, and so on. Complex instructions (e.g., floating-point arithmetic) are not directly supported and can be implemented "in software". See Appendix A.3 for how vnTinyRAM can be expressed in our random-access machine formalism (i.e., given w, k, how to construct $\mathbf{M}$ to express w-bit vnTinyRAM with k registers).

Costs on vnTinyRAM. The performance of our zk-SNARK $(G^\star, P^\star, V^\star)$ on vnTinyRAM is easy to characterize, because it is determined by few quantities. For the key generator $G^\star$, the relevant quantities are:
- the constant time and space complexity of $G^\star$, when given as input a description of vnTinyRAM; and
- the constant sizes of the generated proving key pk and verification key vk.

For the proving algorithm $P^\star$, which proceeds step by step alongside the original computation, they are:
- the constant time necessary to incrementally compute the new (constant-size) proof at each step; and
- the constant space needed to compute the new proof (on top of the space needed by the original program).[21]

Finally, the verifier $V^\star$ takes as input a program $\mathcal{P}$ and a time bound T, and runs in time $O(|\mathcal{P}| + \log T)$; in our implementation, we fix $T \leq 2^{300}$ (plenty enough), so that $V^\star$ runs in time $O(|\mathcal{P}|)$.

In Figure 7, we report our measurements for two settings of vnTinyRAM: $(w, k) = (16, 16)$ and $(w, k) = (32, 16)$, i.e., 16-bit and 32-bit vnTinyRAM with 16 registers. (The same settings as in [BCTV14].)

Comparison with [BCTV14]. In Figure 8, we compare the efficiency of [BCTV14]'s preprocessing zk-SNARK and our scalable zk-SNARK, for a (random) program $\mathcal{P}$ of 10^4 instructions, as a function of T (the number of vnTinyRAM computation steps).

The (approximate) asymptotic efficiency for [BCTV14] was obtained by linearly interpolating [BCTV14]'s measurements (which were collected on a machine with similar characteristics as our benchmarking machine). As for our measurements, we use the relevant numbers from Figure 7.

Conclusion. Our experiments demonstrate that, as expected, our approach is slower for small computations but, on the other hand, offers scalability to large computations by avoiding any space-intensive computations.

Indeed, [BCTV14] (as well as other preprocessing zk-SNARK implementations [PGHR13, BCGTV13a]) require space-intensive computations to maintain their efficiency. As T grows, such approaches simply run out of memory, and must resort to "computing in blocks", sacrificing time complexity.[22]

In contrast, our zk-SNARK, while requiring more time per execution step, merely requires a constant amount of memory to prove any number of execution steps. In particular, our zk-SNARK becomes more space-efficient than [BCTV14]'s zk-SNARK when $T > 422$ for 16-bit vnTinyRAM, and when $T > 321$ for 32-bit vnTinyRAM; moreover, these savings in space grow unbounded as T increases.

[21] The prover also needs to store the Merkle tree's intermediate hashes, which incurs a linear overhead in the program's space complexity. Since this overhead is small, and can even be reduced by saving only the high levels of the Merkle tree (and recomputing, "on demand", the local neighborhood of accessed leaves), we focus on the fixed additive overhead needed to generate the proof.

[22] Extending known preprocessing zk-SNARK implementations with block-computing techniques, and precisely quantifying their cost, remains a challenging open question that we leave to future work.

	16-bit vnTinyRAM $(w,k)=(16,16)$		32-bit vnTinyRAM $(w,k)=(32,16)$	
key generator $G^\star$				
	TIME			
	1 thread	4 threads	1 thread	4 threads
total	33.8 s	12.9 s	42.0 s	15.5 s
	SPACE			
memory	861 MB	1,125 MB	1,068 MB	1,343 MB
pk size	43 MB		55 MB	
vk size	1.3 kB		1.3 kB	
prover $P^\star$				
	TIME			
	1 thread	4 threads	1 thread	4 threads
per step	24.2 s	8.3 s	26.2 s	9.0 s
	SPACE			
memory	800 MB	1,063 MB	993 MB	1,268 MB
proof	374 B		374 B	
verifier $V^\star$				
	TIME			
$\lvert\mathcal{P}\rvert = 10$	23.2 ms		23.9 ms	
$\lvert\mathcal{P}\rvert = 10^2$	23.7 ms		24.5 ms	
$\lvert\mathcal{P}\rvert = 10^3$	29.8 ms		30.8 ms	
$\lvert\mathcal{P}\rvert = 10^4$	90.9 ms		94.2 ms	
in general	$\approx (23.08 + 0.00676\lvert\mathcal{P}\rvert)$ ms		$\approx (23.78 + 0.00702\lvert\mathcal{P}\rvert)$ ms	

Figure 7: Performance of our scalable zk-SNARK on 16-bit and 32-bit vnTinyRAM. (The reported times are the average of 20 experiments, with standard deviation less than 1%.)

		key generator		key sizes		prover		verifier
		TIME	SPACE	$\lvert$pk$\rvert$	$\lvert$vk$\rvert$	TIME	SPACE	TIME
16-bit vnTinyRAM	[BCTV14]	$0.08 \cdot T$ s	$1.8 \cdot T$ MB	$0.3 \cdot T$ MB	40.4 kB	$0.04 \cdot T$ s	$1.9 \cdot T$ MB	24.2 ms
$(w,k)=(16,16)$	**this work**	33.8 s	861 MB	43 MB	1.3 kB	$24.2 \cdot T$ s	800 MB	90.9 ms
32-bit vnTinyRAM	[BCTV14]	$0.13 \cdot T$ s	$3.1 \cdot T$ MB	$0.4 \cdot T$ MB	80.3 kB	$0.05 \cdot T$ s	$3.1 \cdot T$ MB	41.0 ms
$(w,k)=(32,16)$	**this work**	42.0 s	1,068 MB	55 MB	1.3 kB	$26.2 \cdot T$ s	993 MB	94.2 ms

Figure 8: Comparison between [BCTV14]'s preprocessing zk-SNARK and our scalable zk-SNARK.

VIPS. Finally, being scalable, our zk-SNARK implementation is the first to achieve a well-defined clock rate of *verified instructions per second* (VIPS). For vnTinyRAM, we obtain the following VIPS values:

	16-bit vnTinyRAM $(w, k) = (16, 16)$	32-bit vnTinyRAM $(w, k) = (32, 16)$
1 thread	VIPS $= \frac{1}{24.2}$Hz	VIPS $= \frac{1}{26.2}$Hz
4 threads	VIPS $= \frac{1}{8.3}$Hz	VIPS $= \frac{1}{9.0}$Hz

While perhaps too slow for most applications, our prototype empirically demonstrates the feasibility of the bootstrapping approach as a way to achieve scalability of zk-SNARKs and, more generally, to achieve the rich functionality of proof-carrying data.

8 Open problems

Higher clock rate. There are ample opportunities for improving the clock rate of "verified instructions per second". Besides potential improvements in the cryptographic protocol and elliptic curves, there is also an engineering challenge. In particular, the algorithms are highly amenable to parallelism and hardware support. Since each step of proof generation in our zk-SNARK is a *constant-size* operation, it could even be carefully optimized and wholly implemented in a fixed-sized, general-purpose "proving processor" hardware.

Other PCD-friendly cycles. The PCD-friendly 2-cycle proposed in this paper facilitates a great improvement in the efficiency of recursively composing pairing-based zk-SNARKs. Do there exist any other PCD-friendly 2-cycles, not based on MNT curves? Or cycles of length greater than 2? Are these easier to find, achieve smaller bit size and higher 2-adicity, or admit faster nondeterministic pairing verification? Investigating these questions may lead to further efficiency improvements to recursive proof composition. Another consideration is that, with MNT-based PCD-friendly cycles, increasing the security level is costly, since one of the curves has low embedding degree ($k = 4$, for which 128-bit security requires $q_4 \geq 2^{750}$ [FST10]).

Alternative zk-SNARKs constructions. What are the advantages or disadvantages of pairing-based zk-SNARKs in which the pairing is not instantiated via a pairing-friendly elliptic curve, but instead via lattice techniques [GGH13]? Moreover, are there preprocessing zk-SNARKs that are not based on pairings? (E.g., can they be based on groups without bilinear maps?)

Acknowledgments

We thank Andrew V. Sutherland for generous help in running the CM method on elliptic curves with large discriminants. We thank Damien Stehlé and Daniele Micciancio for discussions about the security of subset-sum functions. We thank Koray Karabina for answering questions about algorithms in [KT08].

This work was supported by: the Broadcom Foundation and Tel Aviv University Authentication Initiative; the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370; the Check Point Institute for Information Security; the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258; the Israeli Centers of Research Excellence I-CORE program (center 4/11); the Israeli Ministry of Science and Technology; the Leona M. & Harry B. Helmsley Charitable Trust; the Simons Foundation, with a Simons Award for Graduate Students in Theoretical Computer Science; and the Skolkovo Foundation with agreement dated 10/26/2011.

A Computation models

We introduce notions and notations for two computation models used in this paper: *arithmetic circuits* (see Appendix A.1) and *random-access machines* (see Appendix A.2).

A.1 Arithmetic circuits

We work with circuits that are not boolean but *arithmetic*. Given a field $\mathbb{F}$, an **$\mathbb{F}$-arithmetic circuit** takes inputs that are elements in $\mathbb{F}$, and its gates output elements in $\mathbb{F}$. We naturally associate a circuit with the function it computes. The circuits we consider only have *bilinear gates*,[23] and a circuit's *size* is defined as the number of gates. To model nondeterminism we consider circuits with an *input* $x \in \mathbb{F}^n$ and an auxiliary input $a \in \mathbb{F}^h$, called a *witness*. Arithmetic circuit satisfiability is analogous to the boolean case, as follows.

Definition A.1. *Let $n, h, l \in \mathbb{N}$ respectively denote the input, witness, and output size. The* **circuit satisfaction problem** *of an $\mathbb{F}$-arithmetic circuit $C\colon \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ (with bilinear gates) is defined by the relation $\mathcal{R}_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, a) = 0^l\}$; its language is $\mathcal{L}_C = \{x \in \mathbb{F}^n : \exists\, a \in \mathbb{F}^h,\ C(x, a) = 0^l\}$.*

At times, we also write $C(x, a) = 0$ to mean $C(x, a) = 0^l$ for an unspecified l. All the arithmetic circuits we consider are over fields $\mathbb{F}_p$ with p prime.

A.2 Random-access machines

There are many possible definitions of *random-access machines* [CR72, AV77]. Here we formulate a concrete, yet relatively flexible, definition that suffices for the purposes of this paper. Informally, a machine is specified by a configuration for random-access memory (number of addresses, and number of bits stored at each address) and a CPU. At each step, the CPU gets the current state and the next instruction from memory; executes the instruction; communicates with memory (by storing or loading data); and then outputs the next state and the address for the next instruction. (Thus, random-access memory contains both program and data.)

More precisely, a (non-deterministic) **random-access machine** with verification over a finite field $\mathbb{F}$ is a tuple $\mathbf{M} = (A, W, N, \mathrm{CPU}_{\mathsf{exe}}, \mathrm{CPU}_{\mathsf{ver}})$ where:

- $A, W \in \mathbb{N}$ specify that (random-access) memory $\mathcal{M}$ contains A addresses each storing W bits (i.e., that memory is a function $\mathcal{M}\colon [A] \to \{0,1\}^W$).
- $N \in \mathbb{N}$ specifies the length, in bits, of a CPU state.
- $\mathrm{CPU}_{\mathsf{exe}}$ is a (stateful) function for *executing* the CPU (see below).
- $\mathrm{CPU}_{\mathsf{ver}}$ is an $\mathbb{F}$-arithmetic circuit for *verifying* the CPU's execution (see below).

The machine $\mathbf{M}$ takes as input a program $\mathcal{P}$ and an auxiliary input $\mathcal{G}$, and computes on them. More precisely:

- A **program** for $\mathbf{M}$ is a function $\mathcal{P}\colon [A] \to \{0,1\}^W$ that specifies the initial memory contents. The program $\mathcal{P}$ is typically represented in sparse form, by listing the (few) addresses and values for non-zero memory entries, which may store any code and data to the machine.
- An **auxiliary input** for $\mathbf{M}$ is a sequence $\mathcal{G} = (g_0, g_1, g_2, \dots)$. Each g_i consists of W bits and is accessed at the i-th computation step. The auxiliary input is treated as a *nondeterministic guess*.

Then, the **computation** of $\mathbf{M}$ on program $\mathcal{P}$ and auxiliary input $\mathcal{G}$, denoted $\mathbf{M}(\mathcal{P}; \mathcal{G})$, proceeds as follows. Initialize the CPU state and instruction address to zero: $s_{\mathsf{cpu},0} := 0^N$, $a_{\mathsf{pc},0} := 0$. Next, for $i = 0, 1, 2 \dots$:

1. $\mathrm{CPU}_{\mathsf{exe}}$ is given the current CPU state ($s_{\mathsf{cpu},i} \in \{0,1\}^N$), address of the instruction to be executed ($a_{\mathsf{pc},i} \in [A]$), instruction to be executed ($v_{\mathsf{pc},i} := \mathcal{M}_i(a_{\mathsf{pc},i}) \in \{0,1\}^W$), and guess ($g_i \in \{0,1\}^W$).
2. $\mathrm{CPU}_{\mathsf{exe}}$ outputs an address ($a_{\mathsf{mem},i} \in [A]$), a value ($v_{\mathsf{st},i} \in \{0,1\}^W$), and a store flag ($f_{\mathsf{st},i} \in \{0,1\}$).

[23] A gate with inputs $x_1, \dots, x_m \in \mathbb{F}$ is *bilinear* if the output is $\langle \alpha, (1, x_1, \dots, x_m)\rangle \cdot \langle \beta, (1, x_1, \dots, x_m)\rangle$ for some $\alpha, \beta \in \mathbb{F}^{m+1}$. In particular, these include addition, multiplication, and constant gates.

3. $\mathrm{CPU}_{\mathsf{exe}}$ is given the value at the address ($v_{\mathsf{ld},i} := \mathcal{M}_i(a_{\mathsf{mem},i}) \in \{0,1\}^W$).
   Set $\mathcal{M}_{i+1}$ equal to $\mathcal{M}_i$; if a store was requested (i.e., $f_{\mathsf{st},i} = 1$), do it (i.e., $\mathcal{M}_{i+1}(a_{\mathsf{mem},i}) := v_{\mathsf{st},i}$).

4. $\mathrm{CPU}_{\mathsf{exe}}$ outputs a new CPU state ($s_{\mathsf{cpu},i+1} \in \{0,1\}^N$), an address for the next instruction ($a_{\mathsf{pc},i+1} \in [A]$), and a flag denoting whether the machine has accepted ($f_{\mathsf{acc},i+1} \in \{0,1\}$). $\mathrm{CPU}_{\mathsf{exe}}$'s state is reset.

Thus $\mathrm{CPU}_{\mathsf{exe}}$ can be thought of as $\mathbf{M}$'s "processor": step after step, $\mathrm{CPU}_{\mathsf{exe}}$ takes the previous state and instruction (and its address), executes the instruction, communicates with random-access memory, and produces the next state and instruction address. In contrast, $\mathrm{CPU}_{\mathsf{ver}}$ is a predicate that verifies the correct input/output relationship of $\mathrm{CPU}_{\mathsf{exe}}$. In other words $\mathrm{CPU}_{\mathsf{exe}}$ satisfies the following property:

> Fix $s_{\mathsf{cpu}}, s'_{\mathsf{cpu}} \in \{0,1\}^N$, $a_{\mathsf{pc}}, a_{\mathsf{mem}}, a'_{\mathsf{pc}} \in [A]$, $v_{\mathsf{pc}}, v_{\mathsf{st}}, v_{\mathsf{ld}}, g \in \{0,1\}^W$, $f_{\mathsf{st}}, f'_{\mathsf{acc}} \in \{0,1\}$, and let x_{ver} be the concatenation of all these. There is a witness a_{ver} such that $\mathrm{CPU}_{\mathsf{ver}}(x_{\mathsf{ver}}, a_{\mathsf{ver}}) = 0$ iff $(a_{\mathsf{mem}}, v_{\mathsf{st}}, f_{\mathsf{st}}) \leftarrow \mathrm{CPU}_{\mathsf{exe}}(s_{\mathsf{cpu}}, a_{\mathsf{pc}}, v_{\mathsf{pc}}, g)$ and, afterwards, $(s'_{\mathsf{cpu}}, a'_{\mathsf{pc}}, f'_{\mathsf{acc}}) \leftarrow \mathrm{CPU}_{\mathsf{exe}}(v_{\mathsf{ld}})$. Moreover, a_{ver} can be efficiently computed from x_{ver}.

Of course, $\mathrm{CPU}_{\mathsf{ver}}$ may simply internally execute $\mathrm{CPU}_{\mathsf{exe}}$ to perform its verification; but, having access to additional advice a_{ver}, $\mathrm{CPU}_{\mathsf{ver}}$ may instead perform "smarter", and more efficient, checks.

We are not concerned about how the function $\mathrm{CPU}_{\mathsf{exe}}$ is specified (e.g., it can be a computed program), but $\mathrm{CPU}_{\mathsf{ver}}$ must be specified as an $\mathbb{F}$-arithmetic circuit (for an appropriate $\mathbb{F}$ that we will discuss).

The language of accepting computations. We define the language of accepting computations on $\mathbf{M}$. A program $\mathcal{P}$ is treated as "given", while the auxiliary input $\mathcal{G}$ is treated as a nondeterministic advice.

Definition A.2. *For a random-access machine $\mathbf{M}$, the language $\mathcal{L}_{\mathbf{M}}$ consists of pairs $(\mathcal{P}, T)$ such that:*
- *$\mathcal{P}$ is a program for $\mathbf{M}$;*
- *T is a time bound;*
- *there exists an auxiliary input $\mathcal{G}$ such that $\mathbf{M}(\mathcal{P}; \mathcal{G})$ accepts in at most T steps.*

We denote by $\mathcal{R}_{\mathbf{M}}$ the relation corresponding to $\mathcal{L}_{\mathbf{M}}$.

In this paper we obtain an implementation of scalable zk-SNARKs for proving/verifying membership in the above language (see Appendix E for a definition). We evaluate our system for a specific choice of machine: vnTinyRAM, a simple RISC von Neumann architecture introduced by [BCTV14] (see below). Of course, other choices of random-access machines are possible, and our implementation supports them.

A.3 The architecture vnTinyRAM

We evaluate our scalable zk-SNARK on an architecture that previously appeared in (preprocessing) zk-SNARK implementations: vnTinyRAM [BCTV14]. (See Section 7.) We explain how to set "$\mathbf{M} =$ vnTinyRAM", i.e., how to specify the architecture vnTinyRAM via the formalism introduced above (and used by our prototype).

Given w, k, we want to construct a tuple $\mathbf{M} = (A, W, N, \mathrm{CPU}_{\mathsf{exe}}, \mathrm{CPU}_{\mathsf{ver}})$ that implements w-bit vnTinyRAM with k registers. First we need to specify the parameters $A, W \in \mathbb{N}$ for random access memory. vnTinyRAM accesses memory, consisting of 2^w bytes, either as bytes or as words; moreover, vnTinyRAM instructions (which are stored in memory) take two words to encode in memory. Thus, we set A, W so that memory consists of $A := \frac{8 \cdot 2^w}{2w}$ addresses, each storing $W := 2w$ bits. Next, we set the CPU state length to $N := (1+k)w + 1$ because, in vnTinyRAM, a CPU state consists of the *program counter* (w bits), k general-purpose *registers* (each of w bits), and a *(condition) flag* (1 bit). Finally, $\mathrm{CPU}_{\mathsf{exe}}$ can be chosen to be any program implementation of vnTinyRAM's CPU, while $\mathrm{CPU}_{\mathsf{ver}}$ can be chosen to be any $\mathbb{F}$-arithmetic circuit for verifying the input-output relationship of $\mathrm{CPU}_{\mathsf{exe}}$. In our implementation, $\mathbb{F}$ is a prime field of 298 bits (since $\mathbb{F} = \mathbb{F}_{r_4}$), and we get the following sizes for the two settings we consider:
- for $(w, k) = (16, 16)$, $|\mathrm{CPU}_{\mathsf{ver}}| = 766$; and
- for $(w, k) = (32, 16)$, $|\mathrm{CPU}_{\mathsf{ver}}| = 1108$.

B Pairings and elliptic curves

The cryptographic primitives we study are based on *pairings*, which we briefly recall in Appendix B.1. Pairings can, in turn, be based on *pairing-friendly elliptic curves*; in Appendix B.2 we review basic notions about these.

B.1 Pairings

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic groups of a prime order r. We denote elements of $\mathbb{G}_1, \mathbb{G}_2$ via calligraphic letters such as $\mathcal{P}, \mathcal{Q}$. We write $\mathbb{G}_1$ and $\mathbb{G}_2$ in additive notation. Let $\mathcal{P}_1$ be a generator of $\mathbb{G}_1$, i.e., $\mathbb{G}_1 = \{\alpha\mathcal{P}_1\}_{\alpha\in\mathbb{F}_r}$; let $\mathcal{P}_2$ be a generator for $\mathbb{G}_2$. (We also view α as an integer, so that $\alpha\mathcal{P}_1$ is well-defined.)

A **pairing** is an efficient map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_T$ is also a cyclic group of order r (which we write in multiplicative notation), satisfying the following properties:

- BILINEARITY. For every nonzero elements $\alpha, \beta \in \mathbb{F}_r$, it holds that $e(\alpha\mathcal{P}_1, \beta\mathcal{P}_2) = e(\mathcal{P}_1, \mathcal{P}_2)^{\alpha\beta}$.
- NON-DEGENERACY. $e(\mathcal{P}_1, \mathcal{P}_2)$ is not the identity in $\mathbb{G}_T$.

When describing cryptographic primitives at high level, the choice of instantiation of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e$ often does not matter. In this paper, however, we discuss implementation details, and such choices matter a great deal. Typically, pairings are based on (pairing-friendly) elliptic curves, discussed next.

B.2 Elliptic curves

We assume familiarity with elliptic curves; here, we only recall the basic definitions in order to fix notation. See, e.g., [Was08, Sil09, FST10, CFAD+12] for more details.

Definition and curve groups. Given a field K, an *elliptic curve* E defined over K, denoted E/K, is a smooth projective curve of genus 1 (defined over K) with a distinguished K-rational point. We denote by $E(K)$ the group of K-rational points on E; when finite, we denote the cardinality of this group by $\#E(K)$. For any $r \in \mathbb{N}$, $E[r]$ denotes the group of r-torsion points in $E(\overline{K})$, and $E(K)[r]$ the group of r-torsion points in $E(K)$. In this paper, we only consider elliptic curves where K is a finite field $\mathbb{F}_q$; so the definitions below are specific to this case.

Trace and CM discrminant. The *trace* of $E/\mathbb{F}_q$ is $t := q + 1 - \#E(\mathbb{F}_q)$. The *Hasse bound* states that $|t| \leq 2\sqrt{q}$. If $\gcd(q, t) = 1$, then $E/\mathbb{F}_q$ is *ordinary*; otherwise, it is *supersingular*. If $E/\mathbb{F}_q$ is ordinary, the *CM discriminant* of E is the square-free part D of the integer $4q - t^2$, non-negative by the Hasse bound.[24]

ECDLP. The *elliptic-curve discrete logarithm problem* (ECDLP) is the following: given $E/\mathbb{F}_q$, $\mathcal{P} \in E(\mathbb{F}_q)$, and $\mathcal{Q} \in \langle\mathcal{P}\rangle$, find $a \in \mathbb{N}$ such that $\mathcal{Q} = a\mathcal{P}$. There are several known methods to solve, with different time and space complexities, the ECDLP. For instance: the Pohlig–Hellman algorithm [PH78] (which reduces the problem to subgroups of prime order); Shanks' [Sha71] baby-step-giant-step method; Pollard's methods (the rho method [Pol78] and the kangaroo method [Pol00], and their parallel variants by van Oorschot and Wiener [vOW99]); the Menezes–Okamoto–Vanstone (MOV) attack using the Weil pairing [MOV91]; the Frey–Rück attack using the Tate pairing [FR94]; and the SSSA attack for curves of trace $t = 1$ [Sem98, Sma99, SA98].

Cryptographic uses require the ECDLP to be hard. For points $\mathcal{P}$ of large prime order r, this is widely believed to be the case. Thus, one only considers curves E with trace $t \neq 1$ and having cyclic subgroups of $E(\mathbb{F}_q)$ of large prime order r. So $\#E(\mathbb{F}_q)$ is either a prime r, or hr for a small *cofactor* h.

Pairings. For cryptographic uses that require efficient computation of pairings (such as the uses considered in this paper), suitable elliptic curves need to satisfy additional requirements, as we now recall.

[24] Alternatively, some authors define the discriminant to be $-D$, or the discriminant of the imaginary quadratic field $\mathbb{Q}(\sqrt{-D})$.

For any $r \in \mathbb{N}$ with $\gcd(q, r) = 1$, the *embedding degree* k of $E/\mathbb{F}_q$ (with respect to r) is the smallest integer such that r divides $q^k - 1$; for such r, a bilinear map $e_r \colon E[r] \times E[r] \to \mu_r$ can be defined, where $\mu_r \subset \mathbb{F}^*_{q^k}$ is the subgroup of r-th roots of unity in $\overline{\mathbb{F}_q}$. The map e_r is known as the *Weil pairing*.

The Weil pairing is not the only bilinear map that can be defined. Depending on properties of the curve E other, sometimes more efficient, pairings can be defined, e.g., the Tate pairing [FR94, FMR06], the Eta pairing [BGOhM07], and the Ate pairing [HSV06]. In each of these cases, the pairing computation requires arithmetic in $\mathbb{F}_{q^k}$, so that k cannot be too large. On the other hand, the ECDLP can be translated (via the pairing itself [MOV91, FR94]) to the discrete logarithm problem over $\mathbb{F}^*_{q^k}$, which is susceptible to subexponential-time attacks via index calculus [Odl85], so that k has to be large enough to achieve the desired level of hardness for the DLP in $\mathbb{F}^*_{q^k}$.

In light of the above considerations, an (ordinary) elliptic curve $E/\mathbb{F}_q$ is said to be *pairing friendly* if (i) $E(\mathbb{F}_q)$ contains a subgroup of large prime order r, and (ii) E has embedding degree k (with respect to r) that is not too large (i.e., computations in the field $\mathbb{F}_{q^k}$ are feasible) and not too small (i.e., the DLP in $\mathbb{F}^*_{q^k}$ is hard enough). The ideal case is when E has prime order r, and the embedding degree k is such that the ECDLP in $E(\mathbb{F}_q)$ and the DLP in $\mathbb{F}^*_{q^k}$ have approximately the same hardness, i.e., are *balanced*.

Instantiations of pairings. A pairing is specified by a prime $r \in \mathbb{N}$, three cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order r, and an efficient bilinear map $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. (See Appendix B.1.) Suppose one uses a curve $E/\mathbb{F}_q$ with embedding degree k to instantiate the pairing. Then $\mathbb{G}_T$ is set to $\mu_r \subset \mathbb{F}^*_{q^k}$. The instantiation of $\mathbb{G}_1$ and $\mathbb{G}_2$ depends on the choice of e; typically, $\mathbb{G}_1$ is instantiated as an order-r subgroup of $E(\mathbb{F}_q)$, while, for efficiency reasons [BKLS02, BLS04], $\mathbb{G}_2$ as an order-r subgroup of $E'(\mathbb{F}_{k/d})$ where E' is a *d-th twist* of E.

C Preprocessing zk-SNARKs for arithmetic circuit satisfiability

At high-level, a *preprocessing* zk-SNARK for arithmetic-circuit satisfiability is a cryptographic primitive that provides short and easy-to-verify non-interactive zero-knowledge proofs of knowledge for the satisfiability of arithmetic circuits. A public proving key is used to generate proofs, and a public verification key is used to verify them; the two keys are jointly generated once, and can then be used any number of times. The adjective "preprocessing" denotes the fact that the key pair *depends* on the arithmetic circuit C whose satisfiability is being proved/verified; in particular, the time to generate a key pair for C is at least linear in the size of C. Below, we informally define this primitive; we refer the reader to, e.g., [BCIOP13] for a formal definition.

Given a field $\mathbb{F}$, a **preprocessing zk-SNARK** for $\mathbb{F}$-arithmetic circuit satisfiability (see Appendix A.1) is a triple of polynomial-time algorithms (G, P, V), with V deterministic,[25] working as follows.

- $G(1^\lambda, C) \to (\mathsf{pk}, \mathsf{vk})$. On input a security parameter λ (presented in unary) and an $\mathbb{F}$-arithmetic circuit C, the *key generator* G probabilistically samples a proving key pk and a verification key vk. We assume, without loss of generality, that pk contains (a description of) the circuit C.

The keys pk and vk are published as public parameters and can be used, any number of times, to prove/verify membership in the language $\mathcal{L}_C$, as follows.

- $P(\mathsf{pk}, x, a) \to \pi$. On input a proving key pk and any $(x, a) \in \mathcal{R}_C$, the *prover* P outputs a non-interactive proof π for the statement "$x \in \mathcal{L}_C$".
- $V(\mathsf{vk}, x, \pi) \to b$. On input a verification key vk, an input x, and a proof π, the *verifier* V outputs $b = 1$ if he is convinced by π that $x \in \mathcal{L}_C$.

The triple (G, P, V) satisfies the following properties.

Completeness. The honest prover can convince the verifier for any instance in the language. Namely, for every security parameter λ, $\mathbb{F}$-arithmetic circuit C, and instance $x \in \mathcal{L}_C$ with a witness a,

$$\Pr\left[V(\mathsf{vk}, x, \pi) = 1 \;\middle|\; \begin{array}{r}(\mathsf{pk}, \mathsf{vk}) \leftarrow G(1^\lambda, C) \\ \pi \leftarrow P(\mathsf{pk}, x, a)\end{array}\right] = 1 \ .$$

Succinctness. For every security parameter λ, $\mathbb{F}$-arithmetic circuit C, and $(\mathsf{pk}, \mathsf{vk}) \in G(1^\lambda, C)$,
- an honestly-generated proof π has $O_\lambda(1)$ bits;
- $V(\mathsf{vk}, x, \pi)$ runs in time $O_\lambda(|x|)$.

Above, O_λ hides a (fixed) polynomial factor in λ.

Proof of knowledge (and soundness). If the verifier accepts a proof for an instance, the prover "knows" a witness for that instance. (Thus, soundness holds.) Namely, for every constant $c > 0$ and every polynomial-size adversary A there is a polynomial-size witness extractor E such that, for every large-enough security parameter λ, for every $\mathbb{F}$-arithmetic circuit C of size λ^c,

$$\Pr\left[\begin{array}{c}V(\mathsf{vk}, x, \pi) = 1 \\ (x, a) \notin \mathcal{R}_C\end{array}\;\middle|\; \begin{array}{r}(\mathsf{pk}, \mathsf{vk}) \leftarrow G(1^\lambda, C) \\ (x, \pi) \leftarrow A(\mathsf{pk}, \mathsf{vk}) \\ a \leftarrow E(\mathsf{pk}, \mathsf{vk})\end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

Statistical zero knowledge. An honestly-generated proof is statistical zero knowledge. Namely, there is a polynomial-time stateful simulator S such that, for all stateful distinguishers D, the following two probabilities are negligibly-close:

[25] In this paper we use the technique of recursive proof composition, which relies on V being deterministic (i.e., no coin flips are needed during proof verification). All known zk-SNARK constructions satisfy this property, so this is effectively not a restriction.

$$\Pr\left[\begin{array}{c|r} (x,a)\in\mathcal{R}_C & C\leftarrow D(1^\lambda) \\ \mathcal{D}(\pi)=1 & (\mathsf{pk},\mathsf{vk})\leftarrow G(1^\lambda,C) \\ & (x,a)\leftarrow D(\mathsf{pk},\mathsf{vk}) \\ & \pi\leftarrow P(\mathsf{pk},x,a) \end{array}\right] \quad \text{and} \quad \Pr\left[\begin{array}{c|r} (x,a)\in\mathcal{R}_C & C\leftarrow D(1^\lambda) \\ D(\pi)=1 & (\mathsf{pk},\mathsf{vk})\leftarrow S(1^\lambda,C) \\ & (x,a)\leftarrow D(\mathsf{pk},\mathsf{vk}) \\ & \pi\leftarrow S(x) \end{array}\right].$$

Remark C.1. All known preprocessing zk-SNARK constructions can in fact be made *perfect* zero knowledge, at the only expense of a negligible probability of error in completeness.

C.1 Known constructions and security

There are many preprocessing zk-SNARK constructions in the literature [Gro10, Lip12, BCIOP13, GGPR13, PGHR13, BCGTV13a, Lip13, BCTV14]. The most efficient ones are based on *quadratic arithmetic programs* (QAPs) [GGPR13, BCIOP13, PGHR13, BCGTV13a, BCTV14]; such constructions provide a linear-time G, quasilinear-time P, and linear-time V.

Three of the above works [PGHR13, BCGTV13a, BCTV14] also investigate and provide implementations of preprocessing zk-SNARKs. As we discuss in Section 3.3, in this work we follow the implementation of [BCTV14], which, at the time of writing, is the fastest one.

Security of zk-SNARKs is based on knowledge-of-exponent assumptions and variants of Diffie–Hellman assumptions in bilinear groups [Gro10, BB04, Gen04]. Knowledge-of-exponent assumptions are fairly strong, but there is evidence that such assumptions may be inherent for constructing zk-SNARKs [GW11, BCCT12].

Remark C.2 (auxiliary input)**.** More generally, the security of zk-SNARKs relies on the *extractability* of certain functions. Extractability is a delicate property that, depending on how it is stated, yields conditions of different relative strength. One aspect that affects this is the choice of *auxiliary input* (a discussion of which was omitted in the informal definition above). For instance, if the adversary is allowed *any* auxiliary input, extraction may be difficult because the auxiliary input may encode an obfuscated strategy [BCCT12]; such intuition can in fact be formalized to yield limitations to extractability [BCPR13]. On the other end of the spectrum, certain notions of extractability can be achieved [BCP13].

The focus of this paper is practical aspects of zk-SNARKs so our perspective on extractability here is that, similarly to the Fiat–Shamir paradigm [FS87], knowledge-of-exponent assumptions, despite not being fully understood, provide solid heuristics in practice since no effective attacks against them are known.

C.2 Instantiations via elliptic curves

Known preprocessing zk-SNARK constructions are based on *pairings* (see Appendix B.1), which can in turn be based on *pairing-friendly elliptic curves* (see Appendix B.2). We recall two facts, used in this paper.

Field for the circuit language. Let E be an elliptic curve that is defined over a finite field $\mathbb{F}_q$, has a group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points with a prime order r (or order divisible by a large prime r), and has embedding degree k with respect to r. Suppose that a preprocessing zk-SNARK (G,P,V) is instantiated with E. Then,

> (G,P,V) works for $\mathbb{F}_r$-arithmetic circuit satisfiability,
> but all of V's arithmetic computations are over $\mathbb{F}_q$ (or extensions of $\mathbb{F}_q$ up to degree k).[26]

This fact motivates most of the discussions in Section 3.1.

2-adicity of a curve. Prior work identified the *2-adicity* of a curve as an important ingredient for efficient implementations of the generator and, especially, the prover [BCGTV13a, BCTV14].

[26] Intuitively, this is because: (a) the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ for a pairing $e\colon \mathbb{G}_1\times\mathbb{G}_2\to\mathbb{G}_T$ on E have prime order r, so that discrete logarithms for elements in $\mathbb{G}_1$ and $\mathbb{G}_2$, which encode information about the arithmetic circuit, "live" in $\mathbb{F}_r$; while (b) $\mathbb{G}_1$-arithmetic, $\mathbb{G}_2$-arithmetic, and pairing computations are conducted over $\mathbb{F}_q$ (or extensions of $\mathbb{F}_q$ up to degree k).

An elliptic curve $E/\mathbb{F}_q$ has *2-adicity* 2^ℓ if the large prime r dividing $\#E(\mathbb{F}_q)$ is such that 2^ℓ divides $r-1$. This property ensures that the multiplicative group of $\mathbb{F}_r$ contains a 2^ℓ-th root of unity, which significantly improves the efficiency of interpolation and evaluation of functions defined over certain domains in $\mathbb{F}_r$.

When instantiating a preprocessing zk-SNARK (G, P, V) with E, the zk-SNARK works for $\mathbb{F}_r$-arithmetic circuit satisfiability, and both G and P need to solve interpolation/evaluation problems over domains of size $|C|$, where C is the $\mathbb{F}_r$-arithmetic circuit given as input to G. Thus, efficiency can be improved if E is sufficiently 2-adic. Concretely, to fully take advantage of the efficiency benefits of 2-adicity, one requires that $2^\ell \geq |C|$, i.e., $\nu_2(r-1) \geq \lceil \log |C| \rceil$ where $\nu_2(\cdot)$ denotes the 2-adic order function.

This fact motivates much of the extensive search for suitable curve parameters, described in Section 3.2.

Remark C.3 (lack of 2-adicity)**.** One can consider other/weaker requirements (e.g., $\nu_3(r-1) \geq \lceil \log_3 |C| \rceil$, or $r-1$ is divisible by a smooth number $M \geq |C|$) which would still somewhat simplify interpolation/evaluation problems over $|C|$-size domains in $\mathbb{F}_r$. The above requirement that $\nu_2(r-1) \geq \lceil \log |C| \rceil$ is, in a sense, the "ideal" one. Moreover, even if E does not satisfy these other/weaker requirements, it is still possible to instantiate the zk-SNARK, but at a higher computational cost (both asymptotically and in practice), due to the necessary use of "heavier" techniques applying to "generic" fields [PGHR13].

C.3 The zk-SNARK verifier protocol

The (pairing-based) preprocessing zk-SNARKs that we use follow those of [BCTV14] (see Section 3.3); in turn, these improve upon and implement those of [PGHR13]. In this paper, we construct arithmetic circuits for verifying the evaluation of the zk-SNARK verifier V: a circuit $C_{V,4}$ for an instantiation based on the curve E_4, and a circuit $C_{V,6}$ for one based on the curve E_6 (see Section 5.1). For completeness, in Figure 9 we summarize V's abstract protocol.

We see that V's protocol consists of two main parts: (a) use the verification key vk and input $\vec{x} \in \mathbb{F}_r^n$ to compute $\mathsf{vk}_{\vec{x}}$ (see Step 1); and (b) use the verification key vk, value $\mathsf{vk}_{\vec{x}}$, and proof π, to compute 12 pairings and perform the required checks (see Step 2, Step 3, Step 4). Thus, the first part requires $O(n)$ scalar multiplications in $\mathbb{G}_1$, while the second part requires $O(1)$ pairing evaluations.

For additional details regarding V (and, more generally, the preprocessing zk-SNARK construction), we refer the reader to [BCTV14, PGHR13]. Indeed, our focus in this work is not *why* V executes these checks, but *how* we can efficiently verify its checks via suitable arithmetic circuits.

ALGEBRAIC SETUP. A prime r, two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order r with generators $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and a pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_T$ is also cyclic of order r. (See Appendix B.1 for a pairing's definition.)

zk-SNARK verifier V for inputs of size n

- INPUTS:
  - verification key $\mathsf{vk} = (\mathsf{vk}_\mathsf{A}, \mathsf{vk}_\mathsf{B}, \mathsf{vk}_\mathsf{C}, \mathsf{vk}_\gamma, \mathsf{vk}^1_{\beta\gamma}, \mathsf{vk}^2_{\beta\gamma}, \mathsf{vk}_\mathsf{Z}, \mathsf{vk}_\mathsf{IC})$, where
    - $\mathsf{vk}_\mathsf{B}, \mathsf{vk}^1_{\beta\gamma}$ are in $\mathbb{G}_1$
    - $\mathsf{vk}_\mathsf{A}, \mathsf{vk}_\mathsf{C}, \mathsf{vk}_\gamma, \mathsf{vk}^2_{\beta\gamma}, \mathsf{vk}_\mathsf{Z}$ are in $\mathbb{G}_2$
    - $\mathsf{vk}_\mathsf{IC} = (\mathsf{vk}_{\mathsf{IC},0}, \mathsf{vk}_{\mathsf{IC},1}, \ldots, \mathsf{vk}_{\mathsf{IC},n}) \in \mathbb{G}_1^{1+n}$
  - input $\vec{x} = (x_1, \ldots, x_n)$, where $x_i \in \mathbb{F}_r$
  - proof $\pi = (\pi_\mathsf{A}, \pi'_\mathsf{A}, \pi_\mathsf{B}, \pi'_\mathsf{B}, \pi_\mathsf{C}, \pi'_\mathsf{C}, \pi_\mathsf{K}, \pi_\mathsf{H})$, where $\pi_\mathsf{A}, \pi'_\mathsf{A}, \pi'_\mathsf{B}, \pi_\mathsf{C}, \pi'_\mathsf{C}, \pi_\mathsf{K}, \pi_\mathsf{H} \in \mathbb{G}_1$ and $\pi_\mathsf{B} \in \mathbb{G}_2$
- OUTPUTS: decision bit

1. Compute $\mathsf{vk}_{\vec{x}} := \mathsf{vk}_{\mathsf{IC},0} + \sum_{i=1}^n x_i \mathsf{vk}_{\mathsf{IC},i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments: $e(\pi_\mathsf{A}, \mathsf{vk}_\mathsf{A}) = e(\pi'_\mathsf{A}, \mathcal{P}_2)\,, e(\mathsf{vk}_\mathsf{B}, \pi_\mathsf{B}) = e(\pi'_\mathsf{B}, \mathcal{P}_2)\,, e(\pi_\mathsf{C}, \mathsf{vk}_\mathsf{C}) = e(\pi'_\mathsf{C}, \mathcal{P}_2)$.
3. Check same coefficients were used: $e(\pi_\mathsf{K}, \mathsf{vk}_\gamma) = e(\mathsf{vk}_{\vec{x}} + \pi_\mathsf{A} + \pi_\mathsf{C}, \mathsf{vk}^2_{\beta\gamma}) \cdot e(\mathsf{vk}^1_{\beta\gamma}, \pi_\mathsf{B})$.
4. Check QAP divisibility: $e(\mathsf{vk}_{\vec{x}} + \pi_\mathsf{A}, \pi_\mathsf{B}) = e(\pi_\mathsf{H}, \mathsf{vk}_\mathsf{Z}) \cdot e(\pi_\mathsf{C}, \mathcal{P}_2)$.
5. Accept if and only if all the above checks succeeded.

Figure 9: Summary of the checks performed by the zk-SNARK verifier V.

D Proof-carrying data for arithmetic compliance predicates

We define a *proof-carrying data system* (PCD system), which is a cryptographic primitive that captures the notion of proof-carrying data [CT10, CT12]. More precisely, we define *preprocessing* PCD systems [BCCT13]. The definitions here are somewhat informal; for details, we refer the reader to [BCCT13].

Proof-carrying data at a glance. Fix a predicate Π. Consider a distributed computation where nodes perform computations; each computation takes as input messages and outputs a new output message. The security goal is to ensure that each output message is *compliant* with the predicate Π. Proof-carrying data ensures this goal by attaching short and easy-to-verify proofs of Π-compliance to each message.

Concretely, a *key generator* $\mathbb{G}$ first sets up a proving key and a verification key. Anyone can then use a *prover* $\mathbb{P}$, which is given as input the proving key, prior messages $\vec{z}_{\mathsf{in}}$ with proofs $\vec{\pi}_{\mathsf{in}}$, and an output message z, to generate a proof π attesting that z is Π-compliant. Anyone can use a *verifier* $\mathbb{V}$, which is given as input the verification key, a message z, and a proof, to verify that z is Π-compliant.

Crucially, proof generation and proof verification time are "history independent": the first only depends on the time to execute Π on input a node's messages, while the second only on the message length.

We now spell out more details, by first specifying the notion of distributed computation, and then that of compliance with a predicate Π. Our discussion is specific to predicates specified as $\mathbb{F}$-arithmetic circuits.

Transcripts. Given $n_{\mathsf{msg}}, n_{\mathsf{loc}}, s \in \mathbb{N}$ and field $\mathbb{F}$, an $\mathbb{F}$-arithmetic **transcript** (for message size n_{msg}, local-data size n_{loc}, and arity s) is a triple $\mathsf{T} = (G, \mathsf{loc}, \mathsf{data})$, where $G = (V, E)$ is a directed acyclic graph G, $\mathsf{loc}\colon V \to \mathbb{F}^{n_{\mathsf{loc}}}$ are node labels, and $\mathsf{data}\colon E \to \mathbb{F}^{n_{\mathsf{msg}}}$ are edge labels. The *output of* T, denoted $\mathsf{out}(\mathsf{T})$, equals $\mathsf{data}(\tilde{u}, \tilde{v})$ where $(\tilde{u}, \tilde{v})$ is the lexicographically-first edge with $\tilde{v}$ a sink.

Intuitively, the label $\mathsf{loc}(v)$ of a node v represents the *local data* used by v in his local computation; the edge label $\mathsf{data}(u, v)$ of a directed edged (u, v) represents the *message* sent from node u to node v. Typically, a party at node v uses the local data $\mathsf{loc}(v)$ and "input messages" $\big(\mathsf{data}(u, v)\big)_{u \in \mathsf{parents}(v)}$ to compute an "output message" $\mathsf{data}(v, w)$ for each child $w \in \mathsf{children}(v)$.

Compliance. Given field $\mathbb{F}$ and $n_{\mathsf{msg}}, n_{\mathsf{loc}}, s \in \mathbb{N}$, an **$\mathbb{F}$-arithmetic compliance predicate** Π (for message size n_{msg}, local-data size n_{loc}, and arity s) is an $\mathbb{F}$-arithmetic circuit with domain $\mathbb{F}^{n_{\mathsf{msg}}} \times \mathbb{F}^{n_{\mathsf{loc}}} \times \mathbb{F}^{s \cdot n_{\mathsf{msg}}} \times \mathbb{F}$. The compliance predicate Π specifies whether a given transcript T is compliant or not, as follows. Consider any transcript T with message size n_{msg}, local-data size n_{loc}, and arity s. We say that $\mathsf{T} = (G, \mathsf{loc}, \mathsf{data})$ is Π-*compliant*, denoted $\Pi(\mathsf{T}) = 0$, if, for every $v \in V$ and $w \in \mathsf{children}(v)$, it holds that

$$\Pi\Big(\mathsf{data}(v, w), \mathsf{loc}(v), \big(\mathsf{data}(u, v)\big)_{u \in \mathsf{parents}(v)}, b_{\mathsf{base}}\Big) = 0 \ ,$$

where $b_{\mathsf{base}} \in \{0, 1\}$ is the *base case flag* (i.e., equals 1 if and only if v is a source). Furthermore, we say that a message z is Π-compliant if there is T such that $\Pi(\mathsf{T}) = 0$ and $\mathsf{out}(\mathsf{T}) = z$.

We are now ready to describe the syntax, semantics, and security of a proof-carrying data system.

Given a field $\mathbb{F}$, a **(preprocessing) proof-carrying data system** (PCD system) for $\mathbb{F}$-arithmetic compliance predicates is a triple of polynomial-time algorithms $(\mathbb{G}, \mathbb{P}, \mathbb{V})$ working as follows.

- $\mathbb{G}(1^\lambda, \Pi) \to (\mathsf{pk}, \mathsf{vk})$. On input a security parameter λ (presented in unary) and an $\mathbb{F}$-arithmetic compliance predicate Π, the *key generator* $\mathbb{G}$ probabilistically samples a proving key pk and a verification key vk. We assume, without loss of generality, that pk contains (a description of) the predicate Π.

The keys pk and vk are published as public parameters and can be used, any number of times, to prove/verify Π-compliance of messages.

- $\mathbb{P}(\mathsf{pk}, z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}}) \to \pi$. On input a proving key pk, outgoing message z, local data z_{loc}, and incoming messages $\vec{z}_{\mathsf{in}}$ with proofs $\vec{\pi}_{\mathsf{in}}$, the *prover* $\mathbb{P}$ outputs a proof π for the statement "z is Π-compliant".

- $\mathbb{V}(\mathsf{vk}, z, \pi) \to b$. On input a verification key vk, a message z, and a proof π, the *verifier* $\mathbb{V}$ outputs $b = 1$ if he is convinced by π that z is Π-compliant.

The triple $(\mathbb{G}, \mathbb{P}, \mathbb{V})$ satisfies the following properties.

Completeness. The honest prover can convince the verifier that the output of any compliant transcript is indeed compliant. Namely, for every security parameter λ, $\mathbb{F}$-arithmetic compliance predicate Π, and distributed-computation generator S (see below),

$$\Pr\left[\begin{array}{c}\Pi(\mathsf{T}) = 0 \\ \mathbb{V}\big(\mathsf{vk}, \mathsf{out}(\mathsf{T}), \pi\big) \neq 1\end{array}\middle|\begin{array}{r}(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathbb{G}(1^\lambda, \Pi) \\ (\mathsf{T}, \pi) \leftarrow \mathsf{ProofGen}(S, \mathsf{pk}, \mathbb{P})\end{array}\right] = 0 \ .$$

Above, $\mathsf{ProofGen}$ as an interactive protocol between a *distributed-computation generator* S and the PCD prover $\mathbb{P}$, in which both are given the compliance predicate Π and the proving key pk. Essentially, at every time step, S chooses to do one of the following actions: add a new unlabeled vertex to the computation transcript so far (this corresponds to adding a new computing node to the computation), label an unlabeled vertex (this corresponds to a choice of local data by a computing node), or add a new labeled edge (this corresponds to a new message from one node to another). In case S chooses the third action, the PCD prover $\mathbb{P}$ produces a proof for the Π-compliance of the new message, and adds this new proof as an additional label to the new edge. When S halts, the interactive protocol outputs the distributed computation transcript T, as well as T's output and corresponding proof. Intuitively, the completeness property requires that if T is compliant with Π, then the proof attached to the output (which is the result of dynamically invoking $\mathbb{P}$ for each message in T, as T was being constructed by S) is accepted by the verifier.

Succinctness. For every security parameter λ, $\mathbb{F}$-arithmetic predicate Π, and $(\mathsf{pk}, \mathsf{vk}) \in \mathbb{G}(1^\lambda, \Pi)$,
- an honestly-generated proof π has $O_\lambda(1)$ bits;
- $\mathbb{V}(\mathsf{vk}, z, \pi)$ runs in time $O_\lambda(|z|)$.

Above, O_λ hides a (fixed) polynomial factor in λ.

Proof of knowledge (and soundness). If the verifier accepts a proof for a message, the prover "knows" a compliant transcript T with output z. (Thus, soundness holds.) Namely, for every constant $c > 0$ and every polynomial-size adversary A there is a polynomial-size witness extractor E such that, for every large-enough security parameter λ, for every $\mathbb{F}$-arithmetic compliance predicate Π of size λ^c,

$$\Pr\left[\begin{array}{c}\mathbb{V}(\mathsf{vk}, z, \pi) = 1 \\ \Big(\mathsf{out}(\mathsf{T}) \neq z \ \vee \ \Pi(\mathsf{T}) \neq 0\Big)\end{array}\middle|\begin{array}{r}(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathbb{G}(1^\lambda, \Pi) \\ (z, \pi) \leftarrow A(\mathsf{pk}, \mathsf{vk}) \\ \mathsf{T} \leftarrow E(\mathsf{pk}, \mathsf{vk})\end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

Statistical zero knowledge. An honestly-generated proof is statistical zero knowledge.[27] Namely, there is a polynomial-time stateful simulator S such that, for all stateful distinguishers D, the following two probabilities are negligibly-close:

$$\Pr\left[\Phi = 1 \middle| \begin{array}{r}\Pi \leftarrow D(1^\lambda) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathbb{G}(1^\lambda, \Pi) \\ (z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}}) \leftarrow D(\mathsf{pk}, \mathsf{vk}) \\ \pi \leftarrow P(\mathsf{pk}, z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})\end{array}\right] \quad \text{and} \quad \Pr\left[\Phi = 1 \middle| \begin{array}{r}\Pi \leftarrow D(1^\lambda) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow S(1^\lambda, \Pi) \\ (z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}}) \leftarrow D(\mathsf{pk}, \mathsf{vk}) \\ \pi \leftarrow S(z)\end{array}\right] ,$$

where, above, $\Phi = 1$ if and only if: (i) if $\vec{\pi}_{\mathsf{in}} = \bot$, then $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 1) = 0$; (ii) if $\vec{\pi}_{\mathsf{in}} \neq \bot$, then $\Pi(z, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, 0) = 0$ and, for each corresponding pair $(z_{\mathsf{in}}, \pi_{\mathsf{in}})$, $\mathbb{V}(\mathsf{vk}, z_{\mathsf{in}}, \pi_{\mathsf{in}}) = 1$; and (iii) $D(\pi) = 1$.

[27] In this paper, we construct PCD systems from preprocessing zk-SNARKs. Hence, analogously to preprocessing zk-SNARKs (cf. Remark C.1) *perfect* zero knowledge can be achieved at the only expense of a negligible error in completeness.

E Scalable zk-SNARKs for random-access machines

At high-level, a zk-SNARK for random-access machines is a cryptographic primitive that provides short and easy-to-verify non-interactive zero-knowledge proofs of knowledge for the correct execution of programs. A public proving key is used to generate proofs, and a public verification key is used to verify them; the two keys are jointly generated once, and can then be used any number of times.

In this work, we seek, and obtain an implementation of, zk-SNARKs that are **scalable**, i.e., that are:

- **Fully succinct.** This property requires that a *single* pair of keys suffices for computations of *any* (polynomial) size. In particular, the time to generate a key pair is short (i.e., bounded by a fixed polynomial in the security parameter) and so is the key length.
- **Incrementally computable.** This property requires that proof generation is carried out *incrementally*, along the original computation, by updating, at each step, a proof of correctness of the computation so far.

Below, we informally define fully-succinct zk-SNARKs for random-access machines, as well as the additional property of incremental computation. We refer the reader to, e.g., [BCCT13] for a formal treatment. (Also see Remark C.2 for a technical comment that applies here too.)

A **fully-succinct zk-SNARK** for random-access machines (see Appendix A.2) is a triple of polynomial-time algorithms $(G^\star, P^\star, V^\star)$ working as follows.

- $G^\star(1^\lambda, \mathbf{M}) \to (\mathsf{pk}, \mathsf{vk})$. On input a security parameter λ (presented in unary) and a random-access machine $\mathbf{M}$, the *key generator* $G^\star$ probabilistically samples a proving key pk and a verification key vk. We assume, without loss of generality, that pk contains (a description of) the machine $\mathbf{M}$.

The keys pk and vk are published as public parameters and can be used, any number of times, to prove/verify membership of instances in the language $\mathcal{L}_{\mathbf{M}}$ of accepting computations on $\mathbf{M}$ (see Definition A.2). The key generator $G^\star$ is thus *succinct and universal* (i.e., it does not depend on the program $\mathcal{P}$, or even computation size, but only on the machine $\mathbf{M}$ used to run programs). The keys pk and vk are used as follows.[28]

- $P^\star(\mathsf{pk}, \mathcal{P}, T, \mathcal{G}) \to \pi$. On input a program $\mathcal{P}$, time bound T, and auxiliary input $\mathcal{G}$ such that $\mathbf{M}(\mathcal{P}; \mathcal{G})$ accepts in $\leq T$ steps, the *prover* $P^\star$ outputs a non-interactive proof π for the statement "$(\mathcal{P}, T) \in \mathcal{L}_{\mathbf{M}}$".
- $V^\star(\mathsf{vk}, \mathcal{P}, T, \pi) \to b$. On input a program $\mathcal{P}$, time bound T, and proof π, the *verifier* $V^\star$ outputs $b = 1$ if he is convinced by π that $(\mathcal{P}, T) \in \mathcal{L}_{\mathbf{M}}$.

The triple $(G^\star, P^\star, V^\star)$ satisfies the following properties.

Completeness. The honest prover can convince the verifier for any instance in the language. Namely, for every security parameter λ, random-access machine $\mathbf{M}$, and instance $(\mathcal{P}, T) \in \mathcal{L}_{\mathbf{M}}$ with a witness $\mathcal{G}$,

$$\Pr\left[V^\star(\mathsf{vk}, \mathcal{P}, T, \pi) = 1 \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{vk}) \leftarrow G^\star(1^\lambda, \mathbf{M}) \\ \pi \leftarrow P^\star(\mathsf{pk}, \mathcal{P}, T, \mathcal{G})\end{array}\right] = 1 \ .$$

Succinctness. For every security parameter λ, random-access machine $\mathbf{M}$, and $(\mathsf{pk}, \mathsf{vk}) \in G^\star(1^\lambda, \mathbf{M})$,

- an honestly-generated proof π has $O_{\lambda,\mathbf{M}}(1)$ bits;
- $V^\star(\mathsf{vk}, \mathcal{P}, T, \pi)$ runs in time $O_{\lambda,\mathbf{M}}(|\mathcal{P}| + \log T)$.

[28] Both pk and vk are public and only consist of $O_{\lambda,\mathbf{M}}(1)$ bits; so, one could think of them as a single *common reference string* $\mathsf{crs} := (\mathsf{pk}, \mathsf{vk})$. We choose not to do so, because it will be more natural to think of them as separate data structures.

Above, $O_{\lambda,\mathbf{M}}$ hides a (fixed) polynomial factor in λ and $|\mathbf{M}|$. (In our implementation, these will be constants.)

Proof of knowledge (and soundness). If the verifier accepts a proof for a polynomial-size computation, the prover "knows" a witness for the instance. (Thus, soundness holds.) Namely, for every constant $c > 0$ and every polynomial-size adversary A there is a polynomial-size witness extractor E such that, for every large enough security parameter λ, for every random-access machine $\mathbf{M}$,

$$\Pr\left[\begin{array}{c|r} T \le \lambda^c & (\mathsf{pk},\mathsf{vk}) \leftarrow G^\star(1^\lambda,\mathbf{M}) \\ V^\star(\mathsf{vk},\mathcal{P},T,\pi)=1 & (\mathcal{P},T,\pi) \leftarrow A(\mathsf{pk},\mathsf{vk}) \\ \big((\mathcal{P},T),\mathcal{G}\big) \notin \mathcal{R}_{\mathbf{M}} & \mathcal{G} \leftarrow E(\mathsf{pk},\mathsf{vk}) \end{array}\right] \le \mathsf{negl}(\lambda) \ .$$

Statistical zero knowledge. An honestly-generated proof is statistical zero knowledge.[29] Namely, there is a polynomial-time stateful simulator S such that, for all stateful distinguishers D, the following two probabilities are negligibly-close:

$$\Pr\left[\begin{array}{c|r} \big((\mathcal{P},T),\mathcal{G}\big) \in \mathcal{R}_{\mathbf{M}} & \mathbf{M} \leftarrow D(1^\lambda) \\ \mathcal{D}(\pi)=1 & (\mathsf{pk},\mathsf{vk}) \leftarrow G^\star(1^\lambda,\mathbf{M}) \\ & (\mathcal{P},T,\mathcal{G}) \leftarrow D(\mathsf{pk},\mathsf{vk}) \\ & \pi \leftarrow P^\star(\mathsf{pk},\mathcal{P},T,\mathcal{G}) \end{array}\right] \text{ and } \Pr\left[\begin{array}{c|r} \big((\mathcal{P},T),\mathcal{G}\big) \in \mathcal{R}_{\mathbf{M}} & \mathbf{M} \leftarrow D(1^\lambda) \\ D(\pi)=1 & (\mathsf{pk},\mathsf{vk}) \leftarrow S(1^\lambda,\mathbf{M}) \\ & (\mathcal{P},T,\mathcal{G}) \leftarrow D(\mathsf{pk},\mathsf{vk}) \\ & \pi \leftarrow S(\mathcal{P},T) \end{array}\right] .$$

Finally, a fully-succinct zk-SNARK is also **incrementally computable** if there exist two algorithms, a *computation supervisor* SV and a *sub-prover* SP, such that, for every security parameter λ, random-access machine $\mathbf{M}$, instance $(\mathcal{P},T) \in \mathcal{L}_{\mathbf{M}}$ with a witness $\mathcal{G} = (g_0,\ldots,g_{T-1})$, key pair $(\mathsf{pk},\mathsf{vk}) \in G^\star(1^\lambda,\mathbf{M})$, and letting $\pi_T := P^\star(\mathsf{pk},\mathcal{P},T,\mathcal{G})$, the following holds.

- For $i = 1,\ldots,T$, $\pi_i = \mathsf{SP}(\mathsf{pk},\mathsf{aux}_i,\pi_{i-1})$.
- For $i = 1,\ldots,T$, aux_i is the final state of memory when $\mathsf{SV}(\mathbf{M},g_i)$ has read-write random access to a memory initialized to the state aux_{i-1}. Moreover, each aux_i has size $O_{\lambda,\mathbf{M}}(S_i)$, where S_i is the space usage of $\mathbf{M}(\mathcal{P};\mathcal{G})$ at time i.[30]
- The proof π_0 is defined as $\perp$, and aux_0 as $\mathcal{P}$.

In particular, SV and SP have time and space complexity $O_{\lambda,\mathbf{M}}(1)$; these costs are incurred each time a new proof is generated from an old one.

E.1 Known constructions and security

Theoretical constructions of fully-succinct zk-SNARKs are known, based on various cryptographic assumptions [Mic00, Val08, BCCT13]. Despite achieving essentially-optimal asymptotics [BFLS91, BGHSV05, BCGT13b, BCGT13a, BCCT13] no implementations of them have been reported in the literature to date.

Of the above, the only approach that also achieves incremental computation is the one of Bitansky et al. [BCCT13], which we follow in this paper. Security in [BCCT13] is based on the security of *preprocessing* zk-SNARKs (see Appendix C.1) and collision-resistant hash functions.

[29] In this paper, we construct fully-succinct zk-SNARKs from preprocessing ones. Hence, analogously to preprocessing zk-SNARKs (cf. Remark C.1) *perfect* zero knowledge can be achieved at the only expense of a negligible error in completeness.

[30] In our implementation, aux_i has size $\approx S_i$. Thus, the effective space overhead, compared to the original computation, is the *additive, constant* cost to run SV and SP.

References

[ADLM+08] Yuriy Arbitman, Gil Dogon, Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFTX: a proposal for the SHA-3 standard, 2008.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, STOC '96, pages 99–108, 1996.

[AM93] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–68, 1993.

[AV77] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *Proceedings on 9th Annual ACM Symposium on Theory of Computing*, STOC '77, pages 30–41, 1977.

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *Proceedings of the 24th Annual International Cryptology Conference*, CRYPTO '04, pages 443–459, 2004.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 111–120, 2013.

[BCGG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages ???–???, 2014.

[BCGT13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference*, ITCS '13, pages 401–414, 2013.

[BCGT13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 585–594, 2013.

[BCGTV13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 90–108, 2013.

[BCGTV13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. TinyRAM architecture specification v2.00, 2013. URL: `http://scipr-lab.org/tinyram`.

[BCIOP13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages 315–333, 2013.

[BCP13] Nir Bitansky, Ran Canetti, and Omer Paneth. How to construct extractable one-way functions against uniform adversaries. Cryptology ePrint Archive, Report 2013/468, 2013.

[BCPR13] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. Indistinguishability obfuscation vs. auxiliary-input extractable functions: One must fall. Cryptology ePrint Archive, Report 2013/641, 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium*, Security '14, pages ???–???, 2014. Available at `http://eprint.iacr.org/2013/879`.

[BDSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.

[BEGKN91] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, FOCS '91, pages 90–99, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32, 1991.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, 1988.

[BFRS+13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles*, SOSP '13, pages 341–357, 2013.

[BGHSV05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pages 120–134, 2005.

[BGOhM07] Paulo S. Barreto, Steven D. Galbraith, Colm Ó hÉigeartaigh, and Scott Michael. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.

[BKLS02] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *Proceedings of the 22Nd Annual International Cryptology Conference*, CRYPTO '02, pages 354–368, 2002.

[BLPRS13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the 45th Annual ACM Symposium on Symposium on Theory of Computing*, STOC '13, pages 575–584, 2013.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In *Proceedings of the 3rd International Conference on Security in Communication Networks*, SCN '02, pages 257–267, 2003.

[BLS04] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.

[BS10] Naomi Benger and Michael Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In *Proceedings of the 3rd International Conference on Arithmetic of Finite Fields*, WAIFI '10, pages 180–195, 2010.

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: Homomorphic encryption for restricted computations. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 350–366, 2012.

[BW05] Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[CFAD+12] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2 edition, 2012.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*, ITCS '12, pages 90–112, 2012.

[CP01] C. Cocks and Richard G. E. Pinch. Identity-based cryptosystems based on the weil pairing. Unpublished manuscript, 2001.

[CR72] Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, 1972.

[CRR11] Ran Canetti, Ben Riva, and Guy N. Rothblum. Practical delegation of computation using multiple servers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 445–454, 2011.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, ICS '10, pages 310–331, 2010.

[CT12] Alessandro Chiesa and Eran Tromer. Proof-carrying data: Secure computation on untrusted platforms (high-level description). *The Next Wave: The National Security Agency's review of emerging technologies*, 19(2):40–46, 2012.

[CTV13] Stephen Chong, Eran Tromer, and Jeffrey A. Vaughan. Enforcing language semantics using proof-carrying data. Cryptology ePrint Archive, Report 2013/513, 2013.

[DEM05] Régis Dupont, Andreas Enge, and François Morain. Building curves with arbitrary small MOV degree over finite prime fields. *Journal of Cryptology*, 18(2):79–89, 2005.

[ES10] Andreas Enge and Andrew V. Sutherland. Class invariants by the CRT method. In *Proceedings of the 9th International Symposium on Algorithmic Number Theory*, ANTS '10, pages 142–156, 2010.

[FMR06] Gerhard Frey, Michael Müller, and Hans-Georg Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 2006.

[FR94] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, 1994.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of the 6th Annual International Cryptology Conference*, CRYPTO '87, pages 186–194, 1987.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[Gen04] Rosario Gennaro. Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In *Proceedings of the 24th Annual International Cryptology Conference*, CRYPTO '04, pages 220–236, 2004.

[GGH96] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. Technical report, 1996. ECCC TR95-042.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '13, pages 1–17, 2013.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '13, pages 626–645, 2013.

[GHS02] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the Tate pairing. In *Proceedings of the 5th International Symposium on Algorithmic Number Theory*, ANTS '02, pages 324–337, 2002.

[GLS09] Parikshit Gopalan, Shachar Lovett, and Amir Shpilka. On the complexity of boolean functions in different characteristics. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity*, CCC '09, pages 173–183, 2009.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '10, pages 321–340, 2010.

[GS06] R. Granger and Nigel Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006.

[GS10] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *Proceedings of the 13th international conference on Practice and Theory in Public Key Cryptography*, PKC'10, pages 209–223, 2010.

[GSCvDD03] Blaise Gassend, G. Edward Suh, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Caches and hash trees for efficient memory integrity verification. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, HPCA '03, pages 295–306, 2003.

[GSMB03] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '03, 2003.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, STOC '11, pages 99–108, 2011.

[HSV06] F. Hess, N. P. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[JJ98] Antoine Joux and Stern Jacques. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.

[KKC13] Taechan Kim, Sungwook Kim, and Jung Hee Cheon. On the final exponentiation in Tate pairing computations. *IEEE Transactions on Information Theory*, 59(6):4033–4041, 2013.

[KRSWF03] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the 2003 Conference on File and Storage Technologies*, FAST '03, 2003.

[KT08] Koray Karabina and Edlyn Teske. On prime-order elliptic curves with embedding degrees $k = 3$, 4, and 6. In *Proceedings of the 8th International Conference on Algorithmic Number Theory*, ANTS-VIII '08, pages 102–117, 2008.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*, TCC '12, pages 169–189, 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '13, pages 41–60, 2013.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming*, ICALP '06, pages 144–155, 2006.

[LMN10] Kristin Lauter, Peter L. Montgomery, and Michael Naehrig. An analysis of affine coordinates for pairing computation. In *Proceedings of the 4th International Conference on Pairing-based Cryptography*, Pairing '10, pages 1–20, 2010.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: a modest proposal for FFT hashing. In *Proceedings of the 15th International Workshop on Fast Software Encryption*, FSE '08, pages 54–72, 2008.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, second edition edition, 1997.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. Preliminary version appeared in FOCS '94.

[MNT01] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 84(5):1234–1243, 2001.

[MOV91] Alfred Menezes, Tatsuaki Okamoto, and Scott Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 80–89, 1991.

[MS01] David Mazières and Dennis Shasha. Don't trust your file server. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, HotOS '01, pages 113–118, 2001.

[MVS00] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to build a trusted database system on untrusted storage. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation*, OSDI '00, pages 10–10, 2000.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, STOC '89, pages 33–43, 1989.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, STOC '90, pages 427–437, 1990.

[Odl85] Andrew M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Proceedings of the 3rd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '85, pages 224–314, 1985.

[PGHR13] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, Oakland '13, pages 238–252, 2013.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance. *Journal IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Pol78] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[Pol00] John M. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13:437–447, 2000.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *Proceedings of the 3rd Conference on Theory of Cryptography*, TCC '06, pages 145–166, 2006.

[Raz87] Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, STOC '90, pages 387–394, 1990.

[SA98] Takakazu Satoh and Kiyomichi Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Mathematici Universitatis Sancti Pauli*, 47(1):81–92, 1998.

[SB06] Michael Scott and Paulo S. Barreto. Generating more MNT elliptic curves. *Designs, Codes and Cryptography*, 38(2):209–217, 2006.

[SBCDPK09] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Proceedings of the 3rd International Conference Palo Alto on Pairing-Based Cryptography*, Pairing '09, pages 78–88, 2009.

[SBVB+13] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th EuoroSys Conference*, EuroSys '13, pages 71–84, 2013.

[SBW11] Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Toward practical and unconditional verification of remote computations. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS '11, pages 29–29, 2011.

[Sco05] Michael Scott. Computing the Tate pairing. In *Proceedings of the The Cryptographers' Track at the RSA Conference 2005*, CT-RSA '05, pages 293–304, 2005.

[Sco07] Michael Scott. Implementing cryptographic pairings. In *Proceedings of the 1st First International Conference on Pairing-Based Cryptography*, Pairing '07, pages 177–196, 2007.

[Sem98] Igor A. Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Mathematics of Computation*, 67(221):353–356, 1998.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. *Symposia on Pure Mathematics*, 20:415–440, 1971.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer, 2 edition, 2009.

[Sma99] Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12(3):193–196, 1999.

[SMBW12] Srinath Setty, Michael McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *Proceedings of the 2012 Network and Distributed System Security Symposium*, NDSS '12, pages ???–???, 2012.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 77–82, 1987.

[Sol03] Jerome A. Solinas. Id-based digital signature algorithms. `http://cacr.uwaterloo.ca/conferences/2003/ecc2003/solinas.pdf`, 2003.

[SS11] Joseph H. Silverman and Katherine E. Stange. Amicable pairs and aliquot cycles for elliptic curves. *Experimental Mathematics*, 20(3):329–357, 2011.

[Sut11] Andrew V. Sutherland. Computing Hilbert class polynomials with the Chinese remainder theorem. *Mathematics of Computation*, 80(273):501–538, 2011.

[Sut12] Andrew V. Sutherland. Accelerating the CM method. *LMS Journal of Computation and Mathematics*, 15:172–204, 12 2012.

[SVPB+12] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Security Symposium*, Security '12, pages 253–268, 2012.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 71–89, 2013.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. *CoRR*, abs/1202.1350, 2012.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Theory of Cryptography Conference*, TCC '08, pages 1–18, 2008.

[Ver10] Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman & Hall/CRC, 2 edition, 2008.