

ZKProof Community Reference

Version 0.3

July 17, 2022

This document is a work in progress.

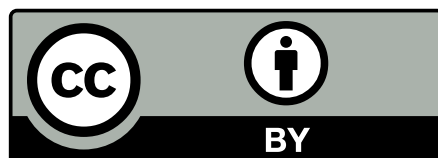
Feedback and contributions are welcome.

Find the latest version at <https://zkproof.org>.

Send your comments to editors@zkproof.org.



ZKPROOF



[Attribution 4.0 International \(CC BY 4.0\)](https://creativecommons.org/licenses/by/4.0/)

Abstract

12 Zero-knowledge proofs enable proving mathematical statements while maintaining the confiden-
13 tiality of supporting data. This can serve as a privacy-enhancing cryptographic tool in a wide
14 range of applications, but its usability is dependent on secure, practical and interoperable deploy-
15 ments. This ZKProof Community Reference — an output of the ZKProof standardization effort
16 — intends to serve as a reference for the development of zero-knowledge-proof technology. The
17 document arises from contributions by the community and for the community. It covers theoretical
18 aspects of definition and theory, as well as practical aspects of implementation and applications.

19 **Keywords:** cryptography; interoperability; privacy, security; standards; zero-knowledge proofs.

20 **About this version.** This is the version 0.3 of the ZKProof Community Reference. It results from
21 many contributions, as described in the [Acknowledgments](#), in the [Version history](#), and in the doc-
22 umentation of previous ZKProof workshops. While at a $0.x$ version, this document should be
23 considered as an incomplete draft, serving as a basis for further development. Reaching a future
24 stable version requires additional revision and substantial contributions from the community.

25 **Citing this version:** ZKProof. *ZKProof Community Reference. Version 0.3*. Ed. by D. Benarroch,
26 L. Brandão, M. Maller, and E. Tromer. Pub. by zkproof.org. July 2022. Updated versions at
27 <https://docs.zkproof.org/reference>

About this community reference

The “ZKProof Community Reference” arises within the scope of the ZKProof open initiative, which seeks to mainstream zero-knowledge proof (ZKP) cryptography. This is an inclusive community-driven process that focuses on interoperability and security, aiming to advance trusted specifications for the implementation of ZKP schemes and protocols.

ZKProof holds annual workshops, attended by world-renowned cryptographers, practitioners and industry leaders. These events are a forum for discussing new proposals, reviewing cutting edge projects, and advancing reference material. That is the genesis of this document, which intends to be a community-built reference for understanding and aiding the development of ZKP systems.

The following items provide guidance for the expected development process of this document, which is open to contributions from and for the community.

Purpose. The purpose of developing the ZKProof Community Reference document is to provide, within the principles laid out by the [ZKProof charter](#), a reference for the development of zero-knowledge-proof technology that is secure, practical and interoperable.

Aim. The aim of the document is to consolidate reference material developed and/or discussed in collaborative processes during the ZKProof workshops. The document intends to be accessible to a large audience, including the general public, the media, the industry, developers and cryptographers.

Scope. The document intends to cover material relevant for its purpose — the development of secure, practical and interoperable technology. The document can also elaborate on introductory concepts or works, to enable an easier understanding of more advanced techniques. When a focus is chosen from several alternative options, the document should include a rationale describing comparative advantages, disadvantages and applicability. However, the document does not intend to be a thorough survey about ZKPs, and does not need to cover every conceivable scenario.

Format. To achieve its accessibility goal, and considering its wide scope, the document favors the inclusion of: a well defined structure (e.g., chapters, sections, subsections); introductory descriptions (e.g., an executive summary and one introduction per chapter); illustrative examples covering the main concepts; enumerated recommendations and requirements; summarizing tables; glossary of technical terms; appropriate references for presented claims and results.

Editorial methodology. The development process of this community reference is proposed to happen in cycles of four phases:

- (i) **open discussion** during ZKProof workshops, with corresponding annotations to serve as reference for subsequent development;
- (ii) **content development**, by voluntary *contributors*, according to a set of contribution proposals and during a defined period;
- (iii) **integration** of contributions into the document, by the *editors*;
- (iv) **public feedback** about the state of the document, to be used as a basis of development in the next cycle.

The team of editors coordinates the process, promoting transparency by means of public calls for contributions and feedback, using editorial discretion towards the improvement of the document quality, and enabling an easy way to identify the changes and their rationale.

ZKProof charter

ZKProof Charter (Boston, May 10th and 11th 2018).

The goal of the ZKProof Standardization effort is to advance the use of Zero Knowledge Proof technology by bringing together experts from industry and academia. To further the goals of the effort, we set the following guiding principles:

- The initiative is aimed at producing documents that are open for all and free to use.
 - As an open initiative, all content issued from the ZKProof Standards Workshop is under Creative Commons Attribution 4.0 International license.
- We seek to represent all aspects of the technology, research and community in an inclusive manner.
- Our goal is to reach consensus where possible, and to properly represent conflicting views where consensus was not reached.
- As an open initiative, we wish to communicate our results to the industry, the media and to the general public, with a goal of making all voices in the event heard.
 - Participants in the event might be photographed or filmed.
 - We encourage you to tweet, blog and share with the hashtag #ZKProof. Our official twitter handle is @ZKProof.

For further information, please refer to contact@zkproof.org

Editors note: The requirement of a Creative Commons license was initially within the scope of the 1st ZKProof workshop. The section below (about intellectual property expectations) widens the scope to cover this Community reference and beyond.

Intellectual property — expectations on disclosure and licensing

ZKProof is an open initiative that seeks to promote the secure and interoperable use of zero-knowledge proofs. To foster open development and wide adoption, it is valuable to promote technologies with open-source implementations, unencumbered by royalty-bearing patents. However, some useful technologies may fall within the scope of patent claims. Since ZKProof seeks to represent the technology, research and community in an inclusive manner, it is valuable to set expectations about the disclosure of intellectual property and the handling of patent claims.

The members of the ZKProof community are hereby strongly encouraged to provide information on known patent claims (their own and those from others) potentially applicable to the guidance, requirements, recommendations, proposals and examples provided in ZKProof documentation, including by disclosing known pending patent applications or any relevant unexpired patent. Particularly, such disclosure is promptly required from the patent holders, or those acting on their behalf, as a condition for providing content contributions to the “Community Reference” and to “Proposals” submitted to ZKProof for consideration by the community. The ZKProof documentation will be updated based on received disclosures about pertinent patent claims.

ZKProof aims to produce documents that are open for all and free to use. As such, the content produced for publication within the context of the ZKProof Standardization effort should be made available under a Creative Commons Attribution 4.0 International license. Furthermore, any technology that is promoted in said ZKProof documentation and that falls within patent claims should be made available under licensing terms that are reasonable, and demonstrably free of unfair discrimination, preferably allowing free open-source implementations.

Please email relevant information to editors@zkproof.org.

Page intentionally blank

109 **Contents**

110 **Contents**

111 Abstract i

112 About this version i

113 About this community reference ii

114 ZKProof charter iii

115 Intellectual property — expectations on disclosure and licensing iii

116 Contents v

117 Executive summary xi

118 **1 Security** **1**

119 1.1 Introduction 1

120 1.1.1 What is a zero-knowledge proof? 1

121 1.1.2 Requirements for a ZK proof system specification 2

122 1.2 Terminology 2

123 1.3 Specifying Statements for ZK 3

124 1.3.1 Circuit representation 4

125 1.3.2 R1CS representation 4

126 1.3.3 Types of relations 5

127 1.4 ZKPs of knowledge vs. ZKPs of membership 6

128 1.4.1 Example: ZKP of knowledge of a discrete logarithm (discrete-log) 6

129 1.4.2 Example: ZKP of knowledge of a hash pre-image 7

130 1.4.3 Example: ZKP of membership for graph non-isomorphism 7

131 1.5 Syntax 8

132 1.5.1 Prove 8

133 1.5.2 Verify 8

134 1.5.3 Setup 9

135 1.6 Definition and Properties 10

136 1.6.1 Completeness 10

137 1.6.2 Soundness 11

138 1.6.3 Proof of knowledge 12

139 1.6.4 Zero knowledge 12

140 1.6.5 Advanced security properties 13

141	1.6.6	Transferability vs. deniability	14
142	1.6.7	Examples of setup and trust	14
143	1.7	Assumptions	15
144	1.8	Efficiency	16
145	1.8.1	Characterization of security properties	17
146	1.8.2	Computational security levels for benchmarking	17
147	1.8.3	Statistical security levels for benchmarking	18
148	2	Paradigms	19
149	2.1	Background	19
150	2.1.1	Information-theoretic (IT) proof system	19
151	2.1.2	Cryptographic compiler (CC)	20
152	2.1.3	Arithmetization	20
153	2.1.4	Complexity and features	21
154	2.2	Information-Theoretic (IT) Proof Systems	22
155	2.2.1	Summary	23
156	2.2.2	Probabilistically Checkable Proof (PCP)	24
157	2.2.2.1	Classical 3-coloring proof	24
158	2.2.2.2	From PCP theorem	25
159	2.2.3	Linear PCP	26
160	2.2.3.1	Hadamard based	26
161	2.2.3.2	From QAP (R1CS)	27
162	2.2.3.3	Line-Point	29
163	2.2.3.4	Fully Linear PCP	29
164	2.2.4	MPC in the head	30
165	2.2.5	Interactive Oracle Proof (IOP)	31
166	2.2.5.1	Interactive PCP	31
167	2.2.5.2	Fast RS IOPP	32
168	2.2.6	Linear IOP	32
169	2.2.6.1	IP based	32
170	2.2.6.2	Polynomial IOP	33
171	2.2.6.3	Fully Linear IOP	34
172	2.2.7	Ideal Linear Commitment (ILC)	35
173	2.3	Cryptographic Compilers (CC)	36
174	2.3.1	CC for zk-PCP	36
175	2.3.2	CC for Linear PCP (LPCP)	38
176	2.3.3	CC for MPC-in-the-Head	39

177	2.3.4	CC for IOP	39
178	2.3.4.1	CC for Interactive PCP	40
179	2.3.5	CC for Linear IOP	40
180	2.3.5.1	CC for Fully Linear IOP	40
181	2.3.5.2	CC for PIOP: Polynomial Commitments	40
182	2.3.6	CC for ILC	41
183	2.4	Specialized ZK Proofs	41
184	2.5	Proof Composition	42
185	2.6	Interactivity	42
186	2.6.1	Advantages of Interactive Proof and Argument Systems	42
187	2.6.2	Disadvantages of Interactive Proof and Argument Systems	44
188	2.6.3	Nuances on transferability vs. interactivity	44
189	2.6.3.1	(Non)-Transferability/Deniability of Zero-Knowledge Proofs	46
190	3	Implementation	47
191	3.1	Overview	47
192	3.1.1	What this chapter is NOT about:	47
193	3.2	Backends: Cryptographic System Implementations	47
194	3.3	Frontends: Constraint-System Construction	48
195	3.4	APIs and File Formats	49
196	3.4.1	Generic API	49
197	3.4.2	R1CS File Format	51
198	3.5	Benchmarks	53
199	3.5.1	What metrics and components to measure	53
200	3.5.2	How to run the benchmarks	53
201	3.5.3	What benchmarks to run	54
202	3.6	Correctness and Trust	56
203	3.6.1	Considerations	56
204	3.6.2	SRS Generation	58
205	3.6.3	Contingency plans	60
206	3.7	Extended Constraint-System Interoperability	60
207	3.7.1	Statement and witness formats	60
208	3.7.2	Statement semantics, variable representation & mapping	61
209	3.7.3	Witness reduction	61
210	3.7.4	Gadgets interoperability	62
211	3.7.5	Procedural interoperability	62
212	3.7.6	Proof interoperability	62

213	3.7.7	Common reference strings	63
214	3.8	Future goals	63
215	3.8.1	Interoperability	63
216	3.8.2	Frontends and DSLs	64
217	3.8.3	Verification of implementations	64
218	4	Applications	65
219	4.1	Introduction	65
220	4.2	Types of verifiability	66
221	4.3	Previous works	67
222	4.4	Gadgets within predicates	67
223	4.5	Identity framework	71
224	4.5.1	Overview	71
225	4.5.2	Motivation for Identity and Zero Knowledge	71
226	4.5.3	Terminology / Definitions	71
227	4.5.4	The Protocol Description	72
228	4.5.5	A use-case example of credential aggregation	77
229	4.6	Asset Transfer	79
230	4.6.1	Privacy-preserving asset transfers and balance updates	79
231	4.6.2	Zero-Knowledge Proofs in the asset-tracking model	80
232	4.6.3	Zero-Knowledge proofs in the balance model	83
233	4.7	Regulation Compliance	85
234	4.7.1	Overview	85
235	4.7.2	An example in depth: Proof of compliance for aircraft	86
236	4.7.3	Protocol high level	87
237	4.8	Conclusions	88
238		Acknowledgments	89
239		References	91
240	A	Acronyms and glossary	103
241	A.1	Acronyms	103
242	A.2	Glossary	103
243	B	Version history	105

244 **List of Figures**

245 Figure 2.1: Various IT proof systems 22

246 Figure 3.1: Parties and objects in a NIZK 50

247 **List of Tables**

248 Table 1.1: Example scenarios for zero-knowledge proofs 3

249 Table 3.1: APIs and interfaces by types of universality and preprocessing 50

250 Table 4.1: List of gadgets 68

251 Table 4.2: Commitment gadget 68

252 Table 4.3: Signature gadget 69

253 Table 4.4: Encryption gadget 69

254 Table 4.5: Distributed-decryption gadget 69

255 Table 4.6: Random-function gadget 69

256 Table 4.7: Set-membership gadget 70

257 Table 4.8: Mix-net gadget 70

258 Table 4.9: Generic-computation gadget 70

259 Table 4.10: Holder identification 74

260 Table 4.11: Issuer identification 74

261 Table 4.12: Credential Issuance 75

262 Table 4.13: Credential Revocation 75

Executive summary

264 Zero-knowledge proofs (ZKPs) are an important privacy-enhancing tool from cryptography. They
265 allow proving the veracity of a statement, related to confidential data, without revealing any in-
266 formation beyond the validity of the statement. ZKPs were initially developed by the academic
267 community in the 1980s, and have seen tremendous improvements since then. They have since
268 become feasible in practice for application in multiple domains of interest to the industry, and to
269 a large community of developers and researchers. ZKPs can have a positive impact in industries,
270 agencies, and for personal use, by allowing privacy-preserving applications where designated private
271 data can be made useful to third parties, despite not being disclosed to them.

272 The development of this reference document aims to serve the broader community, particularly
273 those interested in understanding ZKP systems, making an impact in their advancement, and
274 using related products. This is a step towards enabling wider adoption of ZKP technology, which
275 may precede the establishment of future standards. However, this document is not a substitution
276 for research papers, technical books, or standards. It is intended to serve as a reference handbook
277 of introductory concepts, basic techniques, implementation suggestions and application use-cases.

278 ZKP systems involve at least two parties: a prover and a verifier. The goal of the prover is to
279 convince the verifier that a *statement* is true, without revealing any additional information. For
280 example, suppose the prover holds a birthdate certificate digitally signed by an authority. In order
281 to access some service, the prover may have to prove being at least 18 years old, that is, that
282 the prover *knows* a certificate, tied to their identify and digitally signed by a trusted certification
283 authority, stating a birthdate consistent with the age claim. A ZKP allows this, without the prover
284 having to reveal the birthdate.

285 This document describes important aspects of the current state of the art in ZKP [security](#), [paradigms](#),
286 [implementation](#), and [applications](#). There are several use-cases and applications where ZKPs can add
287 value. To better assess this it is useful to benchmark implementations under several metrics, evaluate
288 tradeoffs between security and efficiency, and develop an interoperability basis. The security of a
289 proof system is paramount for the system users, but efficiency is also essential for user experience.

290 The “[Security](#)” chapter introduces the baseline terminology and security concepts of ZKP systems.
291 A ZKP system can be described with three components: **setup**, **prove**, **verify**. The **setup**
292 determines the initial state of the prover and the verifier, including private and common elements,
293 such as private and public keys, or a common reference string. The **prove** and **verify** components
294 are the prover and verifier’s algorithms, respectively, possibly interactive. Overall they need to
295 ensure three main security requirements: completeness, soundness, and zero-knowledge.

296 Completeness requires that if both **prove** and **verify** are correct, and if the statement is true, then
297 at the end of the interaction the verifier is convinced of this fact. Soundness requires that not even a
298 malicious prover can convince the verifier of a false statement. Zero knowledge requires that even a
299 malicious verifier cannot extract from the proof process any information beyond the truthfulness of
300 the given statement. Other properties and variations may be built in by design, as desired features,
301 such as succinctness, non-interactivity, transferability and composability, among others.

302 The “[Paradigms](#)” chapter presents an overview of paradigms for constructing zero-knowledge proofs
303 for general statements, namely those related to non-deterministic polynomial (NP) relations. The
304 presentation describes a modern perspective of composition of an *information theoretic* (IT) system,

305 based on ideal components, a *cryptographic compiler* (CC), where cryptographic elements come into
 306 play, and an *arithmetization* system to encode the actual public data (statement and instance) and
 307 private data (witness) that determine the specific proof taking place. The provided organization
 308 allows for a systematized covering/characterization of many modern ZKP systems, many of which
 309 enable succinct proofs. Future updates to this chapter should include additional examples for
 310 specialized languages/relations, which can follow a different approach.

311 The “**Implementation**” chapter focuses on devising a framework for the implementation of ZKPs,
 312 which is important for interoperability. One important aspect to consider upfront is the represen-
 313 tation of statements. In a ZKP protocol, the statement needs to be converted into a mathematical
 314 object. For example, in the case of proving that an age is at least 18, the statement is equivalent to
 315 proving that the private birthdate Y_1 - M_1 - D_1 (year-month-day) satisfies a relation with the present
 316 date Y_2 - M_2 - D_2 , namely that their distance is greater than or equal to 18 years. This simple example
 317 can be represented as a disjunction of conditions: $Y_2 > Y_1 + 18$, or $Y_2 = Y_1 + 18 \wedge M_2 > M_1$, or $Y_2 = Y_1 + 18 \wedge$
 318 $M_2 = M_1 \wedge D_2 \geq D_1$. An actual conversion suitable for ZKPs, namely for more complex statements, can
 319 pose an implementation challenge. There are nonetheless various techniques that enable converting
 320 a statement into a mathematical object, such as a circuit. This document gives special attention to
 321 representations based on a Rank-1 constraint system (R1CS) and quadratic arithmetic programs
 322 (QAP), which are adopted by several ZKP solutions in use today. Also, the document gives special
 323 emphasis to implementations of non-interactive proof systems.

324 The privacy enhancement offered by ZKPs can be applied to a wide range of scenarios. The “**Appli-**
 325 **cations**” chapter presents three use-cases that can benefit from ZKP systems: identity framework;
 326 asset transfer; regulation compliance. In a privacy-preserving identity framework, one can for ex-
 327 ample prove useful personal attributes, such as age and state of residency, without revealing more
 328 detailed personal data such as birthdate and address. In an asset-transfer setting, financial institu-
 329 tions that facilitate transactions usually require knowing the identities of the sender and receiver,
 330 and the asset type and amount. ZKP systems enable a privacy-preserving variant where the trans-
 331 action is performed between anonymous parties, while at the same time ensuring they and their
 332 assets satisfy regulatory requirements. In a regulation compliance setting, ZKPs enables an auditor
 333 to obtain proof that a process satisfies a number of requirements, without having to learn details
 334 about how they were achieved. These use cases, as well as a wide range of many other conceivable
 335 privacy-preserving applications, can be enabled by a common set of tools, or gadgets, for example
 336 including commitments, signatures, encryption and circuits.

337 The development of secure, practical, and interoperable ZKP applications requires a balanced inter-
 338 play between security concepts and implementation guidelines. Solutions provided by ZKP technol-
 339 ogy must be ensured by careful security practices and realistic assumptions. This document aims
 340 to summarize security properties and implementation techniques that help achieve these goals.

Chapter 1. Security

1.1 Introduction

1.1.1 What is a zero-knowledge proof?

A zero-knowledge proof (ZKP) makes it possible to prove a statement is true while preserving confidentiality of secret information [GMR89]. This makes sense when the veracity of the statement is not obvious on its own, but the prover knows relevant secret information (or has a skill, like super-computation ability) that enables producing a proof. The notion of secrecy is used here in the sense of prohibited leakage, but a ZKP makes sense even if the ‘secret’ (or any portion of it) is known apriori by the verifier(s).

There are numerous uses of ZKPs, useful for proving claims about confidential data, such as:

1. adulthood, without revealing the birth date;
2. solvency (not being bankrupt), without showing the portfolio composition;
3. ownership of an asset, without revealing or linking to past transactions;
4. validity of a chessboard configuration, without revealing the legal sequence of chess moves;
5. correctness (demonstrability) of a theorem, without revealing its mathematical proof.

Some of these claims (commonly known by the prover and verifier, and here described as informal *statements*) require a substrate (called *instance*, also commonly known by the prover and verifier) to support an association with the confidential information (called *witness*, known by the prover and to not be leaked during the proof process). For example, the proof of solvency (the statement) may rely on encrypted and certified bank records (the instance), and with the verifier knowing the corresponding decryption key and plaintext (the witness) as secrets that cannot be leaked. Table 1.1 in Section 1.2 differentiates these elements across several examples. In concrete instantiations, the exemplified ZKPs are specified by means of a more formal *statement of knowledge* of a witness.

FF1.1. Proposed future figure (illustration, diagram, ...):

Illustration of an interaction between prover and verifier, conveying the elements of the proof, including statement, instance, witness, message sending, and convincing

A ZKP system is a specification of how a prover and verifier can interact for the prover to convince the verifier that the statement is true. The proof system must be complete, sound and ZK.

- **Complete:** If the statement is true and both prover and verifier follow the protocol; the verifier will accept.
- **Sound:** If the statement is false, and the verifier follows the protocol; the verifier will not be convinced.
- **Zero-knowledge:** If the statement is true and the prover follows the protocol; the verifier

374 will not learn any confidential information from the interaction with the prover but the fact
375 the statement is true.

376 **Proofs vs. arguments.** The theory of ZKPs distinguishes between *proofs* and *arguments*, as
377 related to the computational power of the prover and verifier. *Proofs* need to be sound even against
378 computationally unbounded provers, whereas *arguments* only need to preserve soundness against
379 computationally bounded provers (often defined as probabilistic polynomial time algorithms). For
380 simplicity, “proof” is used hereafter to designate both *proofs* and *arguments*, although there are
381 theoretical circumstances where the distinction is relevant.

382 1.1.2 Requirements for a zero-knowledge proof system specification

383 A full proof system specification shall include:

- 384 1. Precise specification of the type of statements the proof system is designed to handle
- 385 2. Construction details, including the algorithms used by the prover and verifier
- 386 3. If applicable, a description of the setup used by the prover and verifier
- 387 4. Precise definitions of the security the proof system is intended to provide
- 388 5. A security analysis that proves the ZKP system satisfies the security goals, and a list of the
389 unproven assumptions that underpin security

390 Efficiency claims about a ZKP system must be reported fairly and accurately: should include the
391 relevant performance parameters for the intended usage; when compared with other ZKP systems,
392 should be based on a best effort to compare *apples to apples*.

393 1.2 Terminology

394 **Instance:** Input commonly known to both prover (P) and verifier (V), and used to support the
395 statement of what needs to be proven. This common input may either be local to the prover–verifier
396 interaction, or public in the sense of being known by external parties. Notation: x . (Some scientific
397 articles use “instance” and “statement” interchangeably, but we distinguish between the two.)

398 **Witness:** Private input to the prover. Others may or may not know something about the witness.
399 Notation: w .

400 **Relation:** Specification of relationship between instances and witness. A relation can be viewed
401 as a set of permissible pairs (instance, witness). Notation: R .

402 **Language:** Set of instances that appear as a permissible pair in R . Notation: L .

403 **Statement:** Defined by instance and relation. Claims the instance has a witness in the relation
404 (which is either true or false). Notation: $x \in L$.

405 **Security parameter:** Positive integer indicating the desired security level (e.g. 128 or 256) where
406 higher security parameter means greater security. In most constructions, distinction is made be-

407 tween computational security parameter and statistical security parameter. Notation: k (compu-
408 tational) or s (statistical).

409 **Setup:** The inputs given to the prover and to the verifier, apart from the instance x and the wit-
410 ness w . The setup of each party can be decomposed into a private component (“PrivateSetup $_P$ ” or
411 “PrivateSetup $_V$ ”, respectively not known to the other party) and a common component “Common-
412 Setup = CRS” (known by both parties), where CRS denotes a “common reference string” (required
413 by some zero-knowledge proof systems). Notation: $\text{setup}_P = (\text{PrivateSetup}_P, \text{CRS})$ and $\text{setup}_V =$
414 $(\text{PrivateSetup}_V, \text{CRS})$.”

415 For simplicity, some parameters of the setup are left implicit (possibly inside the CRS), such as the
416 security parameters, and auxiliary elements defining the language and relation. See more details
417 in Section 1.5.3. While the witness (w) and the instance (x) could be assumed as elements of the
418 setup of a concrete ZKP protocol execution, they are often distinguished in their own category. In
419 practice, the term “Setup” is often used with respect to the setup of a proof system that can then
420 be instantiated for multiple executions with varying instances (x) and witnesses (w).

421 Table 1.1 exemplifies at a high level a differentiation between the *statement*, the *instance* and the
422 *witness* elements for the initial examples mentioned in Section 1.1.1.

423 **Table 1.1:** Example scenarios for zero-knowledge proofs

#	Elements Scenarios	Statement being proven	Instance used as substrate	Witness treated as confidential
1	Legal age for purchase	I am an adult	Tamper-resistant identification chip	Birthdate and personal data (signed by a certification authority)
2	Hedge fund solvency	We are not bankrupt	Encrypted & certified bank records	Portfolio data and decryption key
3	Asset transfer	I own this <asset>	A blockchain or other commitments	Sequence of transactions (and secret keys that establish ownership)
4	Chessboard configuration	This <configuration> can be reached	(The rules of Chess)	A sequence of valid chess moves
5	Theorem validity	This <expression> is a theorem	(A set of axioms, and the logical rules of inference)	A sequence of logical implications

431 1.3 Specifying Statements for ZK

432 This document considers types of statements defined by a relation R between instances x and
433 witnesses w . The relation R specifies which pairs (x, w) are considered related to each other, and
434 which are not related to each other. The relation defines a matching language L consisting of
435 instances x that have a witness w in R .

436 A *statement* is either a *membership* claim of the form “ $x \in L$ ”, or a *knowledge* claim of the form “In
437 the scope of relation R , I know a witness for instance x .” For some cases, the *knowledge* and *member-*
438 *ship* types of statement can be informally considered interchangeable, but formally there are techni-
439 cal reasons to distinguish between the two notions. In particular, there are scenarios where a state-
440 ment of knowledge cannot be converted into a statement of membership, and vice-versa (as exem-

441 plified in Section 1.4). The examples in this document are often based on statements of knowledge.
 442 The relation R can for instance be specified as a program (e.g. in C or Java), which given inputs
 443 x and w decides to accept, meaning $(x, w) \in R$, or reject, meaning w is not a witness to $x \in L$.
 444 Examples of such specifications of the relation are detailed in the [Applications track](#). In the
 445 academic literature, relations are often specified either as random access memory (RAM) programs
 446 or through Boolean and arithmetic circuits, described below.

447 1.3.1 Circuit representation

448 A circuit is a directed acyclic graph (DAG) comprised of nodes and labels for nodes, as follows:

- 449 • **Input nodes:** Nodes with in-degree 0; they are labeled with some constant (e.g., 0, 1, ...) or
 450 with input variable names (e.g., v_1, v_2, \dots)
- 451 • **Output node:** A single node with out-degree 0.
- 452 • **Gates:** The internal nodes, describing a computation.

453 **Parameters.** Depending on the application, various parameters may be important, for instance
 454 the number of gates in the circuit, the number of instance variables n_x , the number of witness
 455 variables n_w , the circuit depth, or the circuit width.

456 **Boolean Circuit satisfiability.** The relation R has instances of the form $x = (C, v_1, \dots, v_{n_x})$ and
 457 witnesses $w = (w_1, \dots, w_{n_w})$. For (x, w) to be in the relation, C must be a circuit with fan-in 2
 458 gate nodes that are labeled with Boolean operations, e.g., XOR or AND, v_1, \dots, v_{n_x} must specify truth
 459 values for some of the input nodes, and w_1, \dots, w_{n_w} must specify truth values for the remaining
 460 input variables, such that when evaluating the circuit the output node becomes 1 (true).

461 **Arithmetic Circuit satisfiability.** The relation has instances of the form $x = (F, C, v_1, \dots, v_{n_x})$ and
 462 witnesses $w = (w_1, \dots, w_{n_w})$. For (x, w) to be in the relation, F must be a finite field (e.g., integers
 463 modulo a prime p), C must be a circuit with gate nodes that are labeled with field operations, i.e.,
 464 addition or multiplication, v_1, \dots, v_{n_x} must specify field elements for some of the input nodes, and
 465 w_1, \dots, w_{n_w} must specify field elements for the remaining input variables, such that when evaluating
 466 the circuit the output node becomes 1.

467 **FF1.2. Proposed future figure (illustration, diagram, ...):**

468 Illustrate the various components of a circuit, including the variable

469 1.3.2 R1CS representation

470 A rank-1 constraint system (R1CS) is a system of equations represented by a list of triplets $(\vec{a}, \vec{b}, \vec{c})$
 471 of vectors of elements of some field. Each triplet defines a “constraint” as an equation of the form
 472 $(A) \cdot (B) - (C) = 0$. Each of the three elements — (A), (B), (C) — in such equation is a linear
 473 combination (e.g., $(C) = c_1 \cdot s_1 + c_2 \cdot s_2 + \dots$) of variables s_i of the so called solution \vec{s} vector.

474 **R1CS satisfiability.** For all triplets $(\vec{a}, \vec{b}, \vec{c})$ of vectors in the R1CS, the solution vector \vec{s} must
 475 satisfy $\langle \vec{a}, \vec{s} \rangle \cdot \langle \vec{b}, \vec{s} \rangle - \langle \vec{c}, \vec{s} \rangle = 0$, where $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors. The first

476 element of \vec{s} is fixed to the constant 1 (instead of a variable), to enable encoding constants in the
 477 constraints. The remaining elements represent several kinds of variables:

- 478 • **Witness variables:** known only to the prover; represent external inputs to the constraint
 479 system — the witness of the ZK proof system.
- 480 • **Internal variables:** known only to the prover; internal to the constraint system (represent
 481 the inputs and outputs of multiplication gates);
- 482 • **Instance variables:** known by both prover and verifier.

483 A R1CS does not produce an output from an input (as for example a circuit does), but can be
 484 used to verify the correctness of a computation (e.g., performed by circuits with logic and/or
 485 arithmetic gates). The R1CS checks that the output variables (commonly known by both prover
 486 and verifier) are consistent with all other variables (possibly known only by the prover) in the
 487 solution vector. R1CS is only an intermediate representation, since the actual use in a ZKP system
 488 requires subsequent formulations (e.g., into a QAP) to enable verification without revealing the
 489 secret variables.

490 A R1CS can be used to represent a Boolean circuit satisfiability problem and also to verify compu-
 491 tations in arithmetic circuits. It is sufficient to observe that arbitrary circuits can be represented
 492 using multiplication and linear combination of polynomials, and these in turn correspond to R1CS
 493 constraints. For example:

494 **FF1.3. Proposed future figure (illustration, diagram, ...):**

495 Illustrate an R1CS

- 496 • **Boolean circuits operations:**
 - 497 – **NOT operation:** If x is a Boolean variable, then $1-x$ is the negation of x . Put differently,
 498 if x is 0 or 1, then $1-x$ is respectively 1 or 0.
 - 499 – **AND operation:** can be implemented as $(A) \times (B)$
 - 500 – **XOR operation ($c = a \text{ XOR } b$):** can be implemented as $(2 \cdot a) \times (b) = (a + b - c)$, or
 501 equivalently as $c = a + b - (a \text{ AND } b) * 2$
- 502 • **Arithmetic circuit operations:**
 - 503 – Multiplication gates are directly represented as equations of the form $a * b = c$.
 - 504 – Linear constraints are used to keep track of inputs and outputs across these gates, and
 505 to represent addition and multiplication-by-constants.

506 1.3.3 Types of relations

507 **Special purpose relations.** Circuit satisfiability is a complete problem within the non-deterministic
 508 polynomial (NP) class, i.e., it is NP-complete, but a relation does not have to be that. Examples
 509 of statements that appear in cryptographic usage include that a committed value falls in a certain
 510 range $[A; B]$ or belongs to a set S , that a ciphertext has plaintext 0 or that two ciphertexts encrypt
 511 the same value, that the prover has a secret key associated with a set of public verification keys for
 512 a signature scheme, etc.

513 **Setup-dependent relations.** Sometimes it is convenient to let the relation R take an additional
 514 input setup_R , i.e., let the relation contain triples (setup_R, x, w) . The input setup_R can be used
 515 to specify persistent information. For example, for arithmetic circuit satisfiability, if the same
 516 finite field \mathbb{F} and circuit C are used many times, then $\text{setup}_R = (\mathbb{F}, C)$ and $x = (v_1, \dots, v_{n_x})$. The
 517 input setup_R can also be used to capture trusted input the relation does not check, e.g., a trusted
 518 Rivest–Shamir–Adleman (RSA) modulus.

519 1.4 ZKPs of knowledge vs. ZKPs of membership

520 The theory of ZKPs distinguishes between two types of proofs, based on the type of statement (and
 521 also on the type of security properties — see Sections 1.6.2 and 1.6.3):

- 522 • A ZKP of knowledge (ZKPoK) proves the veracity of a *statement of knowledge*, i.e., it proves
 523 knowledge of private data that supports the statement, without revealing the former.
- 524 • A ZKP of membership proves the veracity of a *statement of membership*, i.e., that the *instance*
 525 belongs to the *language*, as related to the *statement*, but without revealing information that
 526 could not have been produced by a computationally bounded verifier.

527 The *statements* exemplified in Table 1.1 were expressed as facts, but each of them corresponds to
 528 a knowledge of a secret witness that supports the statement in the context of the instance. For
 529 example, the statement “I am an adult” in scenario 1 can be interpreted as an abbreviation of “I
 530 know a birthdate that is consistent with adulthood today, and I also know a certificate (signed by
 531 some trusted certification authority) associating the birthdate with my identity.”

532 The first three use-cases (adulthood, solvency and asset ownership) in Table 1.1 have instances
 533 with some kind of protection, such as physical access control, encryption, signature and/or com-
 534 mitments. The “chessboard configuration” and the “theorem validity” use-cases are different in
 535 that their instances do not contain any cryptographic support or physical protection. Each of
 536 those two statements can be seen as a claim of membership, in the sense of claiming that the ex-
 537 pression/configuration belongs respectively to the language of valid chessboard configurations (i.e.,
 538 reachable by a sequence of moves), or the language of theorems (i.e., of provable expressions). At
 539 the same time, a further specification of the statement can be expressed as a claim of knowledge
 540 of a sequence of legal moves or a sequence of logical implications.

541 1.4.1 Example: ZKP of knowledge of a discrete logarithm (discrete-log)

542 Consider the classical example of proving knowledge of a discrete-log [Sch90]. Let p be a large
 543 prime (e.g., with 4096 bits) of the form $p = 2q + 1$, where q is also a prime. Let g be a generator
 544 of the group $\mathbb{Z}_p^* = \{1, \dots, p - 1\} = \{g^i : i = 1, \dots, p - 1\}$ under multiplication modulo p . Assume
 545 that it is computationally infeasible to compute discrete-logs in this group, and that the primality
 546 of p and q has been verified by both prover and verifier. Let w be a secret element (the witness)
 547 known by the prover, and let $x = g^w \pmod{p}$ be the instance known by both the prover and verifier,
 548 corresponding to the following statement by the prover: “I know the discrete-log (base g) of the
 549 instance (x) , modulo p ” (in other words: “I know a secret exponent that raises the generator (g) into
 550 the instance (x) , modulo p ”). Consider now the relation $R = \{(x, w) : g^w = x \pmod{p}\}$. In this

551 case, the corresponding language $L = \{x : \exists w : (x, w) \in R\}$ is simply the set $\mathbb{Z}_p^* = \{1, 2, \dots, p - 1\}$,
 552 for which membership is self-evident (without any knowledge of w). In that sense, a proof of
 553 membership does not make sense (or can be trivially considered accomplished with even an empty
 554 bit string). Conversely, whether or not the prover knows a witness is a non-trivial matter, since
 555 the current publicly-known state of the art does not provide a way to compute discrete-logs in time
 556 polynomial in the size of the prime modulus (except if with a quantum computer). In summary,
 557 this is a case where a ZKPoK makes sense but a ZKP of membership does not.

558 1.4.2 Example: ZKP of knowledge of a hash pre-image

559 Consider a cryptographic hash function $H : \{0, 1\}^{512} \rightarrow \{0, 1\}^{256}$, restricted to binary inputs of
 560 length 512. In this definition of H , the set of all 256-bit strings is the *co-domain*, which might be
 561 a super-set of the *image* $L = \{H(x) : x \in \{0, 1\}^{512}\}$ (a.k.a. *range*) of H . Let w be a witness (hash
 562 pre-image), known by the prover and unpredictable to the verifier, for some instance $x = H(w)$
 563 that the prover presents to the verifier. Since a cryptographic hash function is one-way, there is
 564 significance in providing a ZKPoK of a pre-image, which proves knowledge of a witness in the re-
 565 lation $R = \{(x, w) : H(w) = x\}$. Such proof also constitutes directly a proof of membership in the
 566 language L , i.e., that the instance x is a member of the image of H . However, interestingly depend-
 567 ing on the known properties of H , this membership predicate might or might not be self-evident
 568 from the instance x .

- 569 • If H is known to have as image the set of all bit-strings of length 256 (i.e., if $L = \{0, 1\}^{256}$),
 570 then membership is self-evident. In this case a ZKP of membership is superfluous, since it is
 571 trivial to verify the property of a bit-string having 256 bits.
- 572 • H may instead have the property that an element x uniformly selected from the co-domain
 573 $\{0, 1\}^{256}$ is not in the image of H , with some noticeable probability (e.g., ≈ 0.368 , if H
 574 is modeled as a random function), and with the membership predicate being difficult to
 575 determine. In this setting it can be useful to have the ability to perform a ZKP of membership.

576 1.4.3 Example: ZKP of membership for graph non-isomorphism

577 In the theoretical context of provers with super-polynomial computation ability (e.g., unbounded),
 578 one can conceive a proof of membership without the notion of witness. Therefore, in this case the
 579 dual notion of a ZKP of knowledge does not apply. A classical example uses the language of pairs
 580 of non-isomorphic graphs [GMW91], for which the proof is about convincing a verifier that two
 581 graphs are not isomorphic. The classical example uses an interactive proof that does not follow
 582 from a witness, but rather from a super-ability, by the prover, in deciding isomorphism between
 583 graphs. The verifier challenges the prover to detect which of the two graphs is isomorphic to a
 584 random permutation of one of the two original graphs. If the prover decides correctly enough
 585 times, without ever failing, then the verifier becomes convinced of the non-isomorphism.

586 This document is not focused on settings that require provers with super-polynomial ability (in an
 587 asymptotic setting). However, this notion of ZKP of membership without witness still makes sense
 588 in other conceivable applications, namely within a concrete setting (as opposed to asymptotic).
 589 This may apply in contexts of proofs of work, or when provers are “supercomputers” or quantum

590 computers, possibly interacting with verifiers with significantly less computational resources. An-
 591 other conceivable setting is when a verifier wants to confirm whether the prover is able to solve a
 592 mathematical problem, for which the prover claims to have found a first efficient technique, e.g.,
 593 the ability to decide fast about graph isomorphism.

594 **FF1.4. Proposed future figure (illustration, diagram, ...):**

595 Illustrate a ZKP protocol for graph non-isomorphism

596 **1.5 Syntax**

597 A proof system (for a relation R defining a language L) is a protocol between a prover and a verifier
 598 sending messages to each other. The prover and verifier are defined by two algorithms, here called
 599 Prove and Verify. The algorithms Prove and Verify may be probabilistic and may keep internal
 600 state between invocations.

601 **1.5.1 Prove**($state, m$) \rightarrow ($state, p$)

602 The Prove algorithm in a given state receiving message m , updates its state and returns a message p .

- 603 • The initial state of Prove must include an instance x and a witness w . The initial state may
 604 also include additional setup information $setup_P$, e.g., $state = (setup_P, x, w)$.
- 605 • If receiving a special initialization message $m = \mathbf{start}$ when first invoked it means the prover
 606 is to initiate the protocol.
- 607 • If Prove outputs a special error symbol $p = \mathbf{error}$, it must output \mathbf{error} on all subsequent
 608 calls as well.

609 **1.5.2 Verify**($state, p$) \rightarrow ($state, m$)

610 The Verify algorithm in a given state receiving message p , updates its state and returns a message m .

- 611 • The initial state of Verify must include an instance x .
- 612 • The initial state of Verify may also include additional setup information $setup_V$, e.g., $state =$
 613 $(setup_V, x)$.
- 614 • If receiving a special initialization message $p = \mathbf{start}$, it means the verifier is to initiate the
 615 protocol.
- 616 • If Verify outputs a special symbol $m = \mathbf{accept}$, it means the verifier accepts the proof of the
 617 statement $x \in L$. In this case, Verify must return $m = \mathbf{accept}$ on all future calls.
- 618 • If Verify outputs a special symbol $m = \mathbf{reject}$, it means the verifier rejects the proof of the
 619 statement $x \in L$. In this case, Verify must return $m = \mathbf{reject}$ on all future calls.

620 The setup information setup_P and setup_V can take many forms. A common example found in the
 621 cryptographic literature is that $\text{setup}_P = \text{setup}_V = k$, where k is a security parameter indicating
 622 the desired security level of the proof system. It is also conceivable that setup_P and setup_V contain
 623 descriptions of particular choices of primitives to instantiate the proof system with, e.g., to use
 624 the SHA-256 hash function or to use a particular elliptic curve. The setup information may also
 625 be generated by a probabilistic process. For example: it may be that setup_P and setup_V include
 626 a common reference string; or, in the case of designated-verifier proofs, setup_P and setup_V may
 627 be correlated in a particular way. When we want to specifically refer to this process, we use a
 628 probabilistic setup algorithm **Setup**.

629 1.5.3 $\text{Setup}(params) \rightarrow (\text{setup}_R, \text{setup}_P, \text{setup}_V, auxi)$

630 **Input.** The **Setup** algorithm may take input parameters ($params$), such as:

- 631 • computational or statistical security parameters, indicating the desired security level;
- 632 • parameters specifying the size of the statements the proof system should work for;
- 633 • choices of cryptographic primitives, e.g., the SHA-256 hash function or an elliptic curve.

634 **Output.** The setup algorithm returns:

- 635 • setup_R : an input for the relation the proof system is for. An important special case is where
 636 the setup_R is just the empty string, i.e., the relation is independent of any setup.
- 637 • setup_P for the prover and setup_V for the verifier.
- 638 • $auxi$: optional additional auxiliary outputs.

639 If the inputs are malformed or an error occurs, the Setup algorithm may output an error symbol.

640 **Examples of possible setups:**

- 641 • NIZK proof system for 3SAT in the uniform reference string model based on trapdoor per-
 642 mutations
 - 643 – $\text{setup}_R = n$, where n specifies the maximal number of clauses
 - 644 – $\text{setup}_P = \text{setup}_V =$ uniform random string of length $N = \text{size}(n, k)$ for some function
 645 $\text{size}(n, k)$ of n and security parameter k
- 646 • Groth-Sahai proofs for pairing-product equations
 - 647 – $\text{setup}_R =$ description of bilinear group defining the language
 - 648 – $\text{setup}_P = \text{setup}_V =$ common reference string including description of the bilinear group
 649 in setup_R plus additional group elements
- 650 • SNARK for QAP such as e.g. Pinocchio
 - 651 – $\text{setup}_R =$ QAP specification including finite field F and polynomials
 - 652 – $\text{setup}_P = \text{setup}_V =$ common reference string including a bilinear group defined over the
 653 same finite field and some group elements

654 The prover and verifier do not use the same group elements in the common reference
 655 string. For efficiency reasons, one may let setup_P be the subset of the group elements the
 656 prover uses, and setup_V another (much smaller) subset of group elements the verifier uses.

- 657 • Cramer-Shoup hash proof systems
 - 658 – setup_R = specifies finite cyclic group of prime order
 - 659 – setup_P = the cyclic group and some group elements
 - 660 – setup_V = the cyclic group and some discrete logarithms

661 It depends on the concrete setting how Setup runs. In some cases, a trusted third party runs an
 662 algorithm to generate the setup. In other cases, Setup may be a multi-party computation offering
 663 resilience against a subset of corrupt and dishonest parties (and the auxiliary output may represent
 664 side-information the adversarial parties learn from the MPC protocol). Yet, another possibility
 665 is to work in the plain model, where the setup does nothing but copy a security parameter, e.g.,
 666 $\text{setup}_P = \text{setup}_V = k$.

667 There are variations of proof systems, e.g., multi-prover proof systems and commit-and-prove sys-
 668 tems; this document only covers standard systems.

669 **Common reference string:** If the setup information is public and known to everybody, we say
 670 the proof system is in the common reference string model. The setup may for instance specify
 671 $\text{setup}_R = \text{setup}_P = \text{setup}_V$, which we then refer to as a common reference string CRS.

672 **Non-interactive proof systems:** A proof system is non-interactive if the interaction consists of
 673 a single message from the prover to the verifier. After receiving the prover’s message π (called a
 674 proof), the verifier then returns accept or reject.

675 **Public verifiability vs. designated verifier:** If setup_V is public information (e.g. in the CRS model)
 676 known to multiple parties in a non-interactive proof system, then they can all verify a proof p . In
 677 this case, the proof is transferable, the prover only needs to create it once after which it can be
 678 copied and transferred to many verifiers. If on the other hand, setup_V is private we refer to it as a
 679 designated verifier proof system.

680 **Public coin:** In an interactive proof system, we say it is public coin if the verifier’s messages are
 681 uniformly random and independent of the prover’s messages.

682 1.6 Definition and Properties

683 A proof system (Setup, Prove, Verify) for a relation R must be complete and sound. It may have
 684 additional desirable security properties such as being a proof of knowledge or being zero knowledge.

685 1.6.1 Completeness

686 Intuitively, a proof system is complete if an honest prover with a valid witness w for a statement
 687 $x \in L$ can convince an honest verifier that the statement is true. A full specification of a proof
 688 system **must** include a precise definition of completeness that captures this intuition. We give an
 689 example of a definition below for a proof system where the prover initiates.

690 Consider a completeness attacker **Adversary** in the following experiment.

- 691 1. Run **Setup**($params$) \rightarrow ($setup_R, setup_P, setup_V, aux_i$)
- 692 2. Let the adversary choose a worst case instance and witness:
693 **Adversary**($params, setup_R, setup_P, setup_V, aux_i$) \rightarrow (x, w)
- 694 3. Run the interaction between Prove and Verify until the prover returns **error** or the verifier
695 accepts or rejects. Let $result$ be the outcome, with the convention that $result = \mathbf{error}$ if the
696 protocol does not terminate. $\langle \mathbf{Prove}(setup_P, x, w, \mathbf{start}) ; \mathbf{Verify}(setup_V, x) \rangle \rightarrow result$
- 697 • **Adversary** wins if $(setup_R, x, w) \in R$ and result is not **accept**.

We define the adversary's advantage, as a function of the parameters, to be

$$\text{Advantage}(params) = \Pr[\mathbf{Adversarywins}]$$

698 A proof system for R running on $params$ is complete if nobody ever constructs an efficient adversary
699 with significant advantage.

700 It depends on the application what is an efficient adversary (computing equipment, running time,
701 memory consumption, usage lifetime, incentives, etc.) and how large an advantage can be tolerated.
702 Special strong cases include statistical completeness (aka unconditional completeness) where the
703 winning probability is small for any adversary, and perfect completeness, where for any adversary
704 the advantage is exactly 0.

705 1.6.2 Soundness

706 Intuitively, a proof system is sound if a cheating prover has little or no chance of convincing an
707 honest verifier that a false statement is true. A full specification of a proof system must include a
708 precise definition of soundness that captures this intuition. We give an example of a definition below.

709 Consider a soundness attacker **Adversary** in the following experiment.

- 710 1. Run **Setup**($params$) \rightarrow ($setup_R, setup_P, setup_V, aux_i$)
- 711 2. Let the (stateful) adversary choose an instance
712 **Adversary**($params, setup_R, setup_P, setup_V, aux_i$) $\rightarrow x$
- 713 3. Let the adversary interact with the verifier and $result$ be the verifier's output (letting $result =$
714 **reject** if the protocol does not terminate). $\langle \mathbf{Adversary} ; \mathbf{Verify}(setup_V, x) \rangle \rightarrow result$
- 715 • **Adversary** wins if $(setup_R, x) \notin L$ and result is **accept**.

716 We define the adversary's advantage as a function of parameters to be

$$717 \text{Advantage}(params) = \Pr[\mathbf{Adversarywins}]$$

718 A proof system for R running on $params$ is sound if nobody ever constructs an efficient adversary
719 with significant advantage.

720 **FF1.5. Proposed future figure (illustration, diagram, ...):**

721 Illustration of the soundness game

722 It depends on the application what is considered an efficient adversary (computing equipment,
723 running time, memory consumption, usage lifetime, etc.) and how large an advantage can be
724 tolerated. Special strong notions of soundness includes statistical soundness (aka unconditional
725 soundness) where any adversary has small chance of winning, and perfect soundness, where for any
726 adversary the advantage is exactly 0.

727 **1.6.3 Proof of knowledge**

728 Intuitively, a proof system is a proof of knowledge if not only it is sound, but also convinces an
729 honest verifier implies that the prover “knows” a witness. The “knowledge” of a witness, by a
730 prover, can be defined as the ability, in an ideal simulation world, to extract the witness from a
731 successful prover. If a proof system is claimed to be a proof of knowledge, then the security analysis
732 of the ZKP protocol **must** include a precise definition of knowledge soundness that captures this
733 intuition.

734 **Contribution wanted:** Include here a game definition for the extractor required by the formal
735 notion of proof of knowledge. This security property also arises naturally in the ideal/real
736 simulation paradigm, in the context of an *ideal ZKP functionality* that, in the ideal world,
737 receives the witness directly from the prover.

738 **FF1.6. Proposed future figure (illustration, diagram, ...):**

739 Game for a PoK; also, side-by-side, the ideal functionality for a PoK

740 **1.6.4 Zero knowledge**

741 Intuitively, a proof system is zero knowledge if it does not leak any information about the prover’s
742 witness beyond what the attacker may already know about the witness from other sources. Zero
743 knowledge is defined through the specification of an efficient simulator that can generate kosher
744 looking proofs without access to the witness. If a proof system is claimed to be zero knowledge,
745 then the full specification **MUST** include a precise definition of zero knowledge that captures this
746 intuition. We give an example of a definition below.

747 **FF1.7. Proposed future figure (illustration, diagram, ...):**

748 Illustrate the simulator interacting with the verifier

749 A proof system is zero knowledge if the designers provide additional efficient algorithms **SimSetup**,
750 **SimProve** such that realistic attackers have small advantage in the game below. Let **Adversary**
751 be an attacker in the following experiment:

- 752 1. Choose a bit uniformly at random $0,1 \rightarrow b$
- 753 2. If $b = 0$ run **Setup**($params$) \rightarrow ($setup_R, setup_P, setup_V, aux_i$)
- 754 3. Else if $b = 1$ run **SimSetup**($params$) \rightarrow ($setup_R, setup_P, setup_V, aux_i, trapdoor$)
- 755 4. Let the (stateful) adversary choose an instance and witness
- 756 **Adversary**($params, setup_R, setup_P, setup_V, aux_i$) \rightarrow (x, w)
- 757 5. If ($setup_R, x, w$) $\notin R$ return $guess = 0$
- 758 6. If $b = 0$ let the adversary interact with the prover and output a guess (letting $guess = 0$ if
- 759 the protocol does not terminate). $\langle \mathbf{Prove}(setup_P, x, w) ; \mathbf{Adversary} \rangle \rightarrow guess$
- 760 7. Else if $b = 1$ let the adversary interact with a simulated prover and output a guess (letting
- 761 $guess = 0$ if the protocol does not terminate)
- 762 $\langle \mathbf{SimProve}(setup_P, x, trapdoor) ; \mathbf{Adversary} \rangle \rightarrow guess$
- 763 • **Adversary** wins if $guess = b$

We define the adversary's advantage as a function of parameters to be

$$\text{Advantage}(params) = |\Pr[\mathbf{Adversary} \text{ wins}] - 1/2|$$

764 Special strong notions include statistical ZK (aka unconditional ZK) where any adversary has small
765 advantage, and perfect ZK, where for any adversary the advantage is exactly 0.

766 A proof system for R running on $params$ is ZK if there is no efficient adversary with significant advan-
767 tage. The notion of efficient adversary (computing equipment, running time, memory consumption,
768 usage lifetime, etc.), and how large an advantage can be tolerated, depends on the application.

769 **Independence of verifier's knowledge:** The ZK property of a proof does not depend on whether
770 or not the verifier knows something about the witness being proven. However, certain proofs may
771 be designed for the specific case where the verifier also knows the witness (or some valid witness),
772 where a more efficient proof generation (via some interaction), and/or verification may be possible.

773 **Multi-theorem zero knowledge.** In the zero-knowledge definition, the adversary interacts with the
774 prover or simulator on a single instance. It is possible to strengthen the zero-knowledge definition
775 to guard also against an adversary that sees proofs for multiple instances.

776 **Honest verifier zero knowledge.** A weaker privacy notion is honest verifier zero-knowledge, where
777 the adversary follows the protocol honestly (i.e., in steps 6 and 7 in the definition it runs the
778 verification algorithm). It is a common design technique to first construct an HVZK proof system,
779 and then use efficient standard transformations to get a proof system with full zero knowledge.

780 **Witness indistinguishability and witness hiding.** Sometimes a weaker notion of privacy than zero
781 knowledge suffices. Witness-indistinguishable proof systems make it infeasible for an adversary to
782 distinguish which out of several possible witnesses the prover has. Witness-hiding proof systems
783 ensure the interaction with an honest prover does not help the adversary to compute a witness.

784 1.6.5 Advanced security properties

785 The literature describes many advanced security notions a proof system may have. These include
786 security under concurrent composition, nonmalleability against person-in-the-middle attacks, secu-

787 rity against reset attacks in settings where the adversary has physical access, simulation soundness
 788 and simulation extractability to assist sophisticated security proofs, and universal composability.

789 **Universal composability.** The UC framework defines a protocol to be secure if it realizes an ideal
 790 functionality in an arbitrary environment. We can think of an ideal zero-knowledge functionality as
 791 taking an input (x, w) from the prover and if and only if $(x, w) \in R$ it sends the message (x, accept)
 792 to the verifier. The ideal functionality is perfectly sound, since no statement without valid witness
 793 will be accepted, and perfectly zero knowledge, since the proof is just the message `accept`. A proof
 794 system is then UC secure, if the real life execution of the system is ‘security-equivalent’ to the
 795 execution of the ideal proof system functionality. Usually it takes more work to demonstrate a
 796 proof system is UC secure, but on the other hand the framework offers strong security guarantees
 797 when the proof system is composed with other cryptographic protocols.

798 1.6.6 Transferability vs. deniability

799 In the traditional notion of zero-knowledge, a ZKP system prevents the verifier from even being
 800 able to convincingly advertise having interacted in a legitimate proof execution. In other words,
 801 the verifier cannot transfer onto others the confidence gained about the proven statement. This
 802 property is sometimes called *deniability* or *non-transferability*, since a prover that has interacted
 803 as a legitimate prover in a proof is later able to *plausibly deny* having done so, even if the original
 804 verifier releases the transcript publicly.

805 Despite *deniability* being often a desired property, its opposite — *transferability*, or public veri-
 806 fiability — can also be considered a feature. The verifier in a legitimate execution of a publicly
 807 verifiable ZKP becomes able to convince an external party that the corresponding proven statement
 808 is true (or, in the case of a statement of knowledge, that some prover did indeed have the claimed
 809 knowledge). Transferability can in some cases be accomplished by simply sharing the transcript
 810 (the verifier’s view) of the interaction (messages exchanged and the internal state of the verifier).

811 **Recommendation 1.1 — Characterize deniability vs. transferability.** For a proper security
 812 analysis of an application making use of a ZKP, it is important to characterize which of deniability
 813 of transferability (or a nuanced version of them) the ZKP is required to have. This is relevant to
 814 assess security/privacy upon composition with other applications.

815 1.6.7 Examples of setup and trust

816 The security definitions assume a trusted setup. There are several variations of what the setup
 817 looks like and the level of trust placed in it.

- 818 • **No setup or trustless setup.** When no trust is required, for example if the setup is just a
 819 copy of a security parameter k , or because everyone can directly verify the setup is correct.
- 820 • **Uniform random string.** all parties have access to a uniform random string $\text{URS} = \text{setup}_R =$
 821 $\text{setup}_P = \text{setup}_V$. We can distinguish between the lighter trust case where the parties just need
 822 to get a uniformly sampled string, which they may for instance get from a trusted common
 823 source of randomness e.g. sunspot activity, and the stronger trust case where zero-knowledge
 824 relies on the ability to simulate the URS generation together with a simulation trapdoor.

- 825 • **Common reference string.** The URS model is a special case of the CRS model. But in
826 the CRS model it is also possible that the common setup is sampled with a non-uniform
827 distribution, which may exclude easy access to a trusted common source. A distinction can
828 be made whether the CRS has a verifiable structure, i.e., it is easy to verify it is well-formed,
829 or whether full trust is required.
- 830 • **Designated verifier setup.** If the setup generates correlated setup_P and setup_V , where
831 setup_V is intended only for a designated verifier, then trust is needed about the setup algo-
832 rithm. This is for instance the case in Cramer-Shoup public-key encryption where a designated
833 verifier NIZK proof is used to provide security under chosen-ciphertext attack. Here the setup
834 is generated as part of the key generation process, and the recipient can be trusted to do this
835 honestly because it is the recipient's own interest to make the encryption scheme secure.
- 836 • **Random oracle model.** The common setup describes a cryptographic hash function, e.g.,
837 SHA256. In the random oracle model, the hash function is heuristically assumed to act like
838 a random oracle that returns a random value whenever it is queried on an input not seen
839 before. There are theoretical examples where the random oracle model fails, exploiting the
840 fact that in real life the hash function is a deterministic function, but in practice the heuristic
841 gives good efficiency and currently no weaknesses are known for 'natural' proof systems.
- 842 • **Threshold secure and/or subversion resistant.** There are several proposals to reduce
843 the trust in the setup such as using secure multi-party computation to generate a CRS, using
844 a multi-string model where there are many CRSs and security only relies on a majority being
845 honestly generated, and subversion resistant CRS where zero-knowledge holds even against a
846 maliciously generated CRS.

847 1.7 Assumptions

848 A full specification of a proof system **must** state the assumptions under which it satisfies the
849 security definitions and demonstrate the assumptions imply the proof system has the claimed
850 security properties.

851 A security analysis may take the form of a mathematical proof by reduction, which demonstrates
852 that a realistic adversary gaining significant advantage against a security property, would make it
853 possible to construct a realistic adversary gaining significant advantage against one of the under-
854 pinning assumptions.

855 To give an example, suppose soundness relies on a collision-resistant hash function. The demon-
856 stration of this fact may take the form of describing a simple and efficient algorithm **Reduction**,
857 which may call a soundness attacker **Adversary** as a subroutine a few times. Furthermore, the
858 demonstration may establish that the advantage **Reduction** has in finding a collision is closely
859 related to the advantage an arbitrary **Adversary** has against soundness, for instance

$$860 \text{Advantage_soundness}(params) \leq 8 \times \text{Advantage_collision}(params)$$

861 Suppose the proof system is designed such that we can instantiate it with the SHA-256 hash
862 function as part of the parameters. If we assume the risk of an attacker with a budget of \$1,000,000
863 finding a SHA-256 collision within 5 years is less than 2^{-128} , then the reduction shows the risk of
864 an adversary with similar power breaking soundness is less than 2^{-125} .

865 **Cryptographic assumptions.** Cryptographic assumptions, i.e. intractability assumptions, specify
 866 what the proof system designers believe a realistic attacker is incapable of computing. Sometimes a
 867 security property may rely on no cryptographic assumptions at all, in which case we say security of
 868 unconditional, i.e., we may for instance say a proof system has unconditional soundness or uncon-
 869 ditional zero knowledge. Usually, either soundness or zero knowledge is based on an intractability
 870 assumption though. The choice of assumption depends on the risk appetite of the designers and
 871 the type of adversary they want to defend against.

872 **Plausibility.** An intractability assumption that has been broken shall not be used. We recom-
 873 mend designing flexible and modular proof systems such that they can be easily updated if an
 874 underpinning cryptographic assumption is shown to be false.

875 Sometimes, but not always, it is possible to establish an order of plausibility of assumptions. It is
 876 for instance known that if you can break the discrete logarithm problem in a particular group, then
 877 you can also break the computational Diffie-Hellman problem in the same group, but not necessarily
 878 the other way around. This means the discrete logarithm assumption is more plausible than the
 879 computational Diffie-Hellman assumption and therefore preferable from a security perspective.

880 **Post-quantum resistance.** There is a chance that quantum computers will be developed within
 881 a few decades. Quantum computers are able to efficiently break some cryptographic assumptions,
 882 e.g., the discrete logarithm problem. If the expected lifetime of the proof system extends beyond
 883 the emergence of quantum computers, then it is necessary to rely on intractability assumptions
 884 that are believed to resist quantum computers. Different security properties may require different
 885 lifetimes. For instance, it may be that proofs are verified immediately and hence post-quantum
 886 soundness is not required, while at the same time an attacker may collect and store proof transcripts
 887 and later try to learn something from them, so post-quantum zero knowledge is required.

888 **Concrete parameters.** It is common in the cryptographic literature to use vague phrasing such as
 889 “the advantage of a polynomial time adversary is negligible” when describing the theory behind a
 890 proof system. However, concrete and precise security is needed for real-world deployment. A proof
 891 system should therefore come with concrete parameter recommendation and a statement about the
 892 level of security they are believed to provide.

893 **System assumptions.** Besides cryptographic assumptions, a proof system may rely on assumptions
 894 about the equipment or environment it works in. As an example, if the proof system relies on a
 895 trusted setup it should be clearly stated what kind of trust is placed in.

896 **Setup.** If the prover or verifier are probabilistic, they require an entropy source to generate
 897 randomness. Faulty pseudorandomness generation has caused vulnerabilities in other types of
 898 cryptographic systems, so a full specification of a proof system should make explicit any assumptions
 899 it makes about the nature or quality of its source of entropy.

900 1.8 Efficiency

901 A specification of a proof system may include claims about efficiency and if it does the units of
 902 measurement MUST be clearly stated. Relevant metrics may include:

- 903 • **Round complexity:** Number of transmissions between prover and verifier. Usually mea-
 904 sured in the number of moves, where a move is a message from one party to the other. An

905 important special case is that of 1-move proof systems, aka non-interactive proof systems,
 906 where the verifier receives a proof from the prover and directly decides whether to accept or
 907 not. Non-interactive proofs may be transferable, i.e., they can be copied, forwarded and used
 908 to convince several verifiers.

- 909 • **Communication:** Total number of bits transmitted between the prover and verifier.
- 910 • **Prover computation:** Computational effort the prover expends over the duration of the
 911 protocol. Sometimes measured as a count of the dominant cryptographic operations (to avoid
 912 system dependence) and sometimes measured in seconds on a particular system (when making
 913 concrete measurements).
- 914 • Depending on the intended usage, many other metrics may be important: memory consump-
 915 tion, energy consumption, entropy consumption, potential for parallelisation to reduce time,
 916 and offline/online computation trade-offs.
- 917 • **Verifier computation:** Computational effort of the verifier over the duration of the protocol.
- 918 • **Setup cost:** Size of setup parameters, e.g. a common reference string, and computational
 919 cost of creating the setup.

920 Readers of a proof system specification may differ in the granularity they need in the efficiency
 921 measurements. Take as an example a proof system consisting of an information theoretic core that
 922 is then compiled with cryptographic primitives to yield the full system. An implementer will likely
 923 want to have a detailed performance analysis of the information theoretic core as well as the cryp-
 924 tographic compilation, since this will guide her choice of trade-offs and optimizations. A consumer
 925 on the other hand will likely want to have a high-level performance analysis and an apples-to-apples
 926 comparison to competing proof systems. We therefore recommend to provide both a detailed anal-
 927 ysis that quantifies all the dominant efficiency costs, and a bottom-line analysis that summarizes
 928 performance for reasonable choices of parameters and identifies the optimal performance region.

929 1.8.1 Characterization of security properties

930 The benchmarking of a technique should clarify the distinct security levels achieved/conjectured
 931 for different security properties, e.g., soundness vs. zero-knowledge. In each case, the security
 932 type should also be clarified with respect to being unconditional, statistical or computational.
 933 When considering computational security, it should be clarified to what extent pre-computations
 934 may affect the security level, and whether/how known attacks may be parallelizable. All security
 935 claims/assertions should be qualified clearly with respect to whether they are based on proven
 936 security reductions or on heuristic conjectures. In either case the security analysis should make
 937 clear which computational assumptions and implementation requirements are needed. It should be
 938 made explicit whether (and how) the security levels relate to classical or quantum adversaries. When
 939 applicable, the benchmarking should characterize the security (including possible unsuitability) of
 940 the technique against quantum adversaries.

941 1.8.2 Computational security levels for benchmarking

942 The benchmarks for each technique **shall** include at least one parametrization achieving a con-
 943 jectured computational security level κ approximately equal to, or greater than, 128 bits. Each
 944 technique **should** also be benchmarked for at least one additional higher computational security

945 level, such as 192 or 256 bits. (If only one, the latter is preferred.) The benchmarking at more
 946 than one level aids the understanding of how the efficiency varies with the security level. The
 947 interest in a security level as high as 256 bits can be considered a cautious (and heuristic) safety
 948 margin, compared for example with intended 128 bits. This is intended to handle the possibility
 949 that the conjectured level of security is later found to have been over-estimated. The evaluation
 950 at computational security below 128 bits may be justified for the purpose of clarifying how the
 951 execution complexity or time varies with the security parameter, but should not be construed as a
 952 recommendation for practical security.

953 **An exception allowing a lower computational security parameter.** With utmost care, a compu-
 954 tational security level may be justified below 128 bits. The following text describes an exception.
 955 In some interactive ZKPs (see Section 2.6), there may be cryptographic properties that only need
 956 to be held during a portion of a protocol execution, which in turn may be required to take less
 957 than a fixed amount of time, say, one minute. For example, a commitment scheme used to en-
 958 able temporary hiding during a coin-flipping protocol may only need to hold until the other party
 959 reveals a secret value. In such case the property may be implemented with less than 128 bits of
 960 security, with special care (namely with respect to composition in a concurrent setting) and if the
 961 difference in efficiency is substantial. Such decreased security level of a component of a protocol
 962 may also be useful for example to enable properties of deniability (non-transferability). Depending
 963 on the application, other exceptions may be acceptable, upon careful analysis, when the witness
 964 whose knowledge is being proven is itself discoverable from the ZK instance with less computational
 965 resources than those corresponding to 128 bits of security.

966 1.8.3 Statistical security levels for benchmarking

967 The soundness security of certain interactive ZKP systems may be based on the ability of the
 968 verifier(s) to validate-or-trust the freshness and entropy of a challenge (e.g., a nonce produced by
 969 a verifier, or randomness obtained by a trusted randomness Beacon). In some of those cases, a
 970 statistical security parameter σ (e.g., 40 or 64 bits) may be used to refer to the error probability
 971 (e.g., 2^{-40} or 2^{-64} , respectively) of a protocol with “one-shot” security, i.e., when the ability of
 972 a malicious prover to succeed without knowledge of a valid witness requires guessing in advance
 973 what the challenge would be. A lower statistical security parameter may be suitable if there is a
 974 mechanism capable to detect and prevent a repetition of failed proof attempts.

975 While an appropriate minimal parameter may depend on the application scenario, benchmarking
 976 **shall** be done with at least one parametrization achieving a conjectured statistical security level
 977 of at least 64 bits. Whenever the efficiency variation is substantial across variations of statistical
 978 security parameter, it is recommended that more than one security level be benchmarked. The
 979 cases of 40, 64, 80 and 128 bits are suggested.

980 Many interactive protocols can be compiled into a non-interactive (NI) version, for example us-
 981 ing a Fiat-Shamir transformation that non-interactively computes the challenge of an otherwise
 982 interactive commit-challenge-response protocol. In the resulting NI version, the prover is the sole
 983 generator of the proof, and so can retry many attempts until obtaining a challenge for which it
 984 can produce a response. If the computational cost of the original protocol is $\tilde{O}(\sigma)$, i.e., not higher
 985 than linear in the statistical security parameter, then the benchmark **should** include at least one
 986 statistical parameter that enables retaining the intended computational security (at least 128 bits)
 987 when the protocol is transformed into a NI version. Computational security remains if the expected
 988 number of needed attempts is of the order of 2^κ .

Chapter 2. Paradigms

2.1 Background

There are many types of zero-knowledge proof (ZKP) systems in the literature, offering trade-offs between communication cost, computational cost, and underlying cryptographic assumptions. Many of these proofs can be decomposed into an **information-theoretic (IT) proof system**, and a **cryptographic compiler (CC)** that compiles such an IT-proof system into a real ZKP.

The properties of the IT proof system are considered in a model taken for granted, not needing justification. However, to achieve a proof that is useful in the real world, the IT proof system then needs to be instantiated by a CC that removes the ideal components. An actual proof implementation in the real world also requires agreement on an **arithmetization** procedure that represents the *statement*, the *instance* and the *witness* in some concrete way.

The IT/CC separation can simplify the protocol design, allowing each part to be analyzed, optimized, and implemented separately. This also facilitates a systematic exploration of the design space, when in search of the best solution for a task at hand. For simplicity, the term “ZK” is sometimes omitted, but the implicit default focus remains on ZKP systems. The IT/CC perspective (considered in Sections 2.2 and 2.3) is essentially focused on general proof systems for NP relations. Other types and aspects of ZKPs exist, explained outside the IT/CC categorization, including the case of some specialized languages (Section 2.4), and aspects of recursive proof composition (Section 2.5) and interactivity (Section 2.6).

FF2.1. Proposed future figure (illustration, diagram, ...):

Diagram representing the three main concepts: IT, CC, Arithmetization

2.1.1 Information-theoretic (IT) proof system

An IT proof system provides soundness and ZK guarantees even when the prover and the verifier are computationally unbounded. However, they use an idealized model of interaction, where the prover and/or verifier’s powers are restricted, structured or augmented in some ideal way.

A common type of idealized model provides access to some black-box communication functionality, which will be concretely realized by the cryptographic compiler. For instance, the idealized model may let the prover produce a long proof vector, and then only give the verifier restricted access to this vector, such as querying just a few elements from the vector (without seeing the rest, but also without letting the prover change the answers depending on the queries; that is, the whole vector is committed to in advance). Another common example is an *ideal commitment*, where a party can commit to a value by sending it to a trusted oracle (while keeping it hidden), and can later allow the other party to retrieve the value (with assurance that the value did not change).

FF2.2. Proposed future figure (illustration, diagram, ...):

Illustration of ideal commitment

The idealized model can also place restrictions on the parties. For example, a restricted prover may consist of multiple separate parties that can coordinate in advance but cannot communicate with each other during the protocol execution. This allows the verifier to challenge them separately, and compare their answers for consistency. The cryptographic compiler, in such cases, shows how to transform the idealized parties into real-world parties that are not subject to artificial restrictions (but may be computationally bounded).

2.1.2 Cryptographic compiler (CC)

A cryptographic compiler transforms an IT proof system into a real-world protocol that involves direct interaction between the prover and verifier. The aforementioned idealized assumptions on interaction models and parties' powers are eliminated, often by *enforcing* them so they no longer need to be axiomatically assumed. This is done using protocols that, typically, rely on simpler cryptographic building blocks. The primitives used by a cryptographic compiler are then realized by (possibly heuristic) concrete instantiations, such as hash functions or pairing-friendly elliptic curves. This yields a concrete scheme that can be implemented and used in real-world applications.

The underlying building blocks may be cryptographic primitives (e.g., collision-resistant hash functions or homomorphic commitment schemes). In this case, the security of the combination of the IT proof system with the compiler depends on the cryptographic assumptions in the realizations of these primitives, and in particular, typically applies only to computationally-bounded parties.

Alternatively, the compiler may eliminate the high-level ideal models assumed by the IT proof system, but introduce new idealized primitives meant to capture finer-grained local assumptions (e.g., a random oracle or a generic bilinear group). In this case its security would hold under some ideal-model assumptions (e.g., bounding the number of queries made to the random oracle), and the real-world realization would either use a further compiler (e.g., to achieve the requisite properties of the finer-grained model under well-defined assumptions), or be heuristic (i.e., intuitively plausible but not formally proven). These, too, are typically plausible only for computationally-bounded parties. The cryptographic compiler may also provide extra desirable properties, such as eliminating interaction, shrinking the size of some of the messages, or even adding a ZK feature to an IT proof system that does not have it.

2.1.3 Arithmetization

Every IT proof systems has some *native representation* in which it accepts the statement to be proven. Common ones include polynomial equations, Boolean circuits and arithmetic circuits. Conversely, applications often prefer higher-level succinct representations, such as finite automata, CPU+memory models, or circuits augmented with lookup tables and repeated elements.

The conversion of a high-level representation to the native representation may be wholly handled by a *frontend*, which is separate from the proof scheme per se. However, many proof systems, as presented and implemented, already include some internal conversion from some natural format to their lower-level native (e.g., polynomial equations). This conversion is referred to as *arithmetization* and consists of the following procedures:

- 1062 • *statement reduction*: generates the native statement representation (e.g., the polynomials to
1063 be checked);
- 1064 • *instance reduction*: generates the native instance representation (e.g., assignments to instance
1065 variables on which the polynomials are evaluated); and
- 1066 • *witness reduction*: generates the native witness representation (e.g., assignments to witness
1067 variable).

1068 2.1.4 Complexity and features

1069 The concrete ZKP schemes that emerge from combining an IT proof system and a crypto compiler
1070 — as well as any other specialized ZKP — are possible when following these proof systems can vary
1071 in characteristics. Important complexity measures for these proofs include: proof length, round
1072 complexity, query complexity (number of queries and size of each query), field (alphabet) size,
1073 complexity of the verifier’s decision predicate (e.g., when expressed as an arithmetic circuit).

1074 **FF2.3. Proposed future figure (illustration, diagram, ...):**

1075 Illustration that comprises the various metrics of interest

1076 **Proof succinctness.** With respect to proof length, there are several categories of interest:

- 1077 (a) Fully succinct: independent of statement size, i.e., $O(1)$ crypto elements.
- 1078 (b) Polylog succinct: number of crypto elements is polylogarithmic in the circuit size.
- 1079 (c) Sqrt succinct: Proportional to square root of circuit size.
- 1080 (d) Depth-succinct: depends on depth of a verification circuit representing the statement.
- 1081 (e) Non succinct: Proof length is not sublinear in the circuit size.

1082 The term SNARK denotes a succinct non-interactive argument of knowledge [BCCGLRT17].

1083 **Complexity reference.** Complexity measures (such as proof succinctness) should be properly
1084 contextualized. For example, it does matter whether a complexity measure takes into account only
1085 the public information (e.g., statement and instance) or whether it also depends on the private part
1086 (e.g., witness). It also matters whether the expressed complexity (e.g., linear, quadratic, square-
1087 root, etc.) of a proof length is given with respect to the statement size refers to the statement
1088 before of after arithmetization. Different references are used across different works.

1089 **Other features.** Further characterization aspects of interest include:

- 1090 1. **Setup trust.** All non-interactive zero-knowledge proofs require a set of global public param-
1091 eters agreed on by all parties. If the global public parameters can be chosen *randomly*, e.g.,
1092 by hashing a famous quote, then we say that the proving system has a public coin setup, or
1093 a trustless setup. This is the most ideal setup when it is possible.

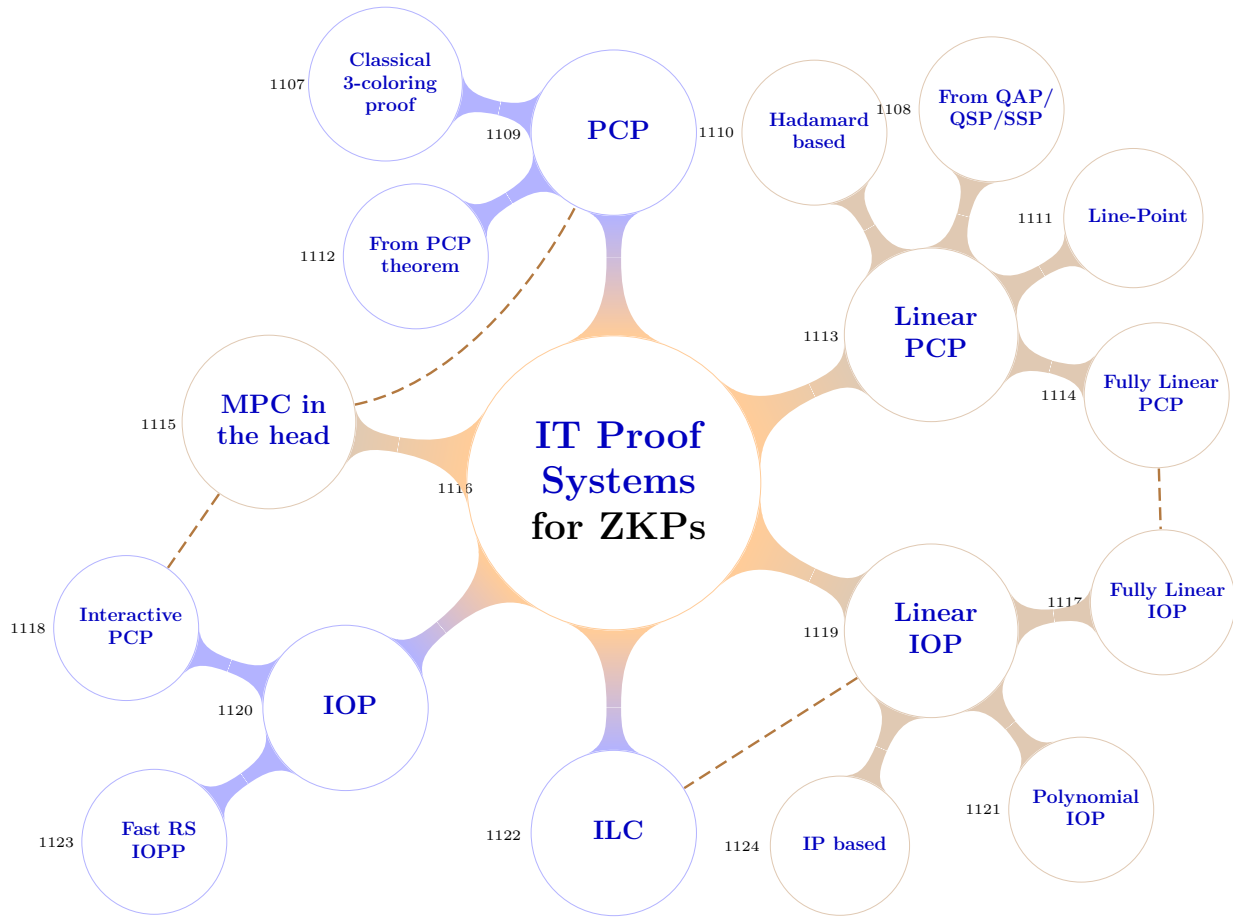
1094 If the global public parameters cannot be chosen randomly and are instead chosen either via
1095 a trusted party or using a multiparty computation, then we say that the proving system has a
1096 private coin setup, or a trusted setup. In this case either the trusted party or a coalition of all

1097 the parties involved in the multiparty computation are able to forge proofs. Trusted setups can
 1098 either be universal or language dependent. A universal setup allows the same setup to be used
 1099 for any language and therefore provides greater flexibility than a language dependent setup.

- 1100 2. **Types of primitives** (e.g., Symmetric-key cryptography (e.g., blockciphers), collision-resist-
 1101 ant hash functions, random oracles, public-key crypto)

1102 2.2 Information-Theoretic (IT) Proof Systems

1103 This section surveys various categories of **IT proof systems**, designed in an idealized model. They
 1104 enable proofs that do not rely on cryptographic assumptions, often resulting in elegant and easier
 1105 to understand constructions. Even though often these proofs cannot be used directly in the real
 1106 world, they are still useful for combination with cryptographic compilers.



1125 **Legend:** **Fast RS IOPP** (aka FRI: Fast Reed-Solomon IOP of Proximity); **ILC** (Ideal Linear Com-
 1126 mitment); **IOP** (Interactive Oracle Proof); **IP** (Interactive Proof); **IT** (Information Theoretic); **MPC**
 1127 ([Secure] Multi-Party Computation); **PCP** (Probabilistic Checkable Proof); **QAP** (Quadratic Arithmetic
 1128 Program); **QSP** (Quadratic Span Program); **SSP** (Square Span Program); **ZKP** (Zero-Knowledge Proof).

1129 **Figure 2.1.** Various IT proof systems

1130 **Notation.** A verifier is said to make “point queries” to the proof Π if the verifier has access to a
 1131 proof oracle O^Π that takes as input an index i and outputs the i -th symbol $\Pi(i)$ of the proof. A
 1132 verifier is said to make “inner-product queries” to the proof $\Pi \in \mathbb{F}^m$ (for some finite field \mathbb{F}) if the
 1133 proof oracle takes as input a vector $q \in \mathbb{F}^m$ and returns the value $\langle \Pi, q \rangle \in \mathbb{F}$. A verifier is said
 1134 to make “matrix-vector queries” to the proof $\Pi \in \mathbb{F}^{m \times k}$ if the proof oracle takes as input a vector
 1135 $q \in \mathbb{F}^k$ and returns the matrix-vector product $(\Pi \cdot q) \in \mathbb{F}^m$.

1136 2.2.1 Summary

1137 In a classical proof of a statement x , the prover sends a proof string π for complete reading by the
 1138 verifier, who then *accepts* if and only if the proof conveys an efficient absolute-verifiability of the
 1139 statement’s truth. For example, a trivial (not ZK) example of *classical* proof of knowledge is the
 1140 direct sending of the known witness w .

1141 The field of proof checking has meanwhile evolved to encompass interactive proofs (where the con-
 1142 vincing comes from an interaction, rather than from the analysis of a sole string) and arguments
 1143 (where certainty is not absolute, but still overwhelming), and often with a ZK feature.

1144 This section explains IT proof systems that often make use of ideal (unrealizable) components and
 1145 are not necessarily ZK, but which allow for a later compilation of real proofs that are both succinct
 1146 and ZK. Figure 2.1 considers the following main types of IT proof systems:

- 1147 a. **Probabilistically Checkable Proof (PCP):** In a PCP proof, the prover sends the verifier
 1148 a (possibly very long) proof string π ; the verifier makes “point queries” to the proof, reads
 1149 the entire statement x , and accepts or rejects.
- 1150 b. **Linear PCP:** In a linear PCP proof, the prover sends the verifier a (possibly very long)
 1151 proof string π , lying in a vector space \mathbb{F}^m . The verifier makes a number of linear queries to
 1152 the proof, reads the entire statement x , and accepts or rejects.
- 1153 c. **MPC-in-the-head:** A proof can be composed of the partial views of a simulated security
 1154 multi-party computation (MPC) execution with semi-honest security. The witness is secret-
 1155 shared between the simulated parties, who then (by simulation) securely compute the ZKP
 1156 predicate. The partial transcript simultaneously ensures soundness and ZK.
- 1157 d. **Interactive Oracle Proof (IOP):** An IOP is a generalization of a PCP, to consider the
 1158 interactive setting. In each round of communication, the verifier sends a challenge string c_i to
 1159 the prover, and then the prover responds with a PCP proof π_i , which the verifier may query
 1160 via point queries. After several rounds of interactions, the verifier accepts or rejects.
- 1161 e. **Linear IOP:** A linear IOP is a generalization of a linear PCP to the interactive setting. (See
 1162 IOP above.) Here the prover sends in each round a proof vector π_i that the verifier may query
 1163 via linear (inner-product) queries.
- 1164 f. **Ideal Linear Commitment (ILC):** The ILC model is similar to linear IOP, except that in
 1165 each round the prover sends a proof matrix rather than a proof vector; the verifier learns the
 1166 product of the proof matrix and the query vector. Each ILC proof matrix may be the output
 1167 of an arbitrary function of the input and the verifier’s messages, not limited to linear functions.

FF2.4. Proposed future figure (illustration, diagram, ...):

One table of figures, pictorially comparing the various IT proof systems

2.2.2 Probabilistically Checkable Proof (PCP)

A probabilistically checkable proof is a proving paradigm that assumes an interaction between the prover and the verifier. The prover sends a first message and can only convince the verifier in later messages if there exists a valid witness. In this section we first present a classic PCP example for proving the 3-coloring problem. Then we provide a more precise explanation of the PCP information theoretic paradigm and the PCP theorem.

2.2.2.1 Classical 3-coloring proof

The graph 3-coloring problem was the basis for the first ZKP system for NP [GMW91]. The *instance* of the problem is an undirected graph $G = (V, E)$. The *statement* (i.e., the goal of what to prove) is that the graph has a 3-coloring, i.e., that each node $i \in V$ (in the set V of vertices) can be assigned one of three colors $c(i) \in \{1, 2, 3\}$, such that any edge $(i, j) \in E$ (i.e., any pair of connected nodes) connects two vertices with distinct colors. The *witness* is a coloring assignment $c : V \rightarrow \{1, 2, 3\}$, mapping each node to one of three colors.

Let $R_{3\text{COL}}$ denote the *relation* of interest, used to test whether c is a good 3-coloring for a given graph G , i.e., satisfying $(G, c) \in R_{3\text{COL}} \Leftrightarrow \#(c) = 3 \wedge c(u) \neq c(v)$ for every $(u, v) \in E$. Since the relation $R_{3\text{COL}}$ is *NP-complete*, having a ZKP for proving that a graph is 3-colorable implies having a ZKP for proving membership in any other NP language (say, associated with a relation R). This is done by having the prover and the verifier locally apply a polynomial-time reduction to convert their inputs for R into inputs for $R_{3\text{COL}}$.

The protocol. The classical ZKP for $R_{3\text{COL}}$ proceeds as follows. (Here we consider an IT system that provides an ideal commitment functionality.)

- The prover, on input (G, c) , picks a random permutation $\phi : \{1, 2, 3\} \rightarrow \{1, 2, 3\}$ defining a random shuffle of the colors, and lets $c'(v) = c(\phi(v))$ be the shuffled coloring.
- For each node $v \in V$, the prover commits to the shuffled color $c'(v)$. (In the real world the prover sends (or interacts, in order to provide) a commitment to the verifier.)
- The verifier asks the prover to open the colors of a uniformly selected edge $(u, v) \in E$.
- The prover opens the commitments, revealing $c'(u)$ and $c'(v)$ to the verifier.
- The verifier accepts G (as a 3-colorable graph) iff the prover successfully opens the two commitments and they contain different values in $\{1, 2, 3\}$.

FF2.5. Proposed future figure (illustration, diagram, ...):

Illustrate the classical R3Col ZKP protocol

Security analysis. The above protocol satisfies the three needed properties:

- 1202 • **Completeness:** if the coloring c is valid then so is its permuted version c' , and so an honest
1203 prover will always successfully open two distinct colors in c' .
- 1204 • **ZK:** only the two shuffled colors $(c'(u), c'(v))$ are revealed, but they could be *simulated* (see
1205 Section 1.6.4) by choosing any random pair of *distinct* colors. This holds even for a malicious
1206 verifier that chooses u and v arbitrarily, as long as the prover checks they form a valid edge.
- 1207 • **Soundness:** if G does not admit a valid coloring, then no matter which colors c^* are commit-
1208 ted to by a malicious prover, there is at least one edge $(u, v) \in E$ that will satisfy $c^*(u) = c^*(v)$.
1209 Therefore, the verifier will reject the proof with at least $1/|E|$ probability. This poor level
1210 of soundness can be amplified, while preserving zero knowledge, by sequentially running the
1211 protocol many times independently (e.g., $\kappa \cdot |E|$ for an error probability or about 2.7^{-k}). Note
1212 that a parallel repetition would not preserve ZK.

1213 2.2.2.2 From PCP theorem

1214 In general, in a PCP for an NP-relation R , the prover computes a probabilistic polynomial-time
1215 mapping from a statement-witness pair (x, w) to a proof string $\pi \in \Sigma^m$ (i.e., a string of length m
1216 over some finite alphabet Σ). The verifier queries a randomly chosen subset of the symbols of π ,
1217 and then decides whether to accept or reject. Crucially, the content of π is produced by the prover
1218 without knowing which indices into π will be queried by the verifier. More precisely, the verifier's
1219 algorithm is specified by a randomized mapping from an instance x to a subset $Q \subset [m]$ of queried
1220 symbols, and a decision predicate $D(x, Q, \pi_Q)$ deciding whether to accept or reject based on the
1221 contents of the queried symbols. The PCP theorem [Mic00; Kil92] roughly states the following.

1222 **Theorem 2.2.1 — The PCP Theorem.** For all languages in NP there exists an efficient PCP
1223 with logarithmic communication complexity.

1224 A *zero-knowledge PCP* (zk-PCP) moreover fulfills the zero-knowledge property (see Section 1.6.4),
1225 where the zero-knowledge simulator is required by default to *perfectly* sample the view (Q, π_Q) of
1226 an *honest* verifier given x alone. We assume that one can efficiently recognize whether a set Q can
1227 be generated by an honest verifier on a given input x . In the literature non-zk PCPs are often
1228 required to have small query complexity but see that the three coloring example is zero-knowledge
1229 and not small query complexity.

1230 **Example: 3-coloring as a PCP.** In the 3-coloring example of section 2.2.2.1 above, π is a string of
1231 length $m = |V|$ over $\Sigma = \{1, 2, 3\}$ comprised of the values $c'(v)$ for $v \in V$, where the randomness
1232 is over the permutation ϕ used to convert c into c' . Here, Q is uniformly distributed over sets of
1233 size 2 corresponding to edges in E , and D accepts if the two queried symbols are distinct. The
1234 basic version of the zk-PCP for $R_{3\text{COL}}$ has a high soundness error of $1 - 1/|E|$. Soundness can
1235 be amplified, while respecting (honest verifier) zero knowledge, by concatenating independently
1236 generated proofs $\pi^{(i)}$ and querying them independently. Note that making multiple queries to the
1237 *same* π would compromise zero knowledge.

1238 **Discussion.** The above notion of zk-PCP is information-theoretic in the sense that both soundness
1239 and (honest-verifier) zero knowledge hold with respect to computationally unbounded parties. The
1240 reason it cannot be *directly* used as a zero-knowledge proof protocol, without additional crypto-
1241 graphic machinery, is that it is not clear how to only allow the verifier a restricted access to π ,
1242 which is necessary for zero knowledge, without allowing a malicious prover to violate the indepen-

1243 dence assumption between π and Q on which soundness relies. Put differently, if Q is sent to the
 1244 prover first, then a malicious prover can violate soundness, and if π is sent to the verifier in its
 1245 entirety, then zero knowledge is compromised even when the verifier is honest. The role of a cryp-
 1246 tographic compiler is to convert an arbitrary zk-PCP (possibly with some syntactic restrictions)
 1247 into a zero-knowledge proof protocol.

1248 2.2.3 Linear PCP

1249 The PCP model and the associated PCP theorem are surprisingly powerful, but entail a high cost:
 1250 the prover needs to explicitly write out a long proof, most of which will not be read. This cost
 1251 can be reduced by changing the idealized model to allow a different and more expressive kinds of
 1252 queries. Intuitively, the prover will produce a small proof, which can be queried via a much larger
 1253 query space, so the query procedure undertakes some of the burden.

1254 **FF2.6. Proposed future figure (illustration, diagram, ...):**

1255 Illustrate a generic linear PCP

1256 A simple and useful instance of this is the *linear PCP* (LPCP) model. Whereas in a standard PCP
 1257 the verifier is allowed to make a bounded number of *point queries*, which read individual symbols of
 1258 the proof π , a *linear PCP* allows each query to take an arbitrary linear combination of the entries
 1259 of π . More concretely, in a linear PCP over a finite field \mathbb{F} the proof is a vector $\pi \in \mathbb{F}^m$ and
 1260 each query $q_i \in \mathbb{F}^m$ returns the inner product $\langle \pi, q_i \rangle$. We require by default that the queries q_i be
 1261 *input-oblivious*, in the sense that they can be picked independently of x . One can construct such
 1262 an LPCP for any NP-relation R with polynomial proof size m , soundness error $O(1/|\mathbb{F}|)$, and a
 1263 constant number of queries. The main price we pay for the more powerful PCP model is that the
 1264 corresponding cryptographic compilers typically need to rely on stronger primitives that live in the
 1265 “public-key cryptography” world, and moreover require strong forms of setup to fully respect the
 1266 efficiency features of the underlying LPCP.

1267 LPCPs were introduced as a way to achieve simple low-communication proof systems [IKO07].
 1268 An initial motivation was to replace complex PCP machinery by simpler one, at the cost of using
 1269 stronger cryptography.

1270 However, it was already observed back then that beyond simplicity, this new approach can lead
 1271 to an efficiency feature that was not possible via the traditional PCP-based approach: prover-to-
 1272 verifier communication that includes only a constant number of “ciphertexts,” or group elements,
 1273 independently of the complexity of R .

1274 Below we describe two types of linear PCP: Hadamard-based and Quadratic Arithmetic Programs
 1275 (QAP). These are the most commonly used linear PCPs in the current literature. Other types of
 1276 linear PCP exist which are not discussed in this document, including Quadratic Span Programs
 1277 (QSP), Square Span Programs (SSP), and Line Point based.

1278 2.2.3.1 Hadamard based

1279 To present the Hadamard-based LPCP, it is convenient to consider the NP-relation as an arithmetic
 1280 circuit over a finite field \mathbb{F} , whose gates perform field additions and multiplications. Concretely,

1281 the prover wants to convince the verifier that there is a witness $w \in \mathbb{F}^m$ such that $C(x, w) = 0$,
 1282 where C is a publicly known arithmetic circuit and $x \in \mathbb{F}^n$ is the public statement.

1283 C and x are first converted into a system of s *quadratic* equations of the form $Q_i(Y) = 0$, where
 1284 each Q_i is a multivariate polynomial of degree (at most) 2 in s variables Y_1, \dots, Y_s , such that there
 1285 is a $w \in \mathbb{F}^m$ satisfying $C(x, w) = 0$ if and only if there is a $y \in \mathbb{F}^s$ satisfying $Q_i(y) = 0$ for all i .

1286 In a Hadamard LPCP for proving the satisfiability of $Q_i(Y)$, the prover, on input (x, w) , first
 1287 computes the values $y \in \mathbb{F}^s$ of all s gates of C on input (x, w) . Then it computes a quadratic sized
 1288 vector $\hat{y} \in \mathbb{F}^{s^2}$ of entries $\hat{y}_{jk} = y_j y_k$, and outputs the proof $\pi = (y, \hat{y})$. Note that every degree-2
 1289 polynomial in y can be obtained by the verifier by making a single linear query to π .

1290 The verifier’s queries achieve two goals: (1) check consistency of the two parts of π with the
 1291 tensoring relation; (2) check satisfiability of all Q_i relations. Goal (1) is realized by comparing two
 1292 different ways of computing a *random* linear combination $\left(\sum_{j=1}^s r_j y_j\right)^2$ of the entries of y : first by
 1293 querying directly, and second by querying for the corresponding combination of $r_j r_k y_j y_k$. Goal
 1294 (2) is realized by querying for $Q_i(y)$ and verifying that $Q_i(y) = 0$. Soundness follows from the
 1295 Schwartz-Zippel Lemma [Mos10].

1296 **Fast verification.** The first feature is *fast verification* given input-independent preprocessing.
 1297 Whereas the cost of generating the queries q_i grows with the size of the verification circuit C , the
 1298 verifier can generate the queries before the input x is known and only keep r_1, \dots, r_n for later use.
 1299 Once x is known, the verifier can compute $\alpha = \sum_{i=1}^n r_i x_i$, using only $O(n)$ arithmetic operations.
 1300 Finally, once the LPCP answers a_1, a_2, a_3 are available, the verifier can decide whether to accept
 1301 by checking that $a_2 = a_1^2$ and $a_3 = \alpha$, which only takes $O(1)$ field operations. This fast verification
 1302 feature is particularly useful when the same queries are reused for proving multiple statements.

1303 **Reusable soundness.** Another feature of the Hadamard LPCP is a strong form of soundness that
 1304 holds even when the same queries are reused for verifying multiple proofs: for any x and proof π^* , the
 1305 prover can predict based on x, π^* alone whether the verifier will accept, except with $O(1/|\mathbb{F}|)$ error
 1306 probability (over the verifier’s unknown random queries). This means that when \mathbb{F} is sufficiently
 1307 big (say, $|\mathbb{F}| \geq 2^{80}$), learning whether the verifier accepted π^* reveals only a negligible amount of
 1308 information about the queries q_j . The latter strong soundness property is useful for making the SRS
 1309 in LPCP-based SNARGs reusable even when the prover can learn whether the verifier accepted each
 1310 instance. This is contrasted with classical (zk-)PCPs, which are inherently susceptible [CDIKLOV19]
 1311 to a “selective failure” attack in which a malicious prover can gradually learn the secret set of queries
 1312 when it is reused for multiple proofs. This can be done by slightly perturbing honestly generated
 1313 proofs and learning whether the verifier accepted each perturbed proof instance.

1314 2.2.3.2 From QAP (R1CS)

1315 Colloquially, constraint systems are often called “circuits”, but they can also take forms differ-
 1316 ent from Boolean and arithmetic circuits. While some schemes compile arithmetic circuits into
 1317 zero-knowledge proofs directly [CD98] it is sometimes more convenient to work with quadratic
 1318 arithmetic programs (QAP). QAPs are *bilinear constraint systems* (BCS) over finite fields where
 1319 each constraint allows a quadratic combination of two inputs. That is, multiplication by scalar,
 1320 addition and multiplication.

1321 To be precise, the BCS considered here is the same as the “system of rank-1 quadratic equations”

1322 defined in [BCGTV13, Appendix E], with a minor change in notation. Here, \vec{w} denotes just the
 1323 witness; whereas, in the latter, \mathbf{w} denotes the concatenation of the instance and the witness (and
 1324 thus added corresponding consistency checks between \mathbf{w} and the instance \mathbf{x}).

1325 These are also essentially identical to the ‘‘Rank 1 Constraint System (R1CS)’’ of libsnark [libsnark].
 1326 An equivalent formulation can be expressed as quadratic arithmetic programs (following [GGPR13;
 1327 PHGR13; BCGTV13; BCTV14b] and others), directly reasoning about polynomial divisibility.

1328 A bilinear constraint system reasons about two vectors: the *instance* consisting of n_x elements
 1329 and denoted $\vec{x} = (x_1, \dots, x_{n_x})$, and the *witness* consisting of n_w elements and denoted $\vec{w} =$
 1330 (w_1, \dots, w_{n_w}) . The constraint system says that the two are related by some number of constraints,
 1331 n_c . each of which is a quadratic equation of a specific form. All of the elements and operations are
 1332 over a large prime field \mathbb{F}_p , which we will represent here as the integers modulo a large prime p .
 1333 The specific prime, and the representation of field elements, are related to the elliptic curve used
 1334 in the QAP-based zkSNARK constructions.

1335 **Definition 2.2.1.** Let n_x, n_w, n_c be positive integers, and $n_z = n_x + n_w + 1$. A *bilinear constraint*
 1336 *system* over \mathbb{F}_q is a tuple $\mathcal{S} = ((\vec{a}_j, \vec{b}_j, \vec{c}_j)_{j=1}^{n_c}, n_x)$ where $\vec{a}_j, \vec{b}_j, \vec{c}_j \in \mathbb{F}_q^{n_z}$. Such a system \mathcal{S} is
 1337 *satisfiable* with an input $\vec{x} \in \mathbb{F}_q^{n_x}$ if the following is true:

1338 For these $\mathcal{S} = ((\vec{a}_j, \vec{b}_j, \vec{c}_j)_{j=1}^{n_c}, n_x)$ and $\vec{x} \in \mathbb{F}_q^{n_x}$;
 1339 there exists a witness $\vec{w} \in \mathbb{F}_q^{n_w}$;
 1340 such that $\langle \vec{a}_j, \vec{z} \rangle \cdot \langle \vec{b}_j, \vec{z} \rangle = \langle \vec{c}_j, \vec{z} \rangle$ for all for $j \in [n_c]$, where $\vec{z} = (1, \vec{x}, \vec{w}) \in \mathbb{F}_q^{n_z}$.

1341 Above, $\langle \dots, \dots \rangle$ denotes inner product of vectors over \mathbb{F}_q , and \cdot denotes multiplication in \mathbb{F}_q .

1342 For example, the bilinear constraint system

$$1343 \quad \mathcal{S} = \left(\left(\begin{array}{l} \vec{a}_1 = (0, 3, 0, 0), \vec{b}_1 = (1, 1, 0, 0), \vec{c}_1 = (4, 0, 5, 0) \\ \vec{a}_2 = (0, 1, 2, 0), \vec{b}_2 = (6, 0, 1, 0), \vec{c}_2 = (7, 0, 0, 1) \end{array} \right), 1 \right) \quad (2.1)$$

1344 represents the two bilinear constraints

$$1345 \quad \begin{aligned} \langle (0, 3, 0, 0), \vec{z} \rangle \cdot \langle (1, 1, 0, 0), \vec{z} \rangle &= \langle (4, 0, 5, 0), \vec{z} \rangle \\ \langle (0, 1, 2, 0), \vec{z} \rangle \cdot \langle (6, 0, 1, 0), \vec{z} \rangle &= \langle (7, 0, 0, 1), \vec{z} \rangle \end{aligned} \quad \text{where } \vec{z} = (1, x_1, w_1, w_2) \quad (2.2)$$

1346 or simply:

$$1347 \quad \begin{aligned} 3x_1 \cdot (1 + x_1) &= 5w_1 + 4 \\ (x_1 + 2w_1) \cdot (w_1 + 6) &= w_2 + 7 \end{aligned} \quad (2.3)$$

1348 By definition, this system \mathcal{S} is satisfiable with input x_1 if the following is true:

1349 For the instance x_1 ;
 1350 there exists a witness w_1, w_2 ;
 1351 such that

$$1352 \quad \begin{aligned} 3x_1 \cdot (1 + x_1) &= 5w_1 + 4 \\ (x_1 + 2w_1) \cdot (w_1 + 6) &= w_2 + 7 \end{aligned} \quad (2.4)$$

1353 When using zk-SNARK to prove this NP statement, the instance will be the public input that is
 1354 visible to the verifier, and the witness will be the additional data known only to the prover, and
 1355 used in the construction of the proof, but whose privacy is protected by the ZK property.

1356 The cost of producing the zk-SNARK proofs (in time and memory) will depend mostly on the
 1357 number of constraints, n_c . With this in mind, note how variable-by-variable multiplication are
 1358 “expensive” (each bilinear multiplication needs a new constraint and increases n_c), but linear com-
 1359 binations come “for free” in each constraint.

1360 A QAP considers a set of matrix equations such that

$$1361 \quad \left(\sum_j A_{ij} z_j \right) \times \left(\sum_j B_{ij} z_j \right) = \sum_j C_{ij} z_j \quad (2.5)$$

1362 for all i . Here A, B, C are public matrices and z is a vector comprising of public and private inputs
 1363 to the constraint system. To encode this as a polynomial system we first choose a set of distinct
 1364 points over the field κ_i . The κ_i evaluation points are often set as roots of unity, for efficiency reasons.

1365 Now define polynomials $u_j(X), v_j(X), w_j(X)$ such that $u_j(\kappa_i) = A_{i,j}, v_j(\kappa_i) = B_{i,j}$ and $w_j(\kappa_i) =$
 1366 $C_{i,j}$ for all j . These polynomials can be found via linear interpolation techniques. Then it is the
 1367 case that eq. (2.5) is satisfied if and only if, for all κ_i , we have that

$$1368 \quad \left(\sum_j u_j(\kappa_i) z_j \right) \left(\sum_j v_j(\kappa_i) z_j \right) = \left(\sum_j w_j(\kappa_i) z_j \right)$$

1369 It is a fact that a polynomial $f(X)$ is such that $f(\alpha) = 0$ if and only if $(X - \alpha)$ divides $f(X)$. Thus
 1370 our equation holds if and only if $(X - \kappa_i)$ divides

$$1371 \quad \left(\sum_j u_j(X) z_j \right) \left(\sum_j v_j(X) z_j \right) - \left(\sum_j w_j(X) z_j \right)$$

1372 for all κ_i . In particular this gives us that eq. (2.5) is satisfied if and only if

$$1373 \quad \left(\sum_j u_j(X) z_j \right) \left(\sum_j v_j(X) z_j \right) - \left(\sum_j w_j(X) z_j \right) = h(X)t(X)$$

1374 for some polynomial $h(X)$ and for $t(X) = \prod_i (X - \kappa_i)$.

1375 An IOP prover sends a proof $\pi = a(X), h(X)$ such that $a(\kappa_j) = z_j$ and $h(X)$ is defined as above.

1376 The verifier chooses random field element r and queries whether

$$1377 \quad \left(\sum_j u_j(r) a(\kappa_j) \right) \left(\sum_j v_j(r) a(\kappa_j) \right) - \left(\sum_j w_j(r) a(\kappa_j) \right) = h(r)t(r)$$

1378 By the Schwartz-Zippel Lemma the latter has $\frac{1}{\mathbb{F}_q}$ probability of occurring unless the prover has a
 1379 valid witness.

1380 2.2.3.3 Line-Point

1381 **Contribution wanted**

1382 2.2.3.4 Fully Linear PCP

1383 See Referencesparadigms:IT:linear-IOP:fully-linear-IOP (Section 2.2.6.3)

1384 **2.2.4 MPC in the head**

1385 A *secure multiparty computation* (MPC) protocol [Lin20] allows n parties to compute a given
 1386 *functionality* f , mapping n local inputs to n local outputs, while “hiding” the inputs. The “MPC-
 1387 in-the-head” paradigm [IKOS09] allows compiling simple kinds of MPC protocols (with perfect
 1388 correctness guarantees) into zk-PCPs.

1389 **FF2.7. Proposed future figure (illustration, diagram, ...):**

1390 Illustrate an MPC in the head and how it becomes a ZKP

1391 **From an MPC protocol to a zk-PCP.** Given an NP-relation $R(x, w)$, define an n -party MPC
 1392 functionality $f(x; w_1, \dots, w_n) = R(x, w_1 \oplus w_2 \oplus \dots \oplus w_n)$, where the input of each party P_i consists
 1393 of the public statement x and a secret input w_i , and where $R(x, w)$ is 1 if $(x, w) \in R$ and is 0
 1394 otherwise. Intuitively, w_i can be thought of as a *secret-share* of w . The output of f is delivered to
 1395 all parties. Now let Π_f be an MPC protocol realizing f .

1396 The zk-PCP prover on input (x, w) generates a proof π as follows. First, it randomly splits w into
 1397 n random *additive shares* w_i of w , subject to $w = w_1 \oplus w_2 \oplus \dots \oplus w_n$. Next, the prover performs “in
 1398 its head” a virtual execution of Π_f on the common input x and the secret inputs (w_1, \dots, w_n) . This
 1399 involves picking a secret random input r_i for each party P_i , and computing the messages exchanged
 1400 until all parties terminate with an output. Then, the proof becomes $\pi = (V_1, \dots, V_n)$, where V_i
 1401 is the entire view of P_i in the virtual execution of Π_f . V_i consists of w_i, r_i and all the messages
 1402 *received* by P_i . The zk-PCP verifier queries a pair of random symbols V_i, V_j and checks that: (1)
 1403 the two views are *consistent*, i.e., the incoming messages in each view coincide with the outgoing
 1404 messages implicit in the other view; (2) the outputs of P_i and P_j implicit in the views are both 1.
 1405 The verifier accepts if both conditions hold and otherwise rejects. Ishai et al. also discussed other
 1406 flexibility in the design. For example, to increase soundness, one could use an active MPC protocol
 1407 or check more than one pair of parties.

1408 **Security.** The above zk-PCP construction is quite easy to analyze:

- 1409 • *Completeness:* follows from the definition of f and the (perfect) correctness of Π_f .
- 1410 • *Zero knowledge:* follows (i) from the fact that the pair of witness shares w_i, w_j reveal no
 1411 information about w and (ii) from the security of the MPC protocol Π_f that it is private
 1412 against any collusion of two parties.
- 1413 • *Soundness:* If there is no w for which $R(x, w) = 1$, there are two cases to analyze:
 1414 1. If the views V_i correspond to a valid execution of Π_f on inputs (w_1^*, \dots, w_n^*) , then the
 1415 output in all views will be 0, and the verifier will always reject
 1416 2. If the views are inconsistent with any valid execution of Π_f ; then there must be at least
 1417 one pair of inconsistent views. The verifier will select it with probability $1/\binom{n}{2}$ and
 1418 reject it. This can be reduced to 2^{-k} via $O(k)$ repetitions.

1419 **Efficiency.** MPC in the head is especially competitive for small problem sizes, or in settings
 1420 where the prover’s computation cost is the main bottleneck. It can be instantiated with basic
 1421 symmetric-key primitives. Note that MPC protocols are designed for a distributed setting, in

1422 which no single entity has full information, whereas the usual setting of proof systems is one where
 1423 the full information is available to the prover. This explains why MPC-based zk-PCPs cannot reach
 1424 the asymptotic level of succinctness of other systems (see Section 2.2.5).

1425 **Applicability.** In the protocol described above, the zk-PCP can be based on any MPC protocol
 1426 Π_f for f , over secure point-to-point channels, that offers security against two “semi-honest” parties
 1427 (i.e., if *all parties* follow the protocol, then each pair of parties learns nothing beyond their inputs
 1428 and the output). That is, the joint view of any pair of parties can be simulated given their inputs
 1429 and outputs of f alone, independently of the inputs of the other parties. Simple protocols of this
 1430 kind exist unconditionally for $n \geq 5$ parties [BGW88; CCD88; Mau06]. The mentioned construc-
 1431 tion can be modified and extended in useful ways. For example: (i) the perfect correctness of the
 1432 MPC protocol can be relaxed at the expense of making the zk-PCP interactive (see Section 2.2.5.1);
 1433 (ii) the MPC model can be augmented of input-independent preprocessing [KKW18]; (iii) using
 1434 MPC protocols for a large number of parties enables achieving negligible soundness error by using
 1435 MPC protocols with security against *malicious* parties.

1436 IT compilers can be used to port to the domain of ZKPs the big array of techniques existing in the
 1437 domain in efficient MPC. For instance, MPC protocols based on algebraic geometric codes [CC06]
 1438 can be converted into (statistically sound) ZKP protocols for Boolean circuit satisfiability in which
 1439 the ratio between the communication complexity and the circuit size is constant, while the sound-
 1440 ness error is negligible in the circuit size [IKOS09]. Similarly, one can exploit MPC protocols that
 1441 make a black-box use of general rings [CFIK03] or other algebraic structures [CDIKMRR13], MPC
 1442 protocols for linear algebra [CD01] problems, and many more.

1443 2.2.5 Interactive Oracle Proof (IOP)

1444 Compared with a PCP system, an IOP system is more powerful because it allows additional inter-
 1445 action between the prover and the verifier. Note the prover does not have any control about the
 1446 (random) challenges to be answered because they are selected by the verifier. The extra interaction
 1447 is helpful in designing more efficient provers. The interactive zk-PCPs achieve *statistical* correct-
 1448 ness guarantees, by passing the problem (in a non-interactive compiler from MPC to zk-PCP) of
 1449 a malicious prover that could choose the randomness of MPC parties in a way that violates MPC
 1450 correctness and hence zk-PCP soundness.

1451 IOP generalizes the notion of [Interactive PCP](#), and coincides with the notion of Probabilistically
 1452 Checkable Interactive Proof.

1453 **FF2.8. Proposed future figure (illustration, diagram, ...):**

1454 Illustrate a generic IOP

1455 2.2.5.1 Interactive PCP

1456 **Earlier applications.** For NP-relations R computed by constant-depth circuits, interactive PCPs
 1457 [KR08] can have surprisingly good parameters. Whereas for classical PCPs with low-communication
 1458 the proof size is bigger than the circuit size of R (likely to be inherent [FS11]), in the interactive
 1459 case the proof size can be reduced to roughly the witness length, still with low communication. This

1460 was subsequently generalized from constant-depth R to low-depth R (e.g., polylogarithmic depth)
 1461 [GKR08]. A *constant-round* variant for space-bounded computations is also possible [RRR16].

1462 **Interactive Oracle Proofs.** The mentioned “interactive PCP” model has a single proof π , whose
 1463 symbols can be queried by the verifier, and additional (low-communication) interaction between
 1464 the verifier and the prover. Taking interaction in PCPs to its full level of generality, there is the
 1465 *Interactive Oracle Proof* (IOP) model [BCS16; RRR16], which allows multiple proofs π_i to be se-
 1466 quentially generated by the prover and queried by the verifier. More concretely, in (the public-coin
 1467 variant of) the IOP model, each π_i comes in response to an unpredictable random challenge r_i .
 1468 The verifier’s queries to the proofs π_i can be made in the end of the interaction. The IOP is *fully*
 1469 *succinct* if the total number of bits from all proofs π_i that are read by the verifiers polylogarithmic
 1470 in the instance size. The additional interaction allowed by the IOP model makes the design of fully
 1471 succinct proofs in this model simpler than in the classical PCP model or even in the interactive
 1472 PCP model discussed above. This simplicity also leads to remarkable efficiency improvements.

1473 2.2.5.2 Fast RS IOPP

1474 **Contribution wanted**

1475 2.2.6 Linear IOP

1476 There are two orthogonal relaxations of the classical PCP model: adding interaction, and using
 1477 linear queries instead of point queries. The linear IOP (LIOP) model [BBCGI19] combines these
 1478 two relaxations in a natural way. As in the IOP model, the prover generates a sequence of proof
 1479 vectors $\pi_i \in \mathbb{F}^{m_i}$ in multiple rounds, where each π_i is a response to a random challenge r_i . In the
 1480 end of this interaction, the verifier can make linear queries to each π_i , as in the LPCP model. (The
 1481 LIOP model is closely related to an earlier Interactive Linear Commitment (ILC) [BCGGHJ17].)

1482 **FF2.9. Proposed future figure (illustration, diagram, ...):**

1483 Illustrate a generic linear IOP

1484 2.2.6.1 IP based

1485 **The GKR protocol.** The “doubly efficient” GKR interactive proof protocol [GKR08] (i.e., with
 1486 efficient prover and efficient verifier), based on the classical *sum-check* protocol [LFKN92] (see also
 1487 [Tha20]), can be cast as an LIOP for NP with the following features.

- 1488 • If $R(x, w)$ is computed by a layered arithmetic circuit of size s and depth d over \mathbb{F} with m
 1489 inputs, then there are $\approx d$ rounds in which the proofs π_i are vectors over \mathbb{F} of size $\approx \log s$
 1490 each, and a single round in which the proof vector of size $\approx m$. (Here, \approx hides multiplicative
 1491 factors that are at most polylogarithmic in s .)
- 1492 • The verifier makes a constant number of linear queries to each proof and applies a decision
 1493 predicate of degree 2. The soundness error is $\approx d/|\mathbb{F}|$. Furthermore, the verifier’s running
 1494 time can be made $\approx d + m$ if the circuit has a *succinct description* of an appropriate kind.

1495 Concretely, the circuit should be *log-space uniform* in the sense that relevant local information
 1496 about a gate can be computed in logarithmic space given the gate label. The latter uniformity
 1497 feature was crucial in the original context of the GKR protocol, namely fast verification of shallow
 1498 polynomial-size *deterministic* circuits, with no witness w . However, in the context of proof systems
 1499 for NP (with or without zero knowledge), an LIOP as above is meaningful even for general verifica-
 1500 tion circuits that do not have a succinct description. Using statement-independent preprocessing,
 1501 one can still achieve fast online verification even in the non-uniform case.

1502 A more sophisticated related protocol applicable to space-bounded computations can also be cast
 1503 as a (constant-round) LIOP for NP [RRR16; Gol18].

1504 With optimizations, the GKR protocol yields good concrete efficiency, as achieved in subsequent im-
 1505 provements [CMT12; VSBW13; Tha13; XZZPS19]. For *layered* arithmetic circuits over large fields,
 1506 Libra [XZZPS19] obtained a *constant computational overhead* on the prover side in the arithmetic
 1507 setting. Virgo++ [ZLWZSXZ21] generalized the GKR protocol to arbitrary arithmetic circuits with
 1508 the same constant computational overhead on the prover as the protocol for layered circuits.

1509 This can be compared to the simpler constant-overhead zk-LPCP for NP discussed above, which
 1510 applies to arbitrary verification circuits (of arbitrary structure, depth, and witness length) but offers
 1511 no form of succinctness. For efficiently verifying *deterministic* low-depth computations, i.e., without
 1512 ZK or even a witness, GKR variants can be used without a cryptographic compiler, i.e., with actual
 1513 interaction, and this is practical when the communication is sufficiently cheap [WJBSTWW17].

1514 2.2.6.2 Polynomial IOP

1515 A polynomial IOP (PIOP) is a generalization of a standard IOP (Section 2.2.5) in which in a
 1516 subset of rounds, the prover is permitted to “send” a polynomial to the verifier, and the verifier is
 1517 permitted to query the polynomial at a small number of evaluation points [BFS20; CHMMVW20].
 1518 Polynomial IOPs can be compiled to succinct arguments using polynomial commitment schemes
 1519 (Section 2.3.5.2). Essentially, any polynomial p sent by the prover in the PIOP is replaced with a
 1520 succinct commitment to said polynomial via a polynomial commitment scheme, and any evaluation
 1521 query the verifier makes to p is answered via a proof of evaluation supplied by the polynomial
 1522 commitment scheme.

1523 Many protocols in the literature can be viewed as PIOPs, even if they were not originally expressed
 1524 in this language. For example, vSQL [ZGKPP17] implicitly adapt the GKR interactive proof
 1525 [GKR08] for circuit evaluation into a PIOP for circuit satisfiability (roughly, for a given circuit
 1526 \mathcal{C} , to prove that there exists a witness w such that $\mathcal{C}(x, w) = y$, the PIOP prover first sends a
 1527 low-degree extension polynomial of the witness w , and then applies the GKR interactive proof to
 1528 establish that $\mathcal{C}(x, w) = y$). There are various related PIOPs [ZXZS20; ZLWZSXZ21; WTTW18;
 1529 XZZPS19]. As another example [BTVW14], a 2-prover interactive proof for circuit satisfiability
 1530 can yield a PIOP (the prover first sends an extension-polynomial of a “correct transcript” for the
 1531 circuit, and then a single invocation of the sum-check protocol [LFKN92] is used to confirm that
 1532 the transcript is correct). Spartan [Set20] extends this PIOP from circuit-satisfiability to R1CS.
 1533 Marlin [CHMMVW20], Aurora [BCRSVW19], and Fractal [COS20] give PIOPs for R1CS based on
 1534 a “univariate sum-check protocol”.

1535 **2.2.6.3 Fully Linear IOP**

1536 In some settings the verifier does not have full access to the input statement x . For instance, the
 1537 input can be partitioned between two or more parties (e.g., banks or hospitals). In these distributed
 1538 settings, a proof system can still be used if the queries performed to the input x can be implemented
 1539 by *local* queries to the shares held by each of the parties. For example, if x is secret-shared using a
 1540 linear secret-sharing scheme (or linearly-homomorphic encryption or commitment scheme), then it
 1541 is possible to efficiently make a linear query to x by *locally* applying a linear query to each share.

1542 The above settings naturally call for a stronger variant of the LPCP and LIOP models in which
 1543 linear queries apply *jointly* to an input $x \in \mathbb{F}^n$ and a proof $\pi \in \mathbb{F}^m$, where the verifier has no direct
 1544 access to the input except via such queries. IT proof systems of this kind were studied [BBCGI19]
 1545 under the name *fully linear* proof systems. This notion extends to a ZK version, where the verifier
 1546 learns *nothing about x and w* except for the fact that $x \in L$ (compared to only hiding w , as in
 1547 the usual definition). This is formalized in a ZK definition where the simulator does not get access
 1548 to the input x . Note that this makes fully linear proof system with low query complexity meaning-
 1549 ful even for polynomial-time languages, and even if $P = NP$. This is akin to proofs of proximity
 1550 [BGHSV04], except that the latter only give the weaker guarantee that the input is *close* to being
 1551 in L , and is also closely related to *holographic proofs* [BFLS91] in which the verifier can query a
 1552 suitable encoding of the input.

1553 Fully linear proof systems are motivated by the goal of efficient *distributed zero-knowledge* proofs
 1554 where the input statement x is known to the prover but is distributed between multiple veri-
 1555 fiers. This should not to be confused with other kinds of distributed zero-knowledge proofs (e.g.,
 1556 [WZCPS18]) that allow for parallel computation across multiple machines.

1557 Thus, a natural goal is to design fully linear PCPs (FLPCPs) and fully linear IOPs (FLIOPs) with
 1558 small proof length and query complexity, and in particular, make these *sublinear in the input length*.
 1559 This goal is well-motivated even for simple languages, such as the “Booleanity check” language
 1560 $\{0, 1\}^n \subseteq \mathbb{F}^n$, or the inner-product language consisting of concatenations of pairs of vectors in $\mathbb{F}^{n/2}$
 1561 whose inner product is 0. It turns out that both the concrete LPCP and LIOP discussed above can
 1562 be made fully linear with the same number of queries. However, whereas the proof length of the
 1563 Hadamard LPCP and the LPCP of GGPR is at least linear in n , the GKR protocol [GKR08] yields
 1564 an FLIOP whose total proof and answer length is comparable to the circuit *depth*, or $O(\log^2 n)$
 1565 for the above examples, with a similar number of rounds. Similarly, the RRR protocol [RRR16]
 1566 implies constant-round sublinear FLIOPs for low-space computations.

1567 For languages that are recognized by low-degree polynomials, as in the above examples, there are
 1568 simpler and more efficient fully linear proof systems [BBCGI19]. For example, a non-interactive
 1569 zk-FLPCP exists for proving that x satisfies a single degree-2 equation (as in L_1 above) with a proof
 1570 of size $O(\sqrt{n})$ and a similar number of queries, which is optimal [Kla03]. This protocol is a ZK
 1571 variant of a communication complexity upper bound [AW09]. For languages recognized by systems
 1572 of constant-degree equations (including both of the above examples), the proof size and query
 1573 complexity can be reduced to $O(\log n)$ at the cost of settling for a zk-FLIOP with $O(\log n)$ rounds
 1574 of interaction. These constructions start with a generalization of a GGPR-style zk-LPCP that takes
 1575 advantage of repeated structures, and then improve efficiency via balancing and recursion.

1576 **2.2.7 Ideal Linear Commitment (ILC)**

1577 The *ideal linear commitment* (ILC) model provides another perspective [BCGGHJ17] of LIOP. A
 1578 proof in the ILC model can be viewed as an LIOP where each proof vector π_i is parsed as a *matrix*
 1579 $\Pi \in \mathbb{F}^{n_i \times m_i}$, and each query to π_i is specified by a *vector* $q \in \mathbb{F}^{m_i}$, returning the matrix-vector
 1580 product Πq . Alternatively, an ILC can be viewed as a standard LIOP if the underlying field is
 1581 replaced by a vector space.

1582 The ILC model relaxes the Linear Interactive Proofs (LIP) model [BCIOP13], since each ILC proof
 1583 matrix may be the output of an arbitrary function of the input and the verifier’s messages, whereas
 1584 each LIP proof matrix must be a linear function of the verifier’s messages.

1585 **FF2.10. Proposed future figure (illustration, diagram, ...):**

1586 Illustrate a generic ILC

1587 Instantiated with linear-time encodable error-correcting codes [Spi96] and with linear-time com-
 1588 putable hash functions [AHIKV17], the compiler respects the asymptotic computational complexity
 1589 of the underlying ILC proof.

1590 They also show a construction of constant-overhead “square-root succinct” zk-ILC for the satisfi-
 1591 ability of arithmetic circuits. Concretely, for an NP-relation $R(x, w)$ computed by an arithmetic
 1592 circuit over \mathbb{F} of size s , the communication is $\approx \sqrt{s}$, the round complexity is $O(\log \log s)$, and both
 1593 parties can be implemented a RAM program whose running time is dominated by $O(s)$ field op-
 1594 erations, where the soundness error is negligible in $\log |\mathbb{F}|$. Combining this ILC with a linear-time
 1595 instantiation of the cryptographic compiler yields (under plausible cryptographic assumptions) a
 1596 square-root succinct zero-knowledge argument for arithmetic circuit satisfiability with constant
 1597 computational overhead.

1598 This ILC approach obtains a ZKP system with constant computational overhead. It has better
 1599 asymptotic succinctness than the simple LPCP-based approach. It is incomparable to the GKR-
 1600 based approach: its succinctness is typically worse (unless the circuit is relatively deep), but it
 1601 achieves constant overhead for arbitrary circuits and witness length. Unlike the other two ap-
 1602 proaches, the hidden multiplicative constants of the ILC-based system seem quite large, and it is
 1603 open whether it can be optimized to have competitive concrete efficiency features.

1604 Note that all three approaches (GKR, ILC, LPCP) for zero knowledge with constant computa-
 1605 tional overhead can only achieve negligible soundness error for *arithmetic* computations over fields
 1606 of super-polynomial size. In the parallel RAM model, an approach for amortizing away the (parallel)
 1607 prover computational overhead, at the cost of a minor increase in the number of processors, was pro-
 1608 posed in the SPARKS system [EFKP20]. However, when considering the standard metric of Boolean
 1609 circuit size or (sequential) running time on a RAM machine, constant-overhead zero knowledge is
 1610 still wide open, and there are no candidate constructions under any cryptographic assumption.

1611 2.3 Cryptographic Compilers (CC)

1612 This section describes cryptographic compilers (CC) for several of the IT systems described in
 1613 Section 2.2. A CC transforms an IT proof system into a real-world protocol that involves direct
 1614 interaction between the prover and the verifier. The primitives used by a CC are realized by
 1615 (possibly heuristic) concrete instantiations, such as concrete hash functions or pairing-friendly
 1616 elliptic curves. This yields a concrete scheme that can be implemented and used in real-world
 1617 applications. The CC may also provide extra desirable properties, such as eliminating interaction,
 1618 shrinking the size of some of the messages, or even adding a ZK feature to an IT proof system that
 1619 does not have it.

1620 While the separation between IT proof systems and cryptographic compilers is useful, it is not
 1621 always precise. For some concrete ZKP schemes, there is no neat way to break them into such
 1622 components, such that when putting them together one obtains the original scheme. Thus, the
 1623 abstractions and classifications presented herein can be considered as useful didactic and taxonomic
 1624 tools, while much room is left for ingenuity and optimizations for specific ZKP schemes, resulting
 1625 in differences in concrete in performance and parameters.

1626 The next subsections describe cryptographic compilers for zk-PCP (§2.3.1) LPCP (§2.3.2), MPC-
 1627 in-the-Head (§2.3.3), IOP (§2.3.4), LIOP (§2.3.5) and ILC (§2.3.6).

1628 **FF2.11. Proposed future figure (illustration, diagram, ...):**

1629 Illustration of generic crypto compilers, including labels for each main type of IT proof system

1630 2.3.1 CC for zk-PCP

1631 **Example: 3-coloring.** In the 3-coloring example $R_{3\text{COL}}$ of section 2.2.2.1 above, we assumed
 1632 the availability of commitments. To realize this protocol in the real world, we use a cryptographic
 1633 compiler that relies on cryptographic commitment schemes which are secure under computational
 1634 assumptions. This compiler proceeds as follows. Given (x, w) , the prover uses the zk-PCP
 1635 prover to generate a proof $\pi \in \Sigma^m$ and uses the underlying commitment scheme to independently
 1636 commits to each of the m symbols. The verifier uses the zk-PCP verifier to pick a subset Q of
 1637 the symbols, which it sends as a challenge to the prover. The prover opens the symbols π_Q after
 1638 checking that Q is valid (in the sense that it can be generated by an honest verifier). The verifier
 1639 uses the decision predicate D of the zk-PCP to decide whether to accept.

1640 **Using standard cryptographic assumptions.** While simple and natural, the analysis of the above
 1641 compiler when using a standard *computationally-hiding* commitment scheme [Gol01] is more sub-
 1642 tle than it may seem. In particular, efficient simulation requires that the distribution of Q have
 1643 polynomial-size support. This indeed applies to the basic version of the zk-PCP for $R_{3\text{COL}}$, but
 1644 not to the one with amplified soundness. As a result, the compiler yields a (constant-round) zero-
 1645 knowledge protocol with poor soundness, which can be amplified via sequential repetition. An
 1646 alternative cryptographic compiler, which avoids the polynomial-size support restriction by using
 1647 a *statistically-hiding* commitment scheme (and an even more subtle analysis), is implicit in the
 1648 work on constant-round zero-knowledge proofs for NP [GK96]. Both of the above compilers yield

1649 zero-knowledge proof protocols in which the communication complexity is bigger than that of com-
 1650 municating π ; ideally, we would like to make it comparable to only communicating π_Q . As we will
 1651 see, this can make a big difference.

1652 When instantiated with statistically-binding (and computationally-hiding) commitments, the above
 1653 compilers yield statistically-sound *proofs* rather than computationally-sound arguments. In this
 1654 case, their high communication cost seems inherent, as there is a strong complexity-theoretic ev-
 1655 idence [GH97; GVW02] that the prover-to-verifier communication in proof systems for hard NP-
 1656 relations cannot be much smaller than the witness size. On the other hand, a different cryptographic
 1657 compiler [IMSX15] can use any *collision resistant hash function* to obtain a zero-knowledge *argu-*
 1658 *ment* whose communication complexity is close to just the size of π_Q , which for some zk-PCPs (that
 1659 will be discussed later) can be much smaller than the witness size. The first compiler of this kind
 1660 (implicit in the work of Kilian [Kil92]) can obtain a similar ZK argument from any PCP, namely
 1661 zk-PCP without the ZK requirement. However, in contrast to compilers based on zk-PCP, Kilian’s
 1662 compiler uses the underlying cryptographic primitive in a *non-black-box* way [RTV04], which makes
 1663 it inefficient in practice.

1664 **Practical NIZK compiler in the random oracle model.** All of the previous black-box compilers can
 1665 be analyzed unconditionally if the underlying “symmetric” cryptographic primitive is abstracted as
 1666 a random oracle. But one can actually go further and make these interactive public-coin protocols
 1667 *non-interactive* via the *Fiat-Shamir* transform [FS87]. Combining the two steps, we get concretely
 1668 efficient compilers from any zk-PCP to NIZK in the random oracle model. The latter is then
 1669 heuristically instantiated with a practical hash function based on, say, SHA-256 or AES. See the
 1670 background part for more discussion of this methodology. Interestingly, even in the random oracle
 1671 model, the NIZK does not entirely dominate the interactive protocol on which it is based, since
 1672 removing interaction can come at a price. For instance, consider an interactive protocol with
 1673 soundness error of $2^{-\sigma}$, where σ may be of the order of 40 or 64, as discussed in Section 1.8.3. In
 1674 the NIZK obtained via the Fiat-Shamir transform, the prover can convince the verifier of a false
 1675 statement with certainty by generating roughly 2^σ random transcripts, until finding one that would
 1676 lead the verifier to accept. This means that the non-interactive variant should rely on a zk-PCP
 1677 with a considerably smaller soundness error, i.e., with higher σ equal to the computational security
 1678 parameter κ . This would increase the concrete communication and computation costs, though by
 1679 a small factor (of about κ/σ).

1680 **NIZK from standard cryptographic assumptions.** A recent line of work shows how to instantiate
 1681 the random oracle in NIZK proofs obtained via the Fiat-Shamir transform under standard crypto-
 1682 graphic assumptions. These works rely on a special type of correlation-intractable hash functions
 1683 [CGH04] together with a special kind of Σ -protocols [Dam10], namely 3-message honest-verifier
 1684 (computational) zero-knowledge proof systems that satisfy some additional properties. The latter
 1685 in turn implicitly rely on a special kind of zk-PCP that can be instantiated with Blum’s graph
 1686 Hamiltonicity protocol [Blu87] but remains to be more systematically explored. In contrast to
 1687 these recent NIZK protocols, the classical approach [FLS99] for basing NIZK for NP on standard
 1688 cryptographic assumptions uses a very different kind of information-theoretic proof systems referred
 1689 to as zero-knowledge proofs in the *hidden bits model*. Known proofs of this type are more involved
 1690 and less efficient than their zk-PCP counterparts. However, the cryptographic compiler from the
 1691 hidden bits model to NIZK can rely on the intractability of factoring or, more generally, a suitable
 1692 kind of *trapdoor permutation* [CL18]. An interesting question is whether one can obtain a similar
 1693 cryptographic compiler from zk-PCP.

1694 **2.3.2 CC for Linear PCP (LPCP)**

1695 Cryptographic compilers for linear PCPs are possible at the cost of an expensive (but reusable)
 1696 trusted setup, and “not efficiently falsifiable” cryptographic assumptions [Nao03]. These compilers
 1697 can yield zk-SNARKs with very short proofs (roughly 1000 bits, or even less in some settings).
 1698 The use of this type of compilers was implicit in some initial schemes [Gro10; Lip12], and was later
 1699 considered explicitly [BCIOP13; GGPR13].

1700 Let us start by considering the following natural attempt for compiling an LPCP into a *designated verifier*
 1701 SNARG with a reusable structured reference string (SRS). The verifier generates
 1702 LPCP queries q_1, \dots, q_d (e.g., $d = 3$ in the case of the Hadamard LPCP), and lets the SRS σ
 1703 include a linearly homomorphic encryption of (each entry of) the verifier’s queries, denoted by
 1704 $\sigma = (E(q_1), \dots, E(q_d))$. The verifier keeps the secret key k_V that can be used for decryption.
 1705 Now, given an input (x, w) and σ , the prover computes an LPCP proof vector π , and uses the
 1706 linear homomorphism of E to compute a short SNARG proof $\hat{\pi}$ consisting of the d ciphertexts
 1707 $\hat{\pi} = (E(\langle \pi, q_1 \rangle), \dots, E(\langle \pi, q_d \rangle))$ that it sends as a SNARG proof to the verifier. The verifier uses
 1708 the secret key k_V to decrypt the LPCP answers $a_1 = \langle \pi, q_1 \rangle, \dots, a_d = \langle \pi, q_d \rangle$, and applies the LPCP
 1709 decision predicate to decide whether to accept or reject.

1710 The above attempt to implement LPCP under the hood of homomorphic encryption clearly satisfies
 1711 the completeness requirement. Moreover, it is tempting to believe that, since the encryption hides
 1712 the queries, the proof π is independent of the queries and thus soundness holds as well. However,
 1713 this simplistic soundness argument is flawed for two reasons.

1714 First, while a standard linearly homomorphic encryption scheme E supports computing linear func-
 1715 tions on encrypted inputs, it provides no guarantee that *only* linear functions can be computed. In
 1716 fact, linearly homomorphic encryption can be fully homomorphic [Gen09], in which case soundness
 1717 completely breaks down. The solution to this problem is essentially to “assume it away” by relying
 1718 on an encryption scheme E that is conjectured to support *only* linear computations on encrypted
 1719 inputs. (Alternatively, this can be extended to *affine* computations that include a constant term.)
 1720 This strong notion of “linear-only encryption” [BCIOP13] will be discussed in more detail below.

1721 A second problem is that there is nothing in the above solution that prevents a malicious prover from
 1722 using a different proof vector π_i^* for each encrypted query $E(q_i)$. This is beyond the capability of a
 1723 malicious prover in the LPCP model and may thus violate soundness. A solution to this problem
 1724 is to have the verifier add to the LPCP an additional query which is a random linear combination
 1725 of of the original queries, namely $q_{d+1} = \rho_1 q_1 + \dots + \rho_d q_d$, and check that the answer to this query
 1726 satisfies $a_{d+1} = \rho_1 a_1 + \dots + \rho_d a_d$. This can be viewed as a simple IT compiler from LPCP to a
 1727 1-round *linear interactive proof* (LIP) [BCIOP13], a stronger information-theoretic proof system
 1728 that can be viewed as restricting a standard interactive proof by allowing provers (both honest and
 1729 malicious) to only compute linear functions of the verifier’s messages.

1730 To sum up: the modified compiler proceeds in two steps. First, an information-theoretic compiler
 1731 is applied to convert the LPCP into a LIP. If the LPCP has proof size m and d queries, the LIP
 1732 resulting from the simple compiler described above has verifier message consisting of $(d + 1) \cdot m$
 1733 field elements and prover message consisting of $d + 1$ field elements. Then, linear-only encryption
 1734 is used to compile the LIP into a designated-verifier SNARG with SRS that consists of $(d + 1) \cdot m$
 1735 ciphertexts and proof that consists of $d + 1$ ciphertexts. This transformation respects all of the
 1736 additional features of LPCPs we discussed in the context of the Hadamard LPCP: zero knowledge,

1737 fast verification, and reusable soundness. The latter means that the SRS of the resulting SNARG
 1738 can be safely reused for multiple proofs. Finally, when the LPCP is also a “proof of knowledge”
 1739 (which is the case for all natural constructions), and when the linear-only encryption is “extractable”
 1740 (a plausible assumption for concrete instantiations), the resulting (zk)-SNARG is also a proof of
 1741 knowledge, namely it is a (zk)-SNARK.

1742 There is one remaining issue: the above approach seems inherently restricted to the *designated*
 1743 *verifier* setting, since only the verifier is able to decrypt the encrypted LIP answers. While this is
 1744 good enough for some applications, many applications of SNARGs require public verification. The
 1745 solution (initially implicit in the work of Groth [Gro10]) is to rely once again on a special property
 1746 of the LPCP, which is respected by the transformation to LIP. Suppose the LIP verifier’s decision
 1747 predicate is *quadratic* in the following sense: to decide whether to accept, the verifier tests equalities
 1748 of the form $p_x(\mathbf{u}, \mathbf{a}) = 0$, where p_x is a *degree-2* polynomial determined by the input statement x ,
 1749 the vector \mathbf{u} contains state information determined by the LIP query, and the vector \mathbf{a} contains
 1750 the LIP answers. Indeed, this is the case for the Hadamard-based LIP. Then, public verification
 1751 can be achieved by using an encryption scheme that allows such quadratic tests to be performed on
 1752 an encrypted input without knowing the decryption key. Fortunately, this kind of functionality is
 1753 supported by pairing-based cryptography. If the SRS σ includes a “pairing-friendly encryption” of
 1754 the LIP query along with the state information \mathbf{u} , which can be implemented using *bilinear groups*,
 1755 the prover on input (x, w) can compute an encryption of the LIP answers \mathbf{a} , and then *everyone* can
 1756 check that the encrypted \mathbf{u}, \mathbf{a} satisfy the quadratic relation defined by x .

1757 Several QAP-based constructions (e.g., [GGPR13; Gro16]) can be roughly recast as an IT proof
 1758 system in the [Linear PCP model](#), and then be cryptographically compiled (e.g., with [BCIOP13])
 1759 in a way that results in different proof systems (less succinct by constant factors).

1760 2.3.3 CC for MPC-in-the-Head

1761 **Contribution wanted: Contribution needed**

1762 2.3.4 CC for IOP

1763 Using a cryptographic compiler [BGGHKMR90] based on one-way functions, the interactive PCPs
 1764 from Section 2.2.5.1 yield statistically-sound ZKP protocols for low-depth (or space-bounded) NP
 1765 relations in which the communication complexity is comparable to the witness length.

1766 As in the case of classical PCPs, there are cryptographic compilers that use a collision-resistant hash
 1767 function (respectively, a classical or even quantum random oracle) [CMS19] to convert fully succinct
 1768 IOPs into interactive (respectively, non-interactive and transparent) fully succinct arguments for
 1769 NP. These compilers naturally extend the ones for classical PCPs. In the interactive variant, the
 1770 prover uses a Merkle tree to succinctly commit to each proof, following which the verifier generates
 1771 and sends the random challenge for the next proof. In the end, the verifier challenges the prover to
 1772 open the subset of proof symbols that the IOP verifier wants to query, to which the prover responds
 1773 by revealing the small amount of relevant information on the Merkle trees. The non-interactive
 1774 variant is obtained from the interactive one via the Fiat-Shamir heuristic, though the analysis in
 1775 the random oracle model is considerably more challenging in the interactive case and requires some

1776 extra assumptions on the underlying IOP.

1777 On the practical side, transparent SNARKs, such as STARK [BBHR18b], Aurora [BCRSVW19]
 1778 and Fractal [COS20], rely on concretely efficient IOPs. While not quite as succinct as the group-
 1779 based SNARKs discussed next (with typical proof size in the range of 100–200 kB in the former
 1780 vs. 1–2 kB in the latter), they have the advantages of being transparent, plausibly post-quantum,
 1781 and avoiding altogether the use of public-key cryptography. The latter can be useful for allowing
 1782 faster provers. A key technical ingredient in these practical IOPs is the FRI protocol [BBHR18a]:
 1783 an interactive test for proximity of an oracle to a Reed-Solomon code. A useful feature of the FRI
 1784 protocol is that it can be realized using a strictly linear number of arithmetic operations. However,
 1785 the IOPs that build on top of it still require quasilinear time on the prover side.

1786 On the theoretical side, the IOP model gives rise to asymptotic efficiency features that are not
 1787 known to be achievable with classical PCPs. See [BCGGRS19; RR20] for recent progress on
 1788 minimizing the *proof size* of IOPs. While proof size is not the main parameter of interest in
 1789 cryptographic applications of IOPs, as it only serves as a lower bound on the prover’s running
 1790 time, new techniques for constructing IOPs are likely to lead to progress on the concrete efficiency
 1791 of IOP-based proof systems. More directly relevant to the concrete efficiency of IOPs are works on
 1792 improving the analysis of their *soundness error* [BKS18; BGKS20; BCIKS20].

1793 2.3.4.1 CC for Interactive PCP

1794 **Contribution wanted**

1795 2.3.5 CC for Linear IOP

1796 **Contribution wanted: CC for Linear IOP (other than fully linear) / IP-based**

1797 2.3.5.1 CC for Fully Linear IOP

1798 By having the prover secret-share each proof vector, a fully linear proof system can be compiled
 1799 [BBCGI19] into a distributed zero-knowledge proof in which the communication complexity is dom-
 1800 inated by the total length of the *proofs* produced by the IT prover (i.e., π to be linear queried), and
 1801 the *answers* to the linear queries. The compilers for distributed zero knowledge can be information-
 1802 theoretic, and do not take advantage of the additional structure offered by polynomial IOPs or the
 1803 low algebraic degree of the verifier’s decision predicate. Note that, unlike for fully linear IOPs, in a
 1804 plain LIOP the dominant communication costs are unlikely to be the proof lengths of the IT prover
 1805 (e.g., the proof length does not influence communication, since proofs could be “compressed” by
 1806 hashing or homomorphic encryption).

1807 2.3.5.2 CC for PIOP: Polynomial Commitments

1808 The key to better cryptographic compilers is the following additional feature of the GKR-based
 1809 LIOP: Each of the proof vectors π_i can be viewed as defining the coefficients of a *polynomial* in a

1810 small number of variables, where each linear query to π_i evaluates the polynomial at a single point.
 1811 More concretely, the long proof vector in the GKR-based LIOP defines a multilinear polynomial in
 1812 $\log m$ variables that encodes (x, w) .

1813 This can be captured by a more refined notion of LIOP in which a proof is interpreted as the coef-
 1814 ficient vector of a polynomial of bounded degree, typically in a small (at most logarithmic) number
 1815 of variables, and a query is interpreted as an evaluation point. This refinement of LIOP, referred
 1816 to as *polynomial IOP* (PIOP) [BFS20], is motivated by possibility of cryptographic compilers that
 1817 take advantage of the extra structure. (Another name for a closely related model is “Algebraic
 1818 Holographic Proof” [CHMMVW20].) Cryptographic compilers for PIOP rely on efficient realiza-
 1819 tions of *polynomial commitment*, a functional commitment scheme that allows the prover to commit
 1820 to a polynomial in a way that supports an efficient proof of evaluation on a given point.

1821 The first systems that combined the GKR protocol with polynomial commitments were Hyrax
 1822 [WTTW18] and (zk)-vSQL [ZGKPP17]. Hyrax used a transparent implementation of polynomial
 1823 commitment based on discrete logarithm at the cost of proof size $\approx d + \sqrt{m}$. The vSQL system could
 1824 eliminate the \sqrt{m} additive term by using a variant of a widely used polynomial commitment scheme
 1825 [KZG10] that relies on a pairing-friendly group and requires a trusted setup. The subsequent Libra
 1826 system [XZZPS19] combined a similar polynomial commitment with an improved zero-knowledge
 1827 variant of the GKR-based PIOP. Virgo [ZXZS20] is a variant of Libra that uses an efficient trans-
 1828 parent polynomial commitment based only on symmetric cryptography, combining an IOP based
 1829 on the FRI protocol with a GKR-based LIOP with $m = O(\log s)$. A similar kind of polynomial
 1830 commitment was proposed in RedShift [KPV19]. Another proposal [BFS20] achieves a transparent
 1831 polynomial commitment scheme with much better succinctness, using groups of an unknown order,
 1832 though coming at the cost of high concrete prover complexity and not being post-quantum secure.
 1833 A subsequent proposal [Lee20] gave a related polynomial commitment scheme that avoids the use
 1834 of class groups (it is based on the symmetric external Diffie Hellman assumption) and mitigates
 1835 the high prover costs. Additional works combining PIOPs with polynomial commitments to obtain
 1836 succinct arguments include the following: Kopsis and Xiphos [SL20] combines the PIOP for R1CS
 1837 implicit in Spartan [Set20] with the polynomial commitment Dory [Lee20] and variants; Cerebrus
 1838 [LSTW21] distills a polynomial commitment scheme from an IOP given in Ligerio [AHIV17] and
 1839 combines it with the same PIOP from Spartan [Set20].

1840 2.3.6 CC for ILC

1841 Using a collision-resistant hash function, an ILC proof system can be compiled into a public-coin
 1842 argument in the plain model with communication complexity roughly $\sum_i (n_i + m_i)$ [BCGGHJ17].

1843 2.4 Specialized ZK Proofs

1844 This section considers ZKPs for special purpose languages, i.e., proofs that do not cover NP. Exam-
 1845 ples include proving knowledge of a discrete logarithm or the factorization of an RSA modulus N .

1846 **Contribution wanted**

1847 2.5 Proof Composition

1848 With recursive composition one can prove there exists a proof of a proof of a proof, etc. This can
 1849 have multiple advantages, ranging from improved efficiency, to improved functionality, to the ability
 1850 to prove machine computations [BCTV14a]. For example, using a recursive zero-knowledge proof it
 1851 is theoretically possible to prove that a `while` loop of an unspecified size in a machine computation
 1852 is satisfied, and furthermore to update the proof incrementally as the computation proceeds — the
 1853 notion of *incrementally-verifiable computation* [Val08]. Recursive proofs can also prove integrity of
 1854 ongoing distributed computation — the notion of *proof-carrying data* [CT10; CTV15] — to assure
 1855 intermediate nodes, as well as a final verifier, that all preceding parts of the computation (done by
 1856 other parties) are correct.

1857 **FF2.12. Proposed future figure (illustration, diagram, ...):**

1858 Illustration of recursive ZK proving

1859 In the realm of blockchains, these techniques found several applications. First, validity of the
 1860 current state of the blockchain can be succinctly proven by considering it as an incrementally
 1861 verifiable computation, so that any newcomer to the system can be certain that the state is correct
 1862 just by checking one recursive proof, as in Mina (nee Coda) [BMRS20]. Second, even within a single
 1863 transaction, multiple proofs can be recursively composed to achieve succinctness (in transaction
 1864 size and verification time), as in Zcash Orchard [HGBNL21].

1865 Approaches to build recursive proofs include recursive composition of zk-SNARKs [Val08; CT10;
 1866 BCCT13; BCTV14a], and its generalization to accumulation / split-accumulation schemes [BGH19;
 1867 BCMS20; COS20; BDFG20].

1868 **Contribution wanted:** more details on construction approaches and additional application

1869 2.6 Interactivity

1870 Several of the proof systems described in Section 2.2 are interactive, including classical interactive
 1871 proofs (IPs), IOPs, and linear IOPs. This means that the verifier sends multiple challenge messages
 1872 to the prover, with the prover replying to challenge i before receiving challenge $i + 1$; soundness
 1873 relies on the prover being unable to predict challenge $i + 1$ when it responds to challenge i . The
 1874 other proof systems are non-interactive, namely classical PCPs and linear PCPs. All of these proof
 1875 systems can be combined with cryptographic compilers (see Section 2.3) to yield argument systems
 1876 that may or may not be interactive, depending on the compiler.

1877 2.6.1 Advantages of Interactive Proof and Argument Systems

1878 1. **Efficiency and Simplicity.** Interactive proof systems can be simpler or more efficient
 1879 than non-interactive ones. As an example, researchers introduced the IOP model [BCS16;
 1880 RRR16], which is interactive, in part because interactivity allowed for circumventing efficiency

1881 bottlenecks arising in state of the art PCP constructions [BCGT13]. As another example,
 1882 some argument systems derived from IPs [WTTW18; XZZPS19] have substantially better
 1883 space complexity for the prover (a key scalability bottleneck) than state of the art PCPs
 1884 [BCGT13] or linear PCPs [GGPR13; Gro16].

1885 Yet, if an interactive protocol is public coin, it can be rendered non-interactive and publicly
 1886 verifiable in most settings via the Fiat-Shamir transformation (see Section 2.3), often with
 1887 little loss in efficiency. This means that protocol designers have the freedom to leverage
 1888 interactivity as a “resource” to simplify protocol design, improve efficiency, weaken or remove
 1889 trusted setup, etc., and still have the option of obtaining a non-interactive argument using
 1890 the Fiat-Shamir transformation.

1891 (Applying the Fiat-Shamir heuristic to an interactive protocol to obtain a non-interactive
 1892 argument may increase soundness error, and may transform statistical security into compu-
 1893 tational security — see Section 1.8.3. However, recent works [BCS16; CCHLRRW19] show
 1894 that when the transformation is applied to specific IP, IOP, and linear IOP protocols of both
 1895 practical and theoretical interest, the blowup in soundness error is only polynomial in the
 1896 number of rounds of interaction.)

1897 2. **Setup.** Cryptographic compilers for linear PCPs currently require a structured reference
 1898 string (SRS) (see Section 3.6.2). Here, an SRS is a structured string that must be generated
 1899 by a trusted third party during a setup phase, and soundness requires that any trapdoor used
 1900 during this trusted setup must not be revealed. In contrast, some compilers that apply to IPs,
 1901 IOPs (as well as PCPs), and linear IPs yields arguments in which the prover and the verifier
 1902 need only access a uniform random string (URS), which can be obtained from a common
 1903 source of randomness. Such a setup is referred as *transparent* setup in the literature.

1904 3. **Cryptographic Primitives.** Argument systems derived from IPs, IOPs, or linear IOPs also
 1905 sometimes rely on more desirable cryptographic primitives. For example, IPs themselves are
 1906 information-theoretically secure, not relying on cryptographic assumptions. And in contrast
 1907 to arguments derived from linear PCPs, those derived from IOPs rely only on symmetric-key
 1908 cryptographic primitives (e.g., [BCS16]). It is also known how to obtain succinct *interactive*
 1909 arguments in the plain model based on falsifiable assumptions like collision-resistant hash
 1910 families [Kil95], but this is not the case for succinct *non-interactive* arguments.

1911 4. **Non-transferability.** In some applications, it is essential that proofs be deniable or *non-*
 1912 *transferable* (i.e., that a verifier cannot convince a third party of the validity of the statement
 1913 — see Section 1.6.6). While these properties are not unique to interactive protocols, interac-
 1914 tion offers a natural way to make proofs non-transferable (for details, see Section 2.6.3).

1915 5. **Interactivity May Limit Adversaries’ Abilities.** Interactive protocols can potentially
 1916 be run with fewer bits of security and hence be more efficient. For example, interactive
 1917 settings may allow for the enforcement of a time limit for the protocol to terminate, limiting
 1918 the runtime of attackers. Alternatively, in an interactive setting it may be possible to ensure
 1919 that adversaries only have one attempt to attack a protocol, while this will not be possible
 1920 in many non-interactive settings. See Section 1.8.2 for details.

1921 6. **Interactivity May Be Inherent to Applications.** Many applications are inherently
 1922 interactive. For example, real-world networking protocols involve multiple messages just to
 1923 initiate a connection. In addition, zero-knowledge protocols are often combined with other
 1924 cryptographic primitives in applications (e.g., oblivious transfer). If the other primitives are
 1925 interactive, then the final cryptographic protocol will be interactive regardless of whether the

1926 zero-knowledge protocol is non-interactive. If an application is inherently interactive, it may
 1927 be reasonable to leverage the interaction as a resource if it can render a protocol simpler,
 1928 more efficient, etc.

1929 2.6.2 Disadvantages of Interactive Proof and Argument Systems

1930 1. **Interactive protocols must occur online.** In an interactive protocol, the proof cannot
 1931 simply be published or posted and checked later at the verifier’s convenience, as can be done
 1932 with non-interactive protocols.

1933 2. **Public Verifiability.** Many applications require that proofs be verifiable by any party at
 1934 any time. Public verifiability may be difficult to achieve for interactive protocols. This is
 1935 because soundness of interactive protocols relies on the prover being unable to predict the
 1936 next challenge it will receive in the protocol. Unless there is a publicly trusted source of
 1937 unpredictable randomness (e.g., a randomness beacon) and a way for provers to timestamp
 1938 messages, it is not clear how any party other than the one sending the challenges can be
 1939 convinced that the challenges were properly generated, and the prover replied to challenge i
 1940 before learning challenge $i + 1$. See Section 2.6.3 below for further details.

1941 3. **Network latency can make interactive protocols slow.** If an interactive protocol
 1942 consists of many messages sent over a network, network latency may contribute significantly
 1943 to the total execution time of the protocol.

1944 4. **Timing or Side Channel Attacks.** Because interactive protocols require the prover to
 1945 send multiple messages, there may be more vulnerability to side channel or timing attacks
 1946 compared to non-interactive protocols. Timing attacks will only affect zero-knowledge, not
 1947 soundness, for public-coin protocols, because the verifier’s messages are simply random coins,
 1948 and timing attacks should not leak information to the prover in this case. In private coin
 1949 protocols, both zero-knowledge and soundness may be affected by these attacks.

1950 5. **Concurrent Security.** If an interactive protocol is not used in isolation, but is instead used
 1951 in an environment where multiple interactive protocols may be executed concurrently, then
 1952 considerable care should be taken to ensure that the protocol remains secure. See for example
 1953 [Gol13, Section 2.1] and the references therein. Issues of concurrent execution security are
 1954 greatly mitigated for non-interactive protocols [GOS06].

1955 6. **Proof Length.** Currently, the zero-knowledge protocols with the shortest known proofs
 1956 are based on [linear PCPs](#), which are non-interactive. These proofs are just a few group
 1957 elements. While (public-coin) zero-knowledge protocols based on IPs or IOPs can be ren-
 1958 dered non-interactive with the Fiat-Shamir heuristic, they currently produce longer proofs.
 1959 The longer proofs may render these protocols unsuitable for some applications (e.g., pub-
 1960 lic blockchain), but they may still be suitable for other applications (even related ones, like
 1961 enterprise blockchain applications).

1962 2.6.3 Nuances on transferability vs. interactivity

1963 The relation between interactivity and transferability/deniability is not without nuances. The
 1964 following paragraphs show several possible combinations.

1965 **Non-interactive and deniable.** A non-interactive ZKP may be non-transferable. This may be
 1966 based for example on a setup assumption such as a local CRS that is itself deniable. In that case,
 1967 a malicious verifier cannot prove to an external party that the CRS was the one used in a real
 1968 protocol execution, leading the external party to have reasonable suspicion that the verifier may
 1969 have simulated the CRS so as to become able to simulate a protocol execution transcript, without
 1970 actual participation of a legitimate prover. Another example of non-transferability is when a ZKP
 1971 intended to prove (i) an assertion (of membership or knowledge) actually proves its disjunction
 1972 with (ii) the knowledge of the secret key of a designated verifier, for example assuming a public key
 1973 infrastructure (PKI). This suffices to convince the original verifier the initial statement (i) is true,
 1974 since the verifier knows that the prover does not actually know the secret key (ii). In other words,
 1975 a success in the interactive proof stems from the initial assertion (i) being truthful. However, for
 1976 any external party, the transcript of the proof may conceivably have been produced by the original
 1977 designated verifier, who can simply do it with the knowledge of the secret key (ii). In that sense,
 1978 the designated verifier would be unable to convince others that the transcript of a legitimate proof
 1979 was not simulated by the verifier.

1980 **Non-interactive and transferable.** If transferability is intended as a feature, then a non-interactive
 1981 protocol can be achieved for example with a public (undeniable) CRS. For example, if a CRS is
 1982 generated by a trusted randomness beacon, and if soundness follows from the inability of the prover
 1983 to control the CRS, then any external party (even one not involved with the prover at the time of
 1984 proof generation) can at a later time verify that a proof transcript could have only been generated
 1985 by a legitimate prover.

1986 **Interactive and deniable.** A classical example (in a standalone setting, without concurrent execu-
 1987 tions) for obtaining the deniability property comes from interactive ZKP protocols proven secure
 1988 based on the use of rewinding. Here, deniability follows from the simulatability of transcripts for
 1989 any malicious verifier. For each interactive step, the simulator learns the challenge issued by the
 1990 possibly malicious verifier, and then rewinds to reselect the preceding message of the prover, so as
 1991 to be able to answer the subsequent challenge. Some techniques require the use of commitments
 1992 and/or trapdoors, and may enable this property even for straight-line simulation (i.e., without
 1993 rewinding), provided there is an appropriate trusted setup.

1994 **Interactive and transferable.** In certain settings it is possible, even from an interactive ZKP proto-
 1995 col execution, to produce a transcript that constitutes a transferable proof. Usually, transferability
 1996 can be achieved when the (possibly malicious) verifier can convincingly show to external parties that
 1997 the challenges selected during a protocol execution were unpredictable at the time of the determi-
 1998 nation of the preceding messages of the prover. The transferable proof transcript is then composed
 1999 of the messages sent by the prover and additional information from the internal state of a malicious
 2000 verifier, including details about the generation of challenges. For example, a challenge produced (by
 2001 the verifier) as a cryptographic hash output (or as a keyed pseudo-random function) of the previous
 2002 messages may later be used to provide assurance that only a legitimate prover would have been
 2003 able to generate a valid subsequent message (response). As another example, if the interactive ZKP
 2004 protocol is composed with a communication protocol where the prover authenticates all sent mes-
 2005 sages (e.g., signed within a PKI, and timestamped by a trusted service), then the overall sequence
 2006 of those certified messages becomes, in the hands of the verifier, a transferable proof. Furthermore,
 2007 from a transferable transcript, the actual transfer can also be performed in an interactive way: the
 2008 verifier (in possession of the transcript) acts as prover in a transferable ZKP of knowledge of a

2009 transferable transcript, thereby transferring to the external verifier a new transferable transcript.

2010 **2.6.3.1 (Non)-Transferability/Deniability of Zero-Knowledge Proofs**

2011 **Off-line non-transferability (deniability) of ZK proofs.** Zero-knowledge proofs are in general
 2012 interactive. Interaction is inherent without a setup. Indeed, Goldreich and Oren showed that for
 2013 non-trivial languages zero-knowledge proofs require at least 3 rounds.

2014 With absence setup, the ZP property guarantees a property called off-line non-transferability, also
 2015 known as deniability — note that a verifier could always compute an equivalent transcript by
 2016 running the simulator. This means the verifier gets no evidence of having received an accepting
 2017 proof from a prover and thus has no advantage in transferring the received proof to others.

2018 **On-line non-transferability of ZK proofs.** The situation is more complicated in case of on-line
 2019 non-transferability. Indeed, in this case a malicious verifier plays with a honest prover in a zero-
 2020 knowledge proof system and at the same time the malicious verifier plays with others in the attempt
 2021 of transferring the proof that he his receiving from the prover. Non-transferability is therefore a form
 2022 of security against man-in-the-middle attacks. Security against such attacks is typically referred to
 2023 as non-malleability when the same zero-knowledge proof system is used by the adversary to try to
 2024 transfer the proof to a honest verifier. When instead different protocols are involved as part of the
 2025 activities of the adversary, some stronger notions are required to model security under such attacks
 2026 (e.g., universal composability).

2027 **Transferability of a NIZK proof: publicly verifiable ZK.** The transferability of a zero-knowledge
 2028 proof could become unavoidable when some forms of setups are considered and the zero-knowledge
 2029 proof makes some crucial use of it. Indeed, notice that both in the common reference string
 2030 model and in the programmable random oracle model one can construct non-interactive zero-
 2031 knowledge proofs. Such proofs cannot be simulated by the verifier with the same setup or the same
 2032 instantiation of the random oracle. More specifically, non-interactive zero-knowledge proofs are
 2033 constructed without the contribution of any verifier, therefore they are publicly verifiable proofs
 2034 that can naturally be transferred among verifiers.

2035 **Designated-verifier NIZK proofs.** With more sophisticated setups other options become possible.
 2036 Consider for instance a verifier possessing a public identity implemented through a public key. In
 2037 this case the prover can compute a non-interactive zero-knowledge proof that makes crucially use
 2038 of the public key of the verifier at the point that the verifier using the corresponding secret key
 2039 could compute an indistinguishable proof. In this case we have that the proof is a non-interactive
 2040 designated-verifier zero-knowledge proof and is non-transferable since the verifier that receives the
 2041 proof could have computed an equivalent proof by herself, therefore there is no evidence to share
 2042 with others about the fact that the proof comes from a honest prover.

2043 **Transferability of interactive ZK proofs.** The use of identities implemented through public keys
 2044 can also have impact in the interactive case. For example, one can design an interactive ZKP
 2045 system with a transferability flavor. If the prover signs the transcript, then the verifier is able to
 2046 transfer the “proof” to whoever believes that the original prover was honest.

2047

Chapter 3. Implementation

2048

3.1 Overview

2049

2050

2051

2052

2053

2054

2055

2056

2057

2058

2059

2060

2061

2062

By having a standard or framework around the implementation of ZKPs, we aim to help platforms adapt more easily to new constructions and new schemes, that may be more suitable because of efficiency, security or application-specific changes. Application developers and the designers of new proof systems all want to understand the performance and security tradeoffs of different ZKP constructions when invoked in various applications. This track focuses on building a standard interface that application developers can use to interact with ZKP proof systems, in an effort to improve facilitate interoperability, flexibility and performance comparison. In this first effort to achieve such an interface, our focus is on non-interactive proof systems (NIZKs) for general statements (NP) that use an R1CS/QAP-style constraint system representation. This includes many, though not all, of the practical general-purpose ZKP schemes currently deployed. While this focus allows us to define concrete formats for interoperability, we recognize that additional constraint system representation styles (e.g., arithmetic and Boolean circuits) are in use, and are within scope of the ongoing effort. We also aim to establish best practices for the deployment of these proof systems in production software.

2063

FF3.1. Proposed future figure (illustration, diagram, ...):

2064

2065

Add generic figure of frontend vs backend vs IR paradigm for implementation systems with arrows showing the flow of the data / input / output

2066

3.1.1 What this chapter is NOT about:

2067

2068

2069

2070

- A unique explanation of how to build ZKP applications
- An exhaustive list of the security requirements needed to build a ZKP system
- A comparison of front-end tools
- A show of preference for some use-cases or others

2071

3.2 Backends: Cryptographic System Implementations

2072

2073

2074

2075

2076

The backend of a ZK proof implementation is the portion of the software that contains an implementation of the low-level cryptographic protocol. It proves statements where the instance and witness are expressed as variable assignments, and relations are expressed via low-level languages (such as arithmetic circuits, Boolean circuits, R1CS/QAP constraint systems or arithmetic constraint satisfaction problems).

FF3.2. Proposed future figure (illustration, diagram, ...):

add a figure with the different components of a backend in a “table hierarchy”

The backend typically consists of a concrete implementation of a proposed ZK proof system(s) (often specified by pseudocode in a paper publication), along with supporting code for the required arithmetic operations, serialization formats, tests, benchmarking, etc. There are numerous backends, most originated as academic research prototypes and available as open-source projects. The offerings and features of backends evolve rapidly, so it can be useful to consult a curated taxonomy [ZKP.Science].

Considerations for the choice of backends include:

- ZK proof system(s) implemented by the backend, and their associated security, assumptions and asymptotic performance (as discussed in the Security Track document)
- Concrete performance (see Benchmarks section)
- Programming language and API style (this consideration may be satisfied by adherence to prospective ZK proof standards; see the the API and File Formats section)
- Platform support
- Availability as open source
- Active community of maintainers and users
- Correctness and robustness of the implementation (as determined, e.g., by auditing and formal verification)
- Applications (as evidence of usability and scrutiny).

3.3 Frontends: Constraint-System Construction

The frontend of a ZK proof system implementation provides means to express statements in a convenient language and to prove such statements in zero knowledge by compiling them into a low-level representation and invoking a suitable ZK backend.

A frontend consists of:

- The specification of a high-level language for expressing statements.
- A compiler that converts relations expressed in high-level language into low-level relations suitable for some backend(s). For example, this may produce an R1CS constraint system.
- Instance reduction: conversion of the instance in a high-level statement to a low-level instance (e.g., assignment to R1CS instance variables).
- Witness reduction: conversion of the witness to a high-level statement to a low-level witness (e.g., assignment to witness variables).
- Typically, a library of “gadgets” with useful and hand-optimized building blocks for statements.

FF3.3. Proposed future figure (illustration, diagram, ...):

add figure that expresses example / abstract circuit that shows what inputs are private / public (witness vs instance) and the way that gadgets can be composed

Languages for expressing statements, which have been implemented in frontends to date include: code library for general-purpose languages, domain-specific language, suitably-adapted general-purpose high-level language, and assembly language for a virtual CPU.

Frontends' compilers, as well as gadget libraries, often implement optimizations aiming at reducing the cost of the constraint systems (e.g., the number of constraints and variables). This includes techniques such as making use of “free linear combinations” in R1CS, using nondeterministic advice given in witness variables (e.g., for integer arithmetic or random-access memory), removing redundancies, using cryptographic schemes tailored for the given algebraic settings (e.g., Pedersen hashing on the Jubjub curve or MiMC for hash functions, RSA verification for digital signatures), and many other techniques. For example, see further discussion in the Zcon0 Circuit Optimisation handout [Hop18].

There are many implemented frontends, including some that provide alternative ways to invoke the same underlying backends. Most have originated as academic research prototypes, and are available as open-source projects. The offerings and features of frontends evolve rapidly, so it can be useful to consult a curated taxonomy [ZKP.Science].

3.4 APIs and File Formats

Our primary goal is to improve interoperability between proving systems and frontend consumers of proving system implementations. We focused on two approaches for building standard interfaces for implementations:

1. We aim to develop a common API for proving systems to expose their capabilities to frontends in a way that is maximally agnostic to the underlying implementation details.
2. We aim to develop a file format for encoding a popular form of constraint systems (namely R1CS), and its assignments, so that proving system implementations and frontends can interact across language and API barriers.

We did not aim to develop standards for interoperability between backends implementing the same (abstract) scheme, such as serialization formats for proofs (see the Extended Constraint-System Interoperability section for further discussion).

3.4.1 Generic API

In order to help compare the performance and usability tradeoffs of proving system implementations, frontend application developers may wish to interact with the underlying proof systems via a generic interface, so that proving systems can be swapped out and the tradeoffs observed in practice. This also helps in an academic pursuit of analysis and comparison.

Figure 3.1 depicts an abstraction of the parties and objects involved a NIZK.

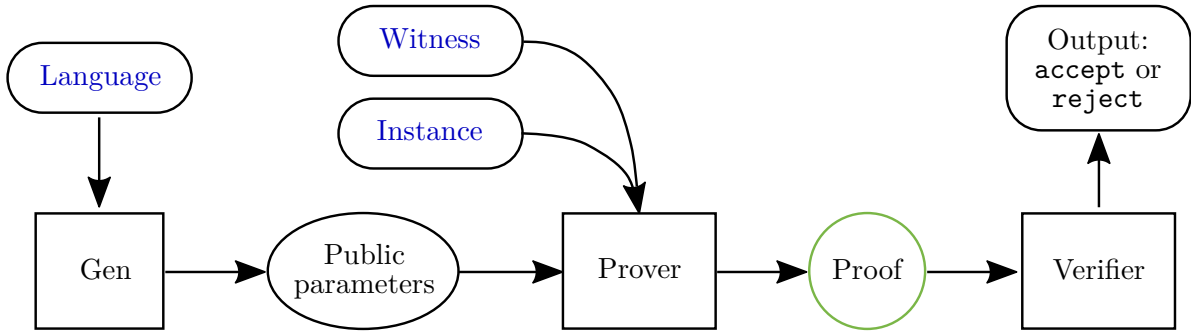


Figure 3.1. Parties and objects in a NIZK

We did not complete a generic API design for proving systems, but we did survey numerous tradeoffs and design approaches for such an API that may be of future value.

We separate the APIs and interfaces between the universal and non-universal NIZK setting. In the universal setting, the NIZK’s CRS generation is independent of the relation (i.e., one CRS enables proving any NP statement). In the non-universal settings, the CRS generation depends on the relation (represented as a constraint system), and a given CRS enables proving the statements corresponding to any instance with respect to the specific relation.

Table 3.1: APIs and interfaces by types of universality and preprocessing

	Preprocessing (GENERATE has superpolylogarithmic runtime / output size as function of constraint system size)	Non-preprocessing (GENERATE runtime and output size is fast and CRS is at most polylogarithmic in constraint system size)
Non-universal (GENERATE needs constraint system as input)	QAP-based [PHGR13], [GGPR13], [BCGTV13]	?
Universal (GENERATE needs just a size bound)	vnTinyRAM, vRAM, Bulletproofs (with explicit CRH)	Bulletproofs (with PRG-based CRH generation)
Universal and scalable (GENERATE needs nothing but security parameter)	(impossible)	“Fully scalable” SNARKs based on PCD (recursive composition)

Proving systems may need to use a generic interface to express several capabilities:

1. The creation of CRS objects in the form of proving and verifying parameters, given the input language or size bound.
2. The serialization of CRS objects into concrete encodings.

- 2173 3. Metadata about the proving system such as the size and characteristic of the field (for arith-
2174 metic constraints).
- 2175 4. Witness objects containing private inputs known only to the prover, and Instance objects
2176 containing public inputs known to the prover and verifier.
- 2177 5. The creation of Proof objects when supplied proving parameters, an Instance, and a Witness.
- 2178 6. The verification of Proof objects given verifying parameters and an Instance.

2179 **Future work:** We would like to see a concrete API design which leverages our tentative model,
2180 with additional work to encode concepts such as recursive composition and the batching of proving
2181 and verification operations.

2182 3.4.2 R1CS File Format

2183 There are many frontends for constructing [constraint systems](#), and many backends that consume
2184 constraint systems (and variable assignments) to create or verify proofs. It can thus be useful to
2185 have a file format [\[ZKP-FF\]](#) that both frontends and backends can use to communicate constraint
2186 systems and variable assignments. Goals include simplicity, ease of implementation, compactness
2187 and avoiding hard-coded limits.

2188 A popular constraint system for ZKPs is the R1CS (Rank-1 Constraint System) — an NP-complete
2189 language for specifying relations as a system of bilinear constraints (i.e., a rank 1 quadratic constraint
2190 system) [\[BCGTV13, Appendix E in extended version\]](#). This is a more intuitive reformulation of QAP
2191 *QAP (Quadratic Arithmetic Program)* [\[PHGR13\]](#). R1CS is the native constraint system language
2192 of many ZK proof constructions, including many ZK proof applications in operational deployment.

2193 **FF3.4. Proposed future figure (illustration, diagram, ...):**

2194 Exemplify a R1CS constraint system (visualize a matrix, the QAP polynomials, etc...)

2195 The remainder of this section proposes a format making heavy use of variable-length integers,
2196 which are prevalent in the (space-efficient) encoding of an R1CS. `VarInt` denotes a variable-length
2197 unsigned integer, and `SignedVarInt` denotes a variable-length signed integer. `VarInt` is typically
2198 used for lengths or version numbers, and `SignedVarInt` for field element constants. The actual
2199 description of a `VarInt` is not yet specified.

2200 We'll be working with primitive variable indices of the following form:

2201 `ConstantVar` \leftarrow `SignedVarInt(0)`
2202 `InstanceVar(i)` \leftarrow `SignedVarInt(-(i + 1))`
2203 `WitnessVar(i)` \leftarrow `SignedVarInt(i + 1)`
2204 `VariableIndex` \leftarrow `ConstantVar` / `InstanceVar(i)` / `WitnessVar(i)`

2205 *ConstantVar* represents an indexed constant in the field, usually assigned to one. *InstanceVar*
2206 represents an indexed variable of the instance, or the public input, serialized with negative indices.
2207 *WitnessVar* represents an indexed variable of the witness, or the private/auxiliary input, serialized
2208 with positive indices. *VariableIndex* represents one of any of these possible variable indices.

2209 We'll also be working with primitive expressions of the following form:

```
2210 Coefficient ← SignedVarInt
2211 Sequence(Entry) ← | length: VarInt | length * Entry |
2212 LinearCombination ← Sequence(| VariableIndex | Coefficient |)
```

- 2213 • Coefficients must be non-zero.
- 2214 • Entries should be sorted by type, then by index:
 - 2215 – | ConstantVar | sorted(InstanceVar) | sorted(WitnessVar) |

```
2216 Constraint ←
2217 | A: LinearCombination | B: LinearCombination | C: LinearCombination |
```

2218 We represent a *Coefficient* (a constant in a linear combination) with a *SignedVarInt*. (TODO: there
 2219 is no constraint on its canonical form.) These should never be zero. We express a *LinearCombi-*
 2220 *nation* as sequences of *VariableIndex* and *Coefficient* pairs. Linear combinations should be sorted
 2221 by type and then by index of the *VariableIndex*; i.e., *ConstantVar* should appear first, *InstanceVar*
 2222 should appear second (ascending) and *WitnessVar* should appear last (ascending).

2223 We express constraints as three *LinearCombination* objects A, B, C, where the encoded constraint
 2224 represents $A * B = C$.

2225 The file format will contain a header with details about the constraint system that are important
 2226 for the backend implementation or for parsing.

```
2227 Header(version, vals) ←
2228 | version: VarInt | vals: Sequence(SignedVarInt) |
```

2229 The *vals* component of the *Header* will contain information such as:

- 2230 • P ← Field characteristic
- 2231 • D ← Degree of extension
- 2232 • N_X ← Number of instance variables
- 2233 • N_W ← Number of witness variables

2234 The representation of elements of extension fields is not currently specified, so D should be 1.

2235 The file format contains a magic byte sequence “R1CSstmt”, a header, and a sequence of constraints,
 2236 as follows:

```
2237 R1CSFile ← | ``R1CSstmt' | Header(0, [ P, D, N_X, N_W, ... ]) | Sequence(Constraint) |
```

2238 Further values in the header are undefined in this specification for version 0, and should be ignored.
 2239 The file extension “r1cs” is used for R1CS circuits.

2240 **Further work:** We wish to have a format for expressing the assignments for use by the backend
 2241 in generating the proof. We reserve the magic “R1CSasig” and the file extension “assignments”
 2242 for this purpose. We also wish to have a format for expressing symbol tables for debugging. We
 2243 reserve the magic “R1CSsymb” and the file extension “r1cssym” for this purpose.

2244 In the future we also wish to specify other kinds of constraint systems and languages that some

2245 proving systems can more naturally consume.

2246 3.5 Benchmarks

2247 As the variety of zero-knowledge proof systems and the complexity of applications has grown, it
 2248 has become more and more difficult for users to understand which proof system is the best for their
 2249 application. Part of the reason is that the tradeoff space is high-dimensional. Another reason is
 2250 the lack of good, unified benchmarking guidelines. We aim to define benchmarking procedures that
 2251 both allow fair and unbiased comparisons to prior work and also aim to give enough freedom such
 2252 that scientists are incentivized to explore the whole tradeoff space and set nuanced benchmarks in
 2253 new scenarios and thus enable more applications.

2254 The benchmark standardisation is meant to document best practices, not hard requirements. They
 2255 are especially recommended for new general-purpose proof systems as well as implementations
 2256 of existing schemes. Additionally the long-term goal is to enable independent benchmarking on
 2257 standardized hardware.

2258 3.5.1 What metrics and components to measure

2259 We recommend that as the primary metrics the **running time (single-threaded)** and the **com-**
 2260 **munication complexity** (proof size, in the case of non-interactive proof systems) of all compo-
 2261 nents should be measured and reported for any benchmark. The measured components should
 2262 at least include the **prover** and the **verifier**. If the setup is significant then this should also be
 2263 measured, further system components like parameter loading and number of rounds (for interactive
 2264 proof systems) are suggested.

2265 The following metrics are additionally suggested:

- 2266 • Parallelizability
- 2267 • Batching
- 2268 • Memory consumption (either as a precise measurement or as an upper bound)
- 2269 • Number of operations (e.g., field operations, multi-exponentiations, FFTs) and their sizes
- 2270 • Disk usage/Storage requirement
- 2271 • Crossover point: point where verifying is faster than running the computation
- 2272 • Largest instance that can be handled on a given system
- 2273 • Witness generation (this depends on the higher-level compiler and application)
- 2274 • Tradeoffs between any of the metrics.

2275 3.5.2 How to run the benchmarks

2276 Benchmarks can be both of analytical and computational nature. Depending on the system either
 2277 may be more appropriate or they can supplement each other. An analytical benchmark consists of

2278 asymptotic analysis as well as concrete formulas for certain metrics (e.g., the proof size). Ideally
 2279 analytical benchmarks are parameterized by a security level or otherwise they should report the
 2280 security level for which the benchmark is done, along with the assumptions that are being used.

2281 Computational benchmarks should be run on a consistent and commercially available machine.
 2282 The use of cloud providers is encouraged, as this allows for cheap reproducibility. The machine
 2283 specification should be reported along with additional restrictions that are put on it (e.g., throt-
 2284 tling, number of threads, memory supplied). Benchmarking machines should generally fall into
 2285 one of the following categories and the machine description should indicate the category. If the
 2286 software implementation makes certain architectural assumptions (such as use of special hardware
 2287 instructions) then this should be clearly indicated.

- 2288 • Battery powered mobile devices
- 2289 • Personal computers such as laptops
- 2290 • Server style machines with many cores and large memories
- 2291 • Server clusters using multiple machines
- 2292 • Custom hardware (should not be used to compare to software implementations)

2293 We recommend that most runs are executed on a single-threaded machine, with parallelizability
 2294 being an optional metric to measure. The benchmarks should be obtained preferably for more than
 2295 one security level, following the recommendations stated in Sections 1.8.2 and 1.8.3.

2296 In order to enable better comparisons we recommend that the metrics of other proof systems/
 2297 implementations are also run on the same machine and reported. The onus is on the library
 2298 developer to provide a simple way to run any instance on which a benchmark is reported. This
 2299 will additionally aid the reproducibility of results. Links to implementations will be gathered at
 2300 `zkp.science` and library developers are encouraged to ensure that their library is properly referenced.
 2301 Further we encourage scientific publishing venues to require the submission of source code if an
 2302 implementation is reported. Ideally these venues even test the reproducibility and indicate whether
 2303 results could be reproduced.

2304 3.5.3 What benchmarks to run

2305 We propose a set of benchmarks that is informed by current applications of zero-knowledge proofs,
 2306 as well as by differences in proving systems. This list in no way complete and should be amended
 2307 and updated as new applications emerge and new systems with novel properties are developed.
 2308 Zero-knowledge proof systems can be used in a black-box manner on an existing application, but
 2309 often designing the application with a proof system in mind can yield large efficiency gains. To
 2310 cover both scenarios we suggest a set of benchmarks that include commonly used primitives (e.g.,
 2311 SHA-256) and one where only the functionality is specified but not the primitives (e.g. a collision-
 2312 resistant hash function at 128-bit classical security).

FF3.5. Proposed future figure (illustration, diagram, ...):

add figure / mindmap of the different layers / levels of use-cases to be run for benchmarking
 (eg: see benchmarks proposal paper in zkproof3)

2316 **Commonly used primitives.**

2317 The following is a list of primitives that can serve as microbenchmarks and are also of independent
 2318 interest. Library developers are free to choose how their library runs a given primitive, but the
 2319 process can be aided by making having available circuit descriptions in commonly used file formats
 2320 (e.g., R1CS).

2321 • **Recommended:**

- 2322 1. SHA-256
- 2323 2. AES
- 2324 3. A simple vector or matrix product at different sizes

2325 • **Further suggestions:**

- 2326 1. Zcash Sapling “spend” relation
- 2327 2. RC4 (for RAM memory access)
- 2328 3. Scrypt
- 2329 4. TinyRAM running for n steps with memory size s
- 2330 5. Number theoretic transform (coefficients to points): small fields; big fields; pattern
 2331 matching.

2332 • **Repetition:**

- 2333 1. The above relations, parallelized by putting n copies in parallel.

2334 **Functionalities.**

2335 The following are examples of cryptographic functionalities that are especially interesting to ap-
 2336 plication developers. The realization of the primitive may be secondary, as long as it achieves the
 2337 security properties. It is helpful to provide benchmarks for a constraint-system implementation of
 2338 a realization of these primitives that is tailored for the NIZK backend.

2339 In all of the following, the primitive underlying the ZKP statement should be given at a level of
 2340 128 bits or higher and match the security of the NIZK proof system.

2341 • Asymmetric cryptography

- 2342 – Signature verification
- 2343 – Public key encryption
- 2344 – Diffie Hellman key exchange over any group with 128 bit security

2345 • Symmetric & Hash

- 2346 – Collision-resistant hash function on a 1024-byte message
- 2347 – Set membership in a set of size 2^{20} (e.g., using Merkle authentication tree)
- 2348 – MAC
- 2349 – AEAD

- 2350 • The scheme’s own verification circuit, with matching parameters, for recursive composition
 2351 (Proof-Carrying Data)

- 2352 • Range proofs [Freely chosen commitment scheme]
- 2353 – Proof that number is in $[0, 2^{64})$
- 2354 – Proof that number is positive
- 2355 • Proof of permutation (proving that two committed lists contain the same elements)

2356 3.6 Correctness and Trust

2357 In this section we explore the requirements for making the implementation of the proof system
 2358 trustworthy. Even if the mathematical scheme fulfills the claimed properties (e.g., it is proven
 2359 secure in the requisite sense, its assumptions hold and security parameters are chosen judiciously),
 2360 many things can go wrong in the subsequent implementation: code bugs, structured reference
 2361 string subversion, compromise during deployment, side channels, tampering attacks, etc. This
 2362 section aims to highlight such risks and offer considerations for practitioners.

2363 3.6.1 Considerations

2364 **Design of high-level protocol and statement.** The specification of the high-level protocol that
 2365 invokes the ZK proof system (and in particular, the NP statement to be proven in zero knowledge)
 2366 may fail to achieve the intended domain-specific security properties.

2367 Methodology for specifying and verifying these protocols is at its infancy, and in practice often relies
 2368 on manual review and proof sketches. Possible methods for attaining assurance include reliance
 2369 on peer-reviewed academic publications (e.g., Zerocash [BCGGMTV14] and Cinderella [DFKP16]),
 2370 reuse of high-level gadgets (see Section 4.4), careful manual specification and proving of protocol
 2371 properties by trained cryptographers, and emerging tools for formal verification.

2372 Whenever nontrivial optimizations are applied to a statement, such as algebraic simplification, or
 2373 replacement of an algorithm used in the original intended statement with a more efficient alternative,
 2374 those optimizations should be supported by proofs at an appropriate level of formality.

2375 **Choice of cryptographic primitives.** Traditional cryptographic primitives (hash functions, PRFs,
 2376 etc.) in common use are generally not designed for efficiency when implemented in circuits for
 2377 ZK proof systems. Within the past few years, alternative “circuit-friendly” primitives have been
 2378 proposed that may have efficiency advantages in this setting (e.g., LowMC and MiMC). We rec-
 2379 ommend a conservative approach to assessing the security of such primitives, and advise that the
 2380 criteria for accepting them need to be as stringent as for the more traditional primitives.

2381 **Implementation of statement.** The concrete implementation of the statement to be proven by
 2382 the ZK proof system (e.g., as a Boolean circuit or an R1CS) may fail to capture the high-level
 2383 specification. This risk increases if the statement is implemented in a low abstraction level, which
 2384 is more prone to errors and harder to reason about.

2385 The use of higher-level specifications and domain-specific languages (see the Front Ends section)
 2386 can decrease the risk of this error, though errors may still occur in the higher-level specifications
 2387 or in the compilation process.

2388 Additionally, risk of errors often arises in the context of optimizations that aim to reduce the size
2389 of the statement (e.g., circuit size or number of R1CS constraints).

2390 Note that correct statement semantics is crucial for security. Two implementations that use the
2391 same high-level protocol, same constraint system and compatible backends may still fail to correctly
2392 interoperate if their instance reductions (from high-level statement to the low-level input required by
2393 the backend) are incompatible — both in completeness (proofs don't verify) or soundness (causing
2394 false but convincing proofs, implying a security vulnerability).

2395 **Side channels.** Developers should be aware of the different processes in which side channel attacks
2396 can be detrimental and take measure to minimize the side channels. These include:

- 2397 • SRS generation — in some schemes, randomly sampled elements which are discarded can be
2398 used, if exposed, to subvert the soundness of the system.
- 2399 • Assignment generation / proving — the private auxiliary data can be exposed, which allows
2400 the attacker to understand the secret data used for the proof.

2401 **Auditing.** First of all, circuit designers should provide a high-level description of their circuit and
2402 statement alongside the low-level circuit, and explain the connections between them.

2403 The high-level description should facilitate auditing of the security properties of the protocol being
2404 implemented, and whether these match the properties intended by the designers or that are likely
2405 to be expected by users.

2406 If the low-level description is not expressed directly in code, then the correspondence between
2407 the code and the description should be clear enough to be checked in the auditing process, either
2408 manually or with tool support.

2409 A major focus of auditing the correctness and security of a circuit implementation will be in verifying
2410 that the low-level description matches the high-level one. This has several aspects, corresponding
2411 to the security properties of a ZK proof system:

- 2412 • An instance for the low-level circuit must reveal no more information than an instance for the
2413 high-level statement. This is most easily achieved by ensuring that it is a canonical encoding
2414 of the high-level instance.
- 2415 • It must not be possible to find an instance and witness for the low-level circuit that does not
2416 correspond to an instance and witness for the high-level statement.

2417 At all levels of abstraction, it is beneficial to use types to clarify the domains and representations
2418 of the values being manipulated. Typically, a given proving system will not be able to **directly**
2419 represent all of the types of value needed for a given high-level statement; instead, the values will
2420 be encoded, for example as field elements in the case of R1CS-based proof systems. The available
2421 operations on these elements may differ from those on the values they are representing; for instance,
2422 field addition does not correspond to integer addition in the case of overflow.

2423 An adversary who is attempting to prove an instance of the statement that was not intended to be
2424 provable, is not necessarily constrained to using instance and witness variables that correspond to
2425 these intended representations. Therefore, close attention is needed to ensuring that the constraint
2426 system explicitly excludes unintended representations.

2427 There is a wide space of design tradeoffs in how the frontend to a proof system can help to address
 2428 this issue. The frontend may provide a rich set of types suitable for directly expressing high-level
 2429 statements; it may provide only field elements, leaving representation issues to the frontend user;
 2430 it may provide abstraction mechanisms by which users can define new types; etc. Auditability of
 2431 statements expressed using the frontend should be a major consideration in this design choice.

2432 If the frontend takes a “gadget” approach to composition of statement elements, then it must be
 2433 clear whether each gadget is responsible for constraining the input and/or output variables to their
 2434 required types.

2435 **Testing.** Methods to test constraint systems include:

- 2436 • Testing for failure: does the implementation accept an assignment that should not be accepted?
- 2437 • Fuzzing the circuit inputs.
- 2438 • Finding missing constraints, e.g., missing Boolean constraints on variables that represent bits,
 2439 or other missing type constraints.
- 2440 • Finding dead constraints, and reporting them (instead of optimising out).
- 2441 • Detection of unintended nondeterminism. For instance, given a partial fixed assignment, solve
 2442 for the remainder and check that there is only one solution.

2443 A proof system implementation can support testing by providing access, for test and debugging
 2444 purposes, to the reason why a given assignment failed to satisfy the constraints. It should also
 2445 support injection of values for instance and witness variables that would not occur in normal use
 2446 (e.g. because they do not represent a value of the correct type). These features facilitate “white
 2447 box testing”, i.e. testing that the circuit implementation rejects an instance and witness *for the*
 2448 *intended reason*, rather than incidentally. Without this support, it is difficult to write correct tests
 2449 with adequate coverage of failure modes.

2450 3.6.2 SRS Generation

2451 A prominent trust issue arises in proving systems which require a parameter setup process (struc-
 2452 tured reference string) that involves secret randomness. These may have to deal with scenarios
 2453 where the process is vulnerable or expensive to perform security. We explore the real world so-
 2454 cial and technical problems that these setups must confront, such as air gaps, public verifiability,
 2455 scalability, handing aborts, and the reputation of participants, and randomness beacons.

2456 ZKP schemes require a URS (*uniform* reference string) or SRS (*structured* reference string) for their
 2457 soundness and/or ZK properties. This necessitates suitable randomness sources and, in the case of
 2458 a common reference string, a securely-executed setup algorithm. Moreover, some of the protocols
 2459 create reference strings that can be reused across applications. We thus seek considerations for
 2460 executing the setup phase of the leading ZKP scheme families, and for sharing of common resources.

2461 This section summarizes an open discussion made by the participants of the Implementation Track,
 2462 aiming to provide considerations for practitioners to securely generate a CRS.

2463 **SRS subversion and failure modes.** Constructing the SRS in a single machine might fit some
 2464 scenarios. For example, this includes a scenario where the verifier is a single entity — the one

2465 who generates the SRS. In that scenario, an aspect that should be considered is subversion zero-
 2466 knowledge — a property of proving schemes allowing to maintain zero-knowledge, even if the SRS
 2467 is chosen maliciously by the verifier.

2468 Strategies for subversion zero knowledge include:

- 2469 • Using a multi-party computation to generate the SRS
- 2470 • Adaptation of either [Gro16; PHGR13]
- 2471 • Updatable SRS — the SRS is generated once in a secure manner, and can then be specialized
 2472 to many different circuits, without the need to re-generate the SRS

2473 Other subversion considerations are discussed in Section 1.6.7.

2474 **SRS generation using MPC.** In order to reduce the need of trust in a single entity generating
 2475 the SRS, it is possible to use a multi-party computation to generate the SRS. This method should
 2476 ideally be secure as long as one participant is honest (per independent computation phase).

2477 **FF3.6. Proposed future figure (illustration, diagram, ...):**

2478 add figure of MPC ceremony representing SRS protocol

2479 Some considerations to strengthen the security of the MPC include:

- 2480 • Have as many participants as possible
 - 2481 – Diversity of participants; reduce the chance they will collude
 - 2482 – Diversity of implementations (curve, MPC code, compiler, operating system, language)
 - 2483 – Diversity of hardware (CPU architecture, peripherals, RAM)
 - 2484 * One-time-use computers
 - 2485 * GCP / EC2 (leveraging enterprise security)
 - 2486 – If you are concerned about your hardware being compromised, then avoid side channels
 2487 (power, audio/radio, surveillance)
 - 2488 * Hardware removal:
 - 2489 · Remove WiFi/Bluetooth chip
 - 2490 · Disconnect webcam / microphone / speakers
 - 2491 · Remove hard disks if not needed, or disable swap
 - 2492 * Air gaps
 - 2493 – Deterministic compilation
 - 2494 – Append-only logs
 - 2495 – Public verifiability of transcripts
 - 2496 – Scalability
 - 2497 – Handling aborts
 - 2498 – Reputation
- 2499 • Information extraction from the hardware is difficult
 - 2500 – Flash drives with hardware read-only toggle

2501 Some protocols (e.g., Powers of Tau) also require sampling unpredictable public randomness. Such
 2502 randomness can be harnessed from proof of work blockchains or other sources of entropy such as stock
 2503 markets. Verifiable Delay Functions can further reduce the ability to bias these sources [BBBF18].

2504 **SRS reusability.** For schemes that require an SRS, it may be possible to design an SRS generation
 2505 process that allows the re-usability of a part of the SRS, thus reducing the attack surface. A good
 2506 example of it is the “Powers of Tau” method [BGM17] for the Groth16 construction [Gro16]. There,
 2507 most of the SRS can be reused before specializing to a specific constraint system.

2508 **Designated-verifier setting.** There are cases where the verifier is a single entity known in advance.
 2509 There are schemes that excel in this setting. Moreover, schemes with public verifiability can be
 2510 specialized to this setting as well.

2511 3.6.3 Contingency plans

2512 The notion of contingency plans deserves future exploration. For example, how does one cope with:

- 2513 • a proof system being compromised?
- 2514 • a specific circuit having a bug?
- 2515 • a ZKP protocol being breached (identifying proofs with invalid witness, etc)

2516 Some ideas that can be expanded:

- 2517 • Scheme-agility and protocol-agility in protocols — when designing the system, allow flexibility
 2518 for the primitives used
- 2519 • Combiners (using multiple proof systems in parallel) — to reduce the reliance on a single
 2520 proof system, use multiple
- 2521 • Discuss ways to identify when ZKP protocol has been breached (identifying proofs with invalid
 2522 witness, etc.)

2523 3.7 Extended Constraint-System Interoperability

2524 The following are stronger forms of interoperability, which have been identified as desirable by
 2525 practitioners. They should be addressed further.

2526 3.7.1 Statement and witness formats

2527 In the R1CS File Format section and associated resources, we define a file format for R1CS con-
 2528 straint systems. There remains to finalize this specification, including instances and witnesses. This
 2529 will enable users to have their choice of frameworks (frontends and backends) and streaming for
 2530 storage and communication, and facilitate creation of benchmark test cases that could be executed
 2531 by any backend accepting these formats.

2532 Crucially, analogous formats are desired for constraint system languages other than R1CS.

2533 3.7.2 Statement semantics, variable representation & mapping

2534 Beyond the above, there is a need for different implementations to coordinate the semantics of the
 2535 statement (instance) representation of constraint systems. For example, a high-level protocol may
 2536 have an RSA signature as part of the statement, leaving ambiguity on how big integers modulo a
 2537 constant are represented as a sequence of variables over a smaller field, and at what indices these
 2538 variables are placed in the actual R1CS instance.

2539 Precise specification of statement semantics, in terms of higher-level abstraction, is needed for
 2540 interoperability of constraint systems that are invoked by several different implementations of the
 2541 instance reduction (from high-level statement to the actual input required by the ZKP prover and
 2542 verifier). One may go further and try to reuse the actual implementation of the instance reduction,
 2543 taking a high-level and possibly domain-specific representation of values (e.g., big integers) and
 2544 converting it into low-level variables. This raises questions of language and platform incompatibility,
 2545 as well as proper modularization and packaging.

2546 Note that correct statement semantics is crucial for security. Two implementations that use the
 2547 same high-level protocol, same constraint system and compatible backends may still fail to cor-
 2548 rectly interoperate if their instance reductions are incompatible — both in completeness (proofs do
 2549 not verify) or soundness (causing false but convincing proofs, implying a security vulnerability).
 2550 Moreover, semantics are a requisite for verification and helpful for debugging.

2551 Some backends can exploit uniformity or regularity in the constraint system (e.g., repeating patterns
 2552 or algebraic structure), and could thus take advantage of formats and semantics that convey the
 2553 requisite information.

2554 At the typical complexity level of today’s constraint systems, it is often acceptable to handle all of
 2555 the above manually, by fresh re-implementation based on informal specifications and inspection of
 2556 prior implementation. We expect this to become less tenable and more error prone as application
 2557 complexity grows.

2558 3.7.3 Witness reduction

2559 Similar considerations arise for the witness reduction, converting a high-level witness representation
 2560 (for a given statement) into the assignment to witness variables. For example, a high-level protocol
 2561 may use Merkle trees of particular depth with a particular hash function, and a high-level instance
 2562 may include a Merkle authentication path. The witness reduction would need to convert these
 2563 into witness variables, that contain all of the Merkle authentication path data (encoded by some
 2564 particular convention into field elements and assigned in some particular order) and moreover the
 2565 numerous additional witness variables that occur in the constraints that evaluate the hash function,
 2566 ensure consistency and Booleanity, etc.

2567 The witness reduction is highly dependent on the particular implementation of the constraint
 2568 system. Possible approaches to interoperability are, as above: formal specifications, code reuse and
 2569 manual ad hoc compatibility.

2570 3.7.4 Gadgets interoperability

2571 At a finer grain than monolithic constraint systems and their assignments, there is need for sharing
 2572 subcircuits and gadgets. For example, `libsark` offers a rich library of highly optimized R1CS
 2573 gadgets, which developers of several front-end compilers would like to reuse in the context of their
 2574 own constraint-system construction framework.

2575 While porting chunks of constraints across frameworks is relatively straightforward, there are chal-
 2576 lenges in coordinating the semantics of the externally-visible variables of the gadget, analogous
 2577 to but more difficult than those mentioned above for full constraint systems: there is a need to
 2578 coordinate or reuse the semantics of a gadget’s externally-visible variables, as well as to coordinate
 2579 or reuse the witness reduction function of imported gadgets in order to convert a witness into an
 2580 assignment to the internal variables.

2581 As for instance semantics, well-defined gadget semantics is crucial for soundness, completeness and
 2582 verification, and is helpful for debugging.

2583 3.7.5 Procedural interoperability

2584 An attractive approach to the aforementioned needs for instance and witness reductions (both at the
 2585 level of whole constraint systems and at the gadget level) is to enable one implementation to invoke
 2586 the instance/witness reductions of another, even across frameworks and programming languages.

2587 This requires communication not of mere data, but invocation of procedural code. Suggested ap-
 2588 proaches to this include linking against executable code (e.g., `.so` files or `.dll`), using some elegant
 2589 and portable high-level language with its associated portable, or using a low-level portable ex-
 2590 cutable format such as `WebAssembly`. All of these require suitable calling conventions (e.g., how
 2591 are field elements represented?), usage guidelines and examples.

2592 Beyond interoperability, some low-level building blocks (e.g., finite field and elliptic curve arithmetic)
 2593 are needed by many or all implementations, and suitable libraries can be reused. To a large extent
 2594 this is already happening, using the standard practices for code reuse using native libraries. Such
 2595 reused libraries may offer a convenient common ground for consistent calling conventions as well.

2596 3.7.6 Proof interoperability

2597 Another desired goal is interoperability between provers and verifiers that come from different
 2598 implementations, i.e., being able to independently write verifiers that make consistent decisions
 2599 and being able to re-implement provers while still producing proofs that convince the old verifier.

2600 This is especially pertinent in applications where proofs are posted publicly, such as in the context
 2601 of blockchains (see Chapter 4), and multiple independent implementations are desired for both
 2602 provers and verifiers.

2603 To achieve such interoperability, provers and verifiers must agree on all of the following:

- 2604 • ZK proof system (fixing all degrees of freedom, e.g., choice of finite fields and elliptic curves)
- 2605 • Formats for instance/statement and witness (see Section 3.7.1)

- 2606 • Format for Prover and Verifier parameters
- 2607 • Format for the proof
- 2608 • A precise specification of the constraint system (e.g., R1CS) and corresponding instance and
- 2609 witness reductions (see Section 3.7.3).

2610 Alternatively: a precise high-level specification along with a precisely-specified, deterministic fron-

2611 tend compilation.

2612 3.7.7 Common reference strings

2613 There is also a need for standardization regarding Common Reference String (CRS), i.e., prover

2614 parameters and verifier parameters. First, interoperability is needed for streaming formats (com-

2615 munication and storage), and would allow application developers to easily switch between different

2616 implementations, with different security and performance properties, to suit their need. Moreover,

2617 for Structured Reference Strings (SRS), there are nontrivial semantics that depend on the ZK proof

2618 system and its concrete realization by backends, as well as potential for partial reuse of SRS across

2619 different circuits in some schemes (e.g., the Powers of Tau protocol).

2620 3.8 Future goals

2621 3.8.1 Interoperability

2622 Many additional aspects of interoperability remain to be analyzed and supported by standards,

2623 to support additional ZK proof system backends as well as additional communication and reuse

2624 scenarios.

2625 **Additional forms of interoperability.** As discussed in the Extended Constraint-System Interop-

2626 erability section above, even within the R1CS realm, there are numerous additional needs beyond

2627 plain constraint systems and assignment representations. These affect security, functionality and

2628 ease of development and reuse.

2629 **Additional relation styles.** The R1CS-style constraint system has been given the most focus

2630 in the Implementation Track discussions in the first workshop, leading to a file format and an

2631 API specification suitable for it. It is an important goal to discuss other styles of constraint

2632 systems, which are used by other ZK proof systems and their corresponding backends. This includes

2633 arithmetic and Boolean circuits, variants thereof which can exploit regular/repeating elements, as

2634 well as arithmetic constraint satisfaction problems.

2635 **Recursive composition.** The technique of recursive composition of proofs, and its abstraction as

2636 Proof-Carrying Data (PCD) [CT10; BCTV14a], can improve the performance and functionality of

2637 ZK proof systems in applications that deal with multi-stage computation or large amounts of data.

2638 This introduces additional objects and corresponding interoperability considerations. For example,

2639 PCD compliance predicates are constraint systems with additional conventions that determine their

2640 semantics, and for interoperability these conventions require precise specification.

2641 **Benchmarks.** We strive to create concrete reference benchmarks and reference platforms, to enable
2642 cross-paper milliseconds comparisons and competitions.

2643 We seek to create an open competition with well-specified evaluation criteria, to evaluate different
2644 proof schemes in various well-defined scenarios.

2645 **3.8.2 Frontends and DSLs**

2646 We would like to expand the discussion on the areas of domain-specific languages, specifically in
2647 aspects of interoperability, correctness and efficiency (even enabling source-to-source optimisation).

2648 The goal of Gadget Interoperability, in the Extended Constraint-System Interoperability section,
2649 is also pertinent to frontends.

2650 **3.8.3 Verification of implementations**

2651 We would to discuss the following subjects in future workshops, to assist in guiding towards best
2652 practices: formal verification, auditing, consistency tests, etc.

Chapter 4. Applications

4.1 Introduction

This chapter aims to overview existing techniques for building ZKP-based applications, including designing the protocols to meet best-practice security requirements. We distinguish between high-level and low-level applications, where the former are the protocols designed for specific use-cases and the latter are the necessary underlying operations or sub-protocols. Each use case admits a circuit, and we discuss the sub-circuits needed to ensure security and functionality of the protocol. We refer to the circuits as *predicates* and the sub-circuits as *gadgets*:

- **Predicate:** The relation or condition that the statement and witness must satisfy. Can be represented as a circuit.
- **Gadget:** The underlying tools needed to construct the predicate. In some cases, a gadget can be interpreted as a security requirement (e.g., using the commitment verification gadget is equivalent to ensuring the privacy of underlying data).

Recall from Section 1.5 the syntax of a proof system between a prover and verifier. As we will see, the protocols can be abstracted and generalized to admit several use-cases; similarly, there exist compilers that will generate the necessary gadgets from commonly used programming languages. Creating the constraint systems is a fundamental part of the applications of ZKP, which is the reason why there is a large variety of front-end software options available.

Functionality vs. performance. The design of ZKPs is subject to the tradeoff between functionality and performance. Users would like to have powerful ZKPs, in the sense that the system permits constructing proofs for any predicate, which leads to the necessity of universal ZKPs. On the other hand, users would like to have efficient constructions. According to Table 3.4.1, it is possible to classify ZKPs as: (i) universal or non-universal; (ii) scalable or non-scalable; and (iii) preprocessing or non-preprocessing.

Item (i) is related to the functionality of the underlying ZKP, while items (ii) and (iii) are related to performance. The utilization of zk-SNARKs allows universal ZKPs with very efficient verifiers. However, many proposals depend upon an expensive preprocessing, which makes such systems hard to scale for some use-cases. A technique called *Proof-Carrying Data* (PCD), originally proposed in Ref. [CT10], allows obtaining *recursive composition* for existing ZKPs in a modular way. This means that zk-SNARKs can be used as a building block to construct scalable and non-preprocessing solutions. The result is not only an efficient verifier, as in zk-SNARKs, but also a prover whose consumption of computational resources is efficient, in particular with respect to memory requirements, as described in Refs. [BCTV14a] and [BCCT13].

Organization. Section 4.2 mentions different types of verifiability properties of interest to applications. Section 4.3 enumerates some prior works. Section 4.4 describes possible gadgets useful for diverse applications. The subsequent three sections present three ZKP use-cases: Section 4.5 describes a use-case related to *identity management*; Section 4.6 examines an application context related to *asset transfer*; Section 4.7 exemplifies one use-case related to *regulation compliance*.

2691 4.2 Types of verifiability

2692 **Verifiability type.** When designing ZK based applications, one needs to keep in mind which of the
2693 following three models (that define the functionality of the ZKP) is needed:

- 2694 1. **Public.** Publicly verifiable as a requirement: a scheme / use-case where there is a system
2695 requirement that the proofs are transferable.
- 2696 2. **Designated.** Designated verifier as a security feature: only the intended receiver of the proof
2697 can verify it, making the proof non-transferable. This property can apply to both interactive
2698 and non-interactive ZKPs.
- 2699 3. **Optional.** There is no need to be able to transfer but also no non-transferability requirement.
2700 This property is applicable both in the interactive and in the non-interactive model.

2701 Section 2.6.3 discusses transferability vs. deniability, which is strongly related to aspects of public
2702 verifiability vs. designated verifiability, both in the interactive and in the non-interactive settings.
2703 As a use-case example, consider some application related to blockchain currency, where aspects of
2704 user-privacy and regulatory-control are relevant.

2705 Publicly-verifiable ZKPs can be appropriate when the validity of a transaction should be public
2706 (e.g., so that everyone knows that some asset changed owner), while some supporting data needs to
2707 remain private (e.g., the secret key of a blockchain address, controlling the ownership of the asset).
2708 However, sometimes even the statement being proven should remain private beyond the scope of
2709 the verifier, and therefore a non-transferable proof should be used. This may apply for example to a
2710 proof of having enough funds available for a purchase, or also of knowing the secret key of a certain
2711 blockchain address. Alice wants to prevent Bob from using the received proof to convince Charley of
2712 the claims made by Alice. For that purpose, Alice can perform a deniability interactive proof with
2713 Bob. Alternatively, Alice can send to Bob a (non-interactive) proof transcript built for Bob as a
2714 *designated verifier*. Depending on the use case, both public-verifiability and designated-verifiability
2715 can make sense as an application goal. It is important to distinguish between both.

2716 The “designation of verifiers” allows resolving possible conflicts between authenticity and privacy
2717 [JSI96]. For example, a voting center wants only Bob to be convinced that the vote he cast was
2718 counted; the voting center designates Bob to be the one convinced by the validity of the proof, in
2719 order to prevent a malicious coercer to force him to prove how he voted. Since the designated-verifier
2720 proofs are non-transferable, Bob cannot transfer the proof even if he wants to.

2721 Suppose Alice wants to convince *only* Bob that a statement θ is true. For that purpose, Alice can
2722 prove the disjunction “Either θ is true or I know the secret key of Bob”. Given that Bob knows his
2723 own secret key, Bob could have produced such proof by himself. Therefore, a third party Charlie
2724 will not be convinced that θ is true after seeing such proof transcript sent from Bob. This holds
2725 even if Bob shares his secret key to Charlie, or if the key has been publicly leaked.

2726 Designated proofs are possible both in the interactive and non-interactive settings. In the interac-
2727 tive setting (e.g., proving being the signer of an undeniable signature) the prover has the ability
2728 to control when the verification takes place. However, in general (without a designated-verifier
2729 approach) the prover may be unable to control who is able to verify the proof, namely if the verifier
2730 is acting as a relay to another controlling party. The use of a designated proof has the potential
2731 to solve this problem.

2732 4.3 Previous works

2733 This section includes an overview of some of the works and applications existing in the ZKP world.

2734 **Contribution wanted:** Contribution needed: add more references

2735 ZKP protocols for anonymous credentials have been studied extensively in academic spaces [CKS10;
2736 BCDEK+14; CDD17; BCDLRSY17; NVV18]. Products such as Miracl, Val:ID, Sovrin [Sov18],
2737 and LibZmix [Mik19] offer practical solutions to achieve privacy-preserving identity frameworks.

2738 Zerocash began as an academic work and was later developed into a product ensuring anonymous
2739 transactions [BCGGMTV14]. Baby ZoE enables Zerocash over Ethereum [zca18]. HAWK also uses
2740 zk-SNARKS to enable smart-contracts with transactional privacy [KMSWP16].

2741 4.4 Gadgets within predicates

2742 Formalizing the security of these protocols is a very difficult task, especially since there is no
2743 predetermined set of requirements, making it an ad-hoc process.

2744 Use-cases must be sure to distinguish between privacy requirements and security guarantees. We
2745 discuss the use-case case of privacy-preserving asset transfer to illustrate the difference.

2746 Secure asset transfer is possible at several financial institutions, provided that the institution has
2747 knowledge of the identities of the sender, recipient, asset, and amount. In a privacy-preserving asset
2748 transfer, the identities of sender and recipient may be concealed even from the entity administering
2749 the transfer. It is important to note that a successful transfer must meet privacy requirements as
2750 well as provide security guarantees.

2751 Privacy requirements might include the anonymity of sender and recipient, concealment of asset
2752 type and asset amount. Security guarantees might include the inability of anyone besides the sender
2753 to initiate a transfer on the sender's behalf or the inability of a sender to execute a transfer of asset
2754 type without sufficient holdings of the asset.

2755 Here we outline a set of initial gadgets to be taken into account. See Table 4.1 for a simple list
2756 of gadgets — this list should be expanded continuously and on a case by case basis. For each of
2757 the gadgets we write the following representations, specifying what is the secret / witness, what is
2758 public / statement:

2759 NP statements for non-technical people:

2760 **For the [public] chess board configurations A and B ;
I know some [secret] sequence S of chess moves;
such that when starting from configuration A , and applying S , all moves are
legal and the final configuration is B .**

2761 General form (Camenisch-Stadler): $\mathbf{Zk} \{ (\text{wit}): \mathbf{P}(\text{wit}, \text{statement}) \}$

2762 Example of ring signature: $\mathbf{Zk} \{ (\text{sig}): \mathbf{VerifySignature}(\mathbf{P1}, \text{sig}) \text{ or } \mathbf{VerifySignature}(\mathbf{P2}, \text{sig}) \}$

Table 4.1: List of gadgets

#	Gadget name	Intuitive description of the initial gadget (before adding ZKP)	Table with examples
G1	Commitment	Envelope	Table 4.2
G2	Signatures	Signature authorization letter	Table 4.3
G3	Encryption	Envelope with a receiver stamp	Table 4.4
G4	Distributed decryption	Envelope with a receiver stamp that requires multiple people to open	Table 4.5
G5	Random function	Lottery machine	Table 4.6
G6	Set membership	Whitelist/blacklist	Table 4.7
G7	Mix-net	Ballot box	Table 4.8
G8	Generic circuits, TMs, or RAM programs	General calculations	Table 4.9

Table 4.2: Commitment gadget (G1; envelope)

Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
I know the value hidden inside this envelope, even though I cannot change it	Knowledge of committed value(s) (openings)	Opening $O = (v, r)$ containing a value and randomness	Commitment C	$C = \text{Comm}(v, r)$
I know that the value hidden inside these two envelopes are equal	Equality of committed values	Openings $O_1 = (v, r_1)$ and $O_2 = (v, r_2)$	Commitments C_1 and C_2	$C_1 = \text{Comm}(v, r_1)$ and $C_2 = \text{Comm}(v, r_2)$
I know that the values hidden inside these two envelopes are related in a specific way	Relationships between committed values – logical, arithmetic, etc.	Openings $O_1 = (v_1, r_1)$ and $O_2 = (v_2, r_2)$	Commitments C_1 and C_2 , relation R	$C_1 = \text{Comm}(v_1, r_1)$, $C_2 = \text{Comm}(v_2, r_2)$, and $R(v_1, v_2) = \text{True}$
The value inside this envelope is within a particular range	Range proofs	Opening $O = (v, r)$	Commitment C , interval I	$C = \text{Comm}(v, r)$ and v is in the range I

2793

Table 4.3: Signature gadget (G2; signature authorization letter)

2794 2795	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2797 2798 2799	Secret valid signature over commonly known message	Knowledge of a secret signature σ on a commonly known message M	Signature σ	Verification key VK , message M	$\text{Verify}(VK, M, \sigma) = \text{True}$
2801 2802 2803	Secret valid signature over committed message	Knowledge of a secret signature σ on a commonly known commitment C of a secret message M	Opening O , signature σ	Verification key VK , commitment C	$C = \text{Comm}(M)$ and $\text{Verify}(VK, M, \sigma) = \text{True}$

2805

Table 4.4: Encryption gadget (G3; envelope with a receiver stamp)

2806 2807	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2809 2810 2811	The output plaintext(s) correspond to the public ciphertext(s).	Knowledge of a secret plaintext M	Secret decryption key SK	Ciphertext(s) C and Encryption key PK	$\text{Dec}(SK, C) = M$, component-wise if \exists multiple C and M

2813

Table 4.5: Distributed-decryption gadget

2814

(G4; envelope with a receiver stamp that requires multiple people to open)

2815 2816	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2818 2819 2820	The output plaintext(s) correspond to the public ciphertext(s).	Knowledge of a secret plaintext M	Secret shares $[SK_i]$ of the decryption key SK	Ciphertext(s) C and Encryption key PK	$SK = \text{Derive}([SK_i])$ and $\text{Dec}(SK, C) = M$, component-wise if \exists multiple C

2822

Table 4.6: Random-function gadget (G5; lottery machine)

2823 2824	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2826 2827	Verifiable random function (VRF)	VRF was computed from a secret seed and a public (or secret) input	Secret seed W	Input X , Output Y	$Y = \text{VRF}(W, X)$

2829

Table 4.7: Set-membership gadget (G6; whitelist/blacklist)

2830 2831	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2833	Accumulator	Set inclusion	Secret element X	Public set S	$X \in S$
2835 2836	Universal accumulator	Set non-inclusion	Secret element X	Public set S	$X \notin S$
2838	Merkle Tree	Element occupies a certain position within the vector	Secret element X	Public vector V	$X = V[i]$ for some i

2840

Table 4.8: Mix-net gadget (G7; ballot box)

2841 2842	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2844	Shuffle	The set of plaintexts in the input and the output ciphertexts respectively are identical.	Permutation π , Decryption key SK	Input ciphertext list C and Output ciphertext list C'	$\forall j, Dec(SK, C'_j) = Dec(SK, \pi(C_j))$
2846	Shuffle and reveal	The set of plaintexts in the input ciphertexts is identical to the set of plaintexts in the output.	Permutation π , Decryption key SK	Input ciphertext list C and Output plaintext list P	$\forall j, P_j = Dec(SK, \pi(C_j))$

2848

Table 4.9: Generic circuits, TMs, or RAM programs gadgets (G8; general calculations)

2849 2850	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2852 2853 2854	There exists some secret input that makes this calculation correct	ZK proof of correctness of circuit/Turing machine/RAM program computation	Secret input w	Program C (either a circuit, TM, or RAM program), public input x , output y	$C(x, w) = y$
2856 2857 2858 2859	This calculation is correct, given that I already know that some sub-calculation is correct	ZK proof of verification + post-processing of another output (Composition)	Secret input w	Program C with subroutine C' , public input x , output y , intermediate value $z = C'(x, w)$, zk proof π that $z = C'(x, w)$	$C(x, w) = y$

2861 4.5 Identity framework

2862 4.5.1 Overview

2863 In this section we describe identity management solutions using zero knowledge proofs. The idea
 2864 is that some user has a set of attributes that will be attested to by an issuer or multiple issuers,
 2865 such that these attestations correspond to a validation of those attributes or a subset of them.

2866 After attestation it is possible to use this information, hereby called a credential, to generate a
 2867 claim about those attributes. Namely, consider the case where Alice wants to show that she is
 2868 over 18 and lives in a country that belongs to the European Union. If two issuers were responsible
 2869 for the attestation of Alice's age and residence country, then we have that Alice could use zero
 2870 knowledge proofs in order to show that she possesses those attributes, for instance she can use zero
 2871 knowledge range proofs to show that her age is over 18, and zero knowledge set membership to
 2872 prove that she lives in a country that belongs to the European Union. This proof can be presented
 2873 to a Verifier that must validate such proof to authorize Alice to use some service. Hence there are
 2874 three parties involved: (i) the credential holder; (ii) the credential issuer; (iii) and the verifier.

2875 4.5.2 Motivation for Identity and Zero Knowledge

2876 Digital identity has been a problem of interest to both academics and industry practitioners since
 2877 the creation of the internet. Specifically, it is the problem of allowing an individual, a company,
 2878 or an asset to be identified online without having to generate a physical identification for it, such
 2879 as an ID card, a signed document, a license, etc. Digitizing Identity comes with some unique
 2880 risks, loss of privacy and consequent exposure to Identity theft, surveillance, social engineering and
 2881 other damaging efforts. Indeed, this is something that has been solved partially, with the help
 2882 of cryptographic tools to achieve moderate privacy (password encryption, public key certificates,
 2883 internet protocols like TLS and several others). Yet, these solutions are sometimes not enough
 2884 to meet the privacy needs to the users / identities online. Cryptographic zero knowledge proofs
 2885 can further enhance the ability to interact digitally and gain both privacy and the assurance of
 2886 legitimacy required for the correctness of a process.

2887 The following is an overview of the generalized version of the identity scheme. We define the
 2888 terminology used for the data structures and the actors, elaborate on what features we include and
 2889 what are the privacy assurances that we look for.

2890 4.5.3 Terminology / Definitions

2891 **Data structures.** In this protocol, we use several different data structures to represent the informa-
 2892 tion being transferred or exchanged between the parties. We have tried to generalize the definitions
 2893 as much as possible, while adapting to the existing Identity standards and previous ZKP works.

- 2894 • **Attribute:** The most fundamental information about a holder in the system (e.g.: age,
 2895 nationality, academic degree, professional certificate, pending debt, etc). These are the prop-
 2896 erties that are factual and from which specific authorizations can be derived.

- 2897 • **(Confidential and Anonymous) Credential:** The data structure that contains one or
2898 more attributes about a holder in the system (e.g.: credit card statement, marital status,
2899 age, address, etc). Since it contains private data, a credential is not shareable.
- 2900 • **(Verifiable) Claim:** A zero-knowledge predicate about the attributes in a credential (or
2901 many of them). A claim must be done about an identity and should contain some form of
2902 logical statement that is included in the constraint system defined by the zk-predicate.
- 2903 • **Proof of Credential:** The zero knowledge proof that is used to verify the claim attested
2904 by the credential. Given that the credential is kept confidential, the proof derived from it is
2905 presented as a way to prove the claim in question.

2906 **The parties in the protocol:**

- 2907 • **Holder:** The party whose attributes will be attested to. The holder holds the credentials
2908 that contain his / her attributes and generates Zero Knowledge Proofs to prove some claim
2909 about these. We say that the holder presents a proof of credential for some claim.
- 2910 • **Issuer:** The party that attests attributes of holders. We say that the issuer issues a credential
2911 to the holder.
- 2912 • **Verifier:** The party that verifies some claim about a holder by verifying the zero knowledge
2913 proof of credential to the claim.

2914 **Remark.** The main difference between this protocol and a non-ZK based Identity protocol is the
2915 fact that in the latter, the holder presents the credentials themselves as the proof for the claim
2916 / authorization, whereas in this protocol, the holder presents a zero knowledge proof that was
2917 computed from the credentials.

2918 **4.5.4 The Protocol Description**

2919 **Functionality.** There are many interesting features that we considered as part of the identity
2920 protocol. There are four basic functionalities that we decided to include from the get go:

- 2921 1. third party anonymous and confidential attribute attestations through **credential issuance**
2922 by the issuer;
- 2923 2. confidentially proving claims using zero knowledge proofs through the **presentation of proof**
2924 **of credential** by the holder;
- 2925 3. **verification of claims** through zero knowledge proof verification by the verifier; and
- 2926 4. unlinkable **credential revocation** by the issuer.

2927 There are other interesting and worth exploring functionalities, not included in this protocol version.
2928 Some of these are credential transfer, authority delegation and trace auditability.

2929 **Privacy requirements.** One should aim for a high level of privacy for each of the actors in
2930 the system, but without compromising the correctness of the protocol. We look at anonymity
2931 properties for each of the actors, confidentiality of their interactions and data exchanges, and
2932 at the unlinkability of public data (in committed form). These usually can be instantiated as
2933 cryptographic requirements such as commitment non-malleability, indistinguishability from random

2934 data, unforgeability, accumulator soundness or as statements in zero-knowledge such as proving
 2935 knowledge of preimages, proving signature verification, etc.

- 2936 • **Holder anonymity:** the underlying physical identity of the holder must be hidden from the
 2937 general public, and if needed from the issuer and verifier too. For this we use pseudo-random
 2938 strings called identifiers, which are tied to a secret only known to the holder.
- 2939 • **Issuer anonymity:** only the holder should know what issuer issued a specific credential.
- 2940 • **Anonymous credential:** when a holder presents a credential, the verifier may not know
 2941 who issued the certificate. He / She may only know that the credential was issued by some
 2942 approved issuer.
- 2943 • **Holder untraceability:** the holder identifiers and credentials cannot be used to track holders
 2944 through time.
- 2945 • **Confidentiality:** no one but the holder and the issuer should know what the credential
 2946 attributes are.
- 2947 • **Identifier linkability:** no one should be able to link two identifier unless there is a proof
 2948 presented by the holder.
- 2949 • **Credential linkability:** No one should be able to link two credentials from the publicly
 2950 available data. Mainly, no two issuers should be able to collude and link two credentials to
 2951 one same holder by using the holder's digital identity.

2952 **In depth view.** For the specific instantiation of the scheme, we examine the different ways in which
 2953 these requirements can be achieved and what are the trade-offs (e.g.: using pairwise identifiers vs.
 2954 one fixed public key; different revocation mechanisms; etc.) and elaborate on the privacy and
 2955 efficiency properties of each.

2956 **Functionalities vs. privacy and robustness requirements.** Several aspects of the instantiation
 2957 method, proof details and privacy/robustness are described in the following four tables related to
 2958 four functionalities/problems: Holder identification (Table 4.10); Issuer identification (Table 4.11);
 2959 Credential issuance (Table 4.12); Credential revocation (Table 4.13).

2960

Table 4.10: Holder identification: how to identify a holder of credentials

Instantiation Method	Proof Details	Privacy / Robustness
2961 2962 2963 2964 2965 2966 2967 2968 2969 Single identifier in the federated realm: PRF based Public Key (idPK) derived from the physical ID of the entity and attested / onboarded by a federal authority	- The first credential an entity must get is the onboarding credential that attests to its identity on the system - Any proof of credential generated by the holder must include a verification that the idPK was issued an onboarding credential	- Physical identity is hidden yet connected to the public key. - Issuers can collude to link different credentials by the same holder. - An entity can have only one identity in the system
2970 2971 2972 2973 2974 2975 2976 Single identifier in the self-sovereign realm: PRF based Public Key (idPK) self derived by the entity.	- Any proof of credential must show the holder knows the preimage of the idPK and that the credential was issued to the idPK in question	- Physical identity is hidden and does not necessarily have to be connected to the public key - Issuers can collude to link different credentials by the same holder - An entity can have several identities and conveniently forget any of them upon issuance of a “negative credential”
2977 2978 2979 2980 2981 2982 2983 Multiple identifiers: Pair-wise identification through identifiers. For each new interaction the holder generates a new identifier.	- Every time a holder needs to connect to a previous issuer, it must prove a connection of the new and old identifiers in ZK - Any proof of credential must show the holder knows the secret of the identifier that the credential was issued to.	- Physical identity is hidden and does not necessarily have to be connected to the public key - A set of colluding issuers is not able to link the credentials by the same holder - An entity can have several identities and conveniently forget any of them upon issuance of a “negative credential”

2985

Table 4.11: Issuer identification

Instantiation Method	Proof Details	Privacy / Robustness
2986 2987 2988 2989 2990 2991 2992 2993 2994 2995 Federated permissions: there is a list of approved issuers that can be updated by either a central authority or a set of nodes	- To accept a credential one must validate the signature against one from the list. To maintain the anonymity of the issuer, ring signatures can be used - For every proof of credential, a holder must prove that the signature in its credential is of an issuer in the approved list	- The verifier / public would not know who the issuer of the credential is but would know it is approved.
2996 2997 2998 2999 3000 3001 3002 3003 3004 3005 Free permissions: anyone can become an issuer, which use identifiers: - Public identifier: type 1 is the issuer whose signature verification key is publicly available - Pair-wise identifiers: type 2 is the issuer whose signature verification key can be identified only pair-wise with the holder / verifier	- The credentials issued by type 1 issuers can be used in proofs to unrelated parties - The credentials issued by type 2 issuers can only be used in proofs to parties who know the issuer in question.	- If ring signatures are used, the type one issuer identifiers would not imply that the identity of the issuer can be linked to a credential, it would only mean that “Key K _a belongs to company A” - Otherwise, only the type two issuers would be anonymous and un-linkable to credentials

3007

Table 4.12: Credential Issuance

Instantiation Method	Proof Details	Privacy / Robustness
3009 Blind signatures: the issuer 3010 signs on a commitment of a 3011 self-attested credential after 3012 seeing a proof of correct at- 3013 testation; a second kind of 3014 proof would be needed in the 3015 system 3016 3017 3018	<ul style="list-style-type: none"> - The proof of correct attestation must contain the structure, data types, ranges and credential type that the issuer allows - In some cases, the proof must contain verification of the attributes themselves (e.g.: address is in Florida, but not know the city) - The proof of credential must not be accepted if the signature of the credential was not verified either in zero-knowledge or as part of some public verification 	<ul style="list-style-type: none"> - Issuer’s signatures on credentials add limited legitimacy: a holder could add specific values / attributes that are not real and the issuer would not know - An Issuer can collude with a holder to produce blind signatures without the issuer being blamed
3019 In the clear signatures: the 3020 issuer generates the attestation, 3021 signing the commitment and 3022 sending the credential in 3023 the clear to the holder 3024	<ul style="list-style-type: none"> - The proof of credential must not be accepted if the signature of the credential was not verified either in zero-knowledge or as part of some public verification 	<ul style="list-style-type: none"> - Issuer must be trusted, since she can see the Holder’s data and could share it with others - The signature of the issuer can be trusted and blame could be allocated to the issuer

3026

Table 4.13: Credential Revocation

Instantiation Method	Proof Details	Privacy / Robustness
3028 Positive accumulator revo- 3029 cation: the issuer revokes 3030 the credential by removing 3031 an element from an accu- 3032 mulator [BCDLRSY17] 3033 3034 3035 3036 3037 3038	<ul style="list-style-type: none"> - The holder must prove set membership of a credential to prove it was issued and was not revoked at the same time - The issuer can revoke a credential by removing the element that represents it from the accumulator 	<ul style="list-style-type: none"> - If the accumulator is maintained by a central authority, then only the authority can link the revocation to the original issuance, avoiding timing attacks by general parties (join-revoke linkability) - If the accumulator is maintained through a public state, then there can be linkability of revocation with issuance since one can track the added values and test its membership
3039 Negative accumulator re- 3040 vocation: the issuer re- 3041 vokes by adding an element 3042 to an accumulator 3043 3044 3045 3046 3047 3048 3049 3050	<ul style="list-style-type: none"> - The holder must prove set membership of a credential to prove it was issued - The issuer can revoke a credential by adding to the negative accumulator the revocation secret related to the credential to be revoked - The holder must prove set non-membership of a revocation secret associated to the credential in question - The verifier must use the most recent version of the accumulator to validate the claim 	<ul style="list-style-type: none"> - Even when the accumulator is maintained through a public state, the revocation cannot be linked to the issuance since the two events are independent of each other

3052

3053

Gadgets. Each of the methods for instantiating the different functionalities use some of the following gadgets that have been described in the Gadgets section. There are three main parts to

3054 the predicate of any proof.

- 3055 1. The first is proving the veracity of the identity, in this case the holder, for which the following
3056 gadgets can / should be used:
 - 3057 • **Commitment** for checking that the identity has been attested to correctly.
 - 3058 • **PRF** for proving the preimage of the identifier is known by the holder.
 - 3059 • **Equality of strings** to prove that the new identifier has a connection to the previous
3060 identifier used or to an approved identifier.
- 3061 2. Then there is the part of the constraint system that deals with the legitimacy of the creden-
3062 tials, the fact that it was correctly issued and was not revoked.
 - 3063 • **Commitment** for checking that the credential was correctly committed to.
 - 3064 • **PRF** for proving that the holder knows the credential information, which is the preimage
3065 of the commitment.
 - 3066 • **Equality of strings** to prove that the credential was issued to an identifier connected
3067 to the current identifier.
 - 3068 • **Accumulators (Set membership / non-membership)** to prove that the commit-
3069 ment to the credential exists in some set (usually an accumulator), implying that it was
3070 issued correctly and that it was not revoked.
- 3071 3. Finally there is the logic needed to verify the rules / constraints imposed on the attributes
3072 themselves. This part can be seen as a general gadget called “credentials”, which allows to
3073 verify the specific attributes embedded in a credential. Depending on the credential type, it
3074 uses the following low level gadgets:
 - 3075 • **Data Type** used to check that the data in the credential is of the correct type.
 - 3076 • **Range Proofs** used to check that the data in the credential is within some range.
 - 3077 • **Arithmetic Operations (field arithmetic, large integers, etc.)** used for verifying
3078 arithmetic operations were done correctly in the computation of the instance.
 - 3079 • **Logical Operators (bigger than, equality, etc.)** used for comparing some value in
3080 the instance to the data in the credentials or some computation derived from it.

3081 Security caveats:

- 3082 1. If the Issuer colludes with the Verifier, they could use the revocation mechanism to reveal
3083 information about the Holder if there is real-time sharing of revocation information.
- 3084 2. Furthermore, if the commitments to credentials and the revocation information can be tracked
3085 publicly and the events are dependent of each other (e.g.: revocation by removing a commit-
3086 ment), then there can be linkability between issuance and revocation.
- 3087 3. In the case of self-attestation or collusion between the issuer and the holder, there is a much
3088 lower assurance of data integrity. The inputs to the ZKP could be spoofed and then the proof
3089 would not be sound.
- 3090 4. The use of Blockchains create a reliance on a trusted oracle for external state. On the other
3091 hand, the privacy guaranteed at blockchain-content level is orthogonal to network-level traffic
3092 analysis.

3093 **4.5.5 A use-case example of credential aggregation**

3094 We are going to focus our description on a specific use case: accredited investors. In this scenario
 3095 the credential holder will be able to show that she is accredited without revealing more information
 3096 than necessary to prove such a claim.

3097 **Use-case description.** As a way to illustrate the above protocol, we present a specific use-case
 3098 and explicitly write the predicate of the proof. Mainly, there is an identity, Alice, who wants to
 3099 prove to some company, Bob Inc. that she is an accredited investor, under the SEC rules, in order
 3100 to acquire some company shares. Alice is the prover; the IRS, the AML entity and The Bank are
 3101 all issuers; and Bob Inc. is the verifier.

3102 The different processes in the adaptation of the use-case are the following:

3103 1. Three confidential credentials are issued to Alice, representing the rules that we apply on an
 3104 entity to be an accredited investor. We assume that the SEC generates the constraint system
 3105 for the accreditation rules as the circuit used to generate the proving and verification keys.
 3106 In the real scenario, there are are Federal Rules for accreditation [ECFR].

3107 (a) The IRS issues a tax credential, C_0 , that testifies to the claim “from 1/1/2017 until
 3108 1/1/2018, Alice, with identifier X_0 , owes 0\$ to the IRS, with identifier Y ” and holds two
 3109 attributes: the net income of Alice, $\$income$, and a bit b such that $b = 1$ if Alice has
 3110 paid her taxes.

3111 (b) The AML entity issues a KYC credential, C_1 , that testifies to claim $T_1 :=$ “Alice, with
 3112 identifier X_1 , has NO relation to a (set of) blacklisted organization(s)”

3113 (c) The Bank issues a net-worth credential, C_2 , that testifies to claim $T_2 :=$ “Alice has a net
 3114 worth of V_{Alice} ”

3115 2. Alice then proves to Bob Inc. that:

3116 (a) “Alice’s identifier, X_{Bob} , is related to the identifiers, X_i for $i = 0, 1, 2$ that are connected
 3117 to the confidential credentials C_i ”

3118 (b) “I know the credentials, which are the preimage of some commitment, C_i , were issued
 3119 by the legitimate issuers”

3120 (c) “The credentials, which are the preimage of some commitment, C_i , that exist in an
 3121 accumulator, U , satisfy the three statements T_i ”

3122 **Instantiation details.** Depending on the options in the tables above, the following have been used:

3123 • Holder identification: we instantiate the identifiers as a unique anonymous identifier (pub-
 3124 lickey)

3125 • Issuance identification: the identity of the issuers is known to all the participants, who can
 3126 publicly verify (using public signature verification keys that are hard coded into the circuit)
 3127 the signature on the credentials they issue.

3128 • Credential issuance: credentials are issued by publishing a signed commitment to a positive
 3129 accumulator and sharing the credential in the clear to Alice.

3130 • Credential revocation: is done by removing the commitment of credential from a dynamic and
 3131 positive accumulator. Alice must prove membership of commitment to show her credential
 3132 was not revoked.

3133 • Credential verification: Bob Inc. then verifies the cryptographic proof with the instance.

3134 Note that the transfer of company shares as well as the issuance of company shares is outside of the
 3135 scope of this use-case, but one could use the “Asset Transfer” section of this document to provide
 3136 that functionality.

3137 On another note, the fact that the proving and verification keys were validated by the SEC is an
 3138 assurance to Bob Inc. that proof verification implies Alice is an accredited investor.

3139 **The Predicate:**

- 3140 • Blue = publicly visible in protocol / statement
- 3141 • Red = secret witness, potentially shared between parties when proving

3142 **Definitions / Notation:**

3143 Public state: **Accumulator**, for issuance and revocation, which includes all the commitments to the
 3144 credentials.

3145 **ConfCred** = Commitment to Cred = { **Revoke**, **certificateType**, **publicKey**, **Attribute(s)** }

3146 Where, again, the IRS, AML and Bank are authorities with well-known public keys. Alice’s **pub-**
 3147 **licKey** is her long term public key and one cannot create a new credential unless her long term ID
 3148 has been endorsed. The goal of the scheme is for the holder to create a fresh **proof of confidential**
 3149 **aggregated credentials to the claim of accredited investor**.

3150 IRS issues a **ConfCred_{IRS}** = Commitment(**openIRS**, **revokeIRS**, “IRS”, **myID**, **\$Income**, **b**), **sigIRS**
 3151 AML issues **ConfCred_{AML}** = Commitment(**openAML**, **revokeAML**, “AML”, **myID**, “OK”), **sigAML**

3152 Holder generates a fresh public key **freshCred** to serve as an ephemeral blinded aggregate credential,
 3153 and a ZKP of the following:

3154 **ZkPoK**{ (witness: **myID**, **ConfCred_{IRS}**, **ConfCred_{AML}**, **sigIRS**, **sigAML**, **\$Income**, , **mySig**, **openIRS**,
 3155 **openAML** statement: **freshCred**, **minIncomeAccredited**) : Predicate:

- 3156 - **ConfCred_{IRS}** is a commitment to the IRS credential (**open_{IRS}**, “IRS”, **myID**, **\$Income**)
- 3157 - **ConfCred_{AML}** is the AML credential to (**open_{AML}**, “AML”, **myID**, “OK”)
- 3158 - **\$Income** >= **minIncomeAccredited**
- 3159 - **b** = 1 = “myID paid full taxes”
- 3160 - **mySig** is a signature on **freshCred** for **myID**
- 3161 - **ProveNonRevoke**()
- 3162 }

3163 Present the credential to relying party: **freshCred** and **zkp**.

3164 **ProveNonRevoke**(rhIRS, w_hrIRS, rhAML, w_hrAML, a_IRS

3165 • **revokeIRS**: revocation handler from IRS. Can be embedded in **ConfCred_{IRS}** as an attribute;

3166 it is used to handle revocations.

3167 • **wit_{rhIRS}**: accumulator witness of **revokeIRS**.

3168 • **revokeAML**: revocation handler from AML. Can be embedded in **ConfCred_{AML}** as an at-

3169 tribute; it is used to handle revocations.

3170 • **wit_{rhAML}**: accumulator witness of **revokeAML**.

3171 • **acc_{IRS}**: accumulator for IRS.

3172 • **CommRevoke_{IRS}**: commitment to **revokeIRS**. The holder generates a new commitment for

3173 each revocation to avoid linkability of proofs.

3174 • **acc_{AML}**: accumulator for AML.

3175 • **CommRevoke_{AML}**: commitment to **revokeAML**. The holder generates a new commitment for

3176 each revocation to avoid linkability of proofs.

3177 **ZkPoK**{ (witness: **rhIRS**, **open_{rhIRS}**, **w_{rhIRS}**, **rhAML**, **open_{rhAML}**, **w_{rhAML}** || statements: **C_{IRS}**, **a_{IRS}**,

3178 **C_{AML}**, **a_{AML}**): Predicate:

3179 • **C_{IRS}** is valid commitment to (**open_{rhIRS}**, **rhIRS**)

3180 • **rhIRS** is part of accumulator **a_{IRS}**, under witness **w_{rhIRS}**

3181 • **rhIRS** is an attribute in **Cert_{IRS}**

3182 • **C_{AML}** is valid commitment to (**open_{rhAML}**, **rhAML**)

3183 • **rhAML** is part of accumulator **a_{AML}**, under witness **w_{rhAML}**

3184 • **rhAML** is an attribute in **Cert_{AML}**

3185 }

3186 • myCred is unassociated with myID, with sigIRS, sigAML etc.

3187 • Withstands partial compromise: even if IRS leaks myID and sigIRS, it cannot be used to

3188 reveal the sigAML or associated myID with myCred

3189 4.6 Asset Transfer

3190 4.6.1 Privacy-preserving asset transfers and balance updates

3191 In this section, we examine two use-cases involving using ZK Proofs (ZKPs) to facilitate private

3192 asset-transfer for transferring fungible or non-fungible digital assets. These use-cases are motivated

3193 by privacy-preserving cryptocurrencies, where users must prove that a transaction is valid, without

3194 revealing the underlying details of the transaction. We explore two different frameworks, and

3195 outline the technical details and proof systems necessary for each.

3196 There are two dominant paradigms for tracking fungible digital assets, tracking ownership of assets
 3197 individually, and tracking account balances. The Bitcoin system introduced a form of asset-tracking
 3198 known as the UTXO model, where Unspent Transaction Outputs correspond roughly to single-use
 3199 “coins”. Ethereum, on the other hand, uses the balance model, and each account has an associated
 3200 balance, and transferring funds corresponds to decrementing the sender’s balance, and incrementing
 3201 the receiver’s balance accordingly.

3202 These two different models have different privacy implications for users, and have different rules
 3203 for ensuring that a transaction is valid. Thus the requirements and architecture for building ZK
 3204 proof systems to facilitate privacy-preserving transactions are slightly different for each model, and
 3205 we explore each model separately below.

3206 In its simplest form, the asset-tracking model can be used to track non-fungible assets. In this
 3207 scenario, a transaction is simply a transfer of ownership of the asset, and a transaction is valid if:
 3208 the sender is the current owner of the asset. In the balance model (for fungible assets), each account
 3209 has a balance, and a transaction decrements the sender’s account balance while simultaneously
 3210 incrementing the receivers. In a “balance” model, a transaction is valid if 1) The amount the
 3211 sender’s balance is decremented is equal to the amount the receiver’s balance is incremented, 2)
 3212 The sender’s balance remains non-negative 3) The transaction is signed using the sender’s key.

3213 4.6.2 Zero-Knowledge Proofs in the asset-tracking model

3214 In this section, we describe a simple ZK proof system for privacy-preserving transactions in the
 3215 asset-tracking (UTXO) model. The architecture we outline is essentially a simplification of the
 3216 ZCash system. The primary simplification is that we assume that each asset (“coin”) is indivisible.
 3217 In other words, each asset has an owner, but there is no associated value, and a transaction is
 3218 simply a transfer of ownership of the asset.

3219 **Motivation:** Allow stakeholders to transfer non-fungible assets, without revealing the ownership
 3220 of the assets publicly, while ensuring that assets are never created or destroyed.

3221 **Parties:** There are three types of parties in this system: a Sender, a Receiver and a distributed
 3222 set of validators. The sender generates a transactions and a proof of validity. The (distributed)
 3223 validators act as verifiers and check the validity of the transaction. The receiver has no direct role,
 3224 although the sender must include the receiver’s public-key in the transaction.

3225 **What is being proven:** At high level, the sender must prove three things to convince the validators
 3226 that a transaction is valid.

- 3227 • The asset (or “note”) being transferred is owned by the sender. (Each asset is represented by
 3228 a unique string)
- 3229 • The sender proves that they have the private spending keys of the input notes, giving them
 3230 the authority to send asset.
- 3231 • The private spending keys of the input assets are cryptographically linked to a signature over
 3232 the whole transaction, in such a way that the transaction cannot be modified by a party who
 3233 did not know these private keys.

3234 **What information is needed by the verifier:**

- 3235 • The verifiers need access to the CRS used by the proof system
- 3236 • The validators need access to the entire history of transactions (this includes all UTXOs,
- 3237 commitments and nullifiers as described later). This history can be stored on a distributed
- 3238 ledger (e.g. the Bitcoin blockchain)

3239 **Possible attacks:**

- 3240 • CRS compromise: If an attacker learns the private randomness used to generate the CRS,
- 3241 the attacker can forge proofs in the underlying system
- 3242 • Ledger attacks: validating a transaction requires reading the entire history of transactions,
- 3243 and thus a verifier with an incorrect view of the transaction history may be convinced to
- 3244 accept an incorrect transaction as valid.
- 3245 • Re-identification attacks: The purpose of incorporating ZKPs into this system is to facilitate
- 3246 transactions without revealing the identities of the sender and receiver. If anonymity is not
- 3247 required, ZKPs can be avoided altogether, as in Bitcoin. Although this system hides the
- 3248 sender and receiver of each transaction, the fact that a transaction occurred (and the time of
- 3249 its occurrence) is publicly recorded, and thus may be used to re-identify individual users.
- 3250 • IP-level attacks: by monitoring network traffic, an attacker could link transactions to spe-
- 3251 cific senders or receivers (each transaction requires communication between the sender and
- 3252 receiver) or link public-keys (pseudonyms) to real-world identities
- 3253 • Man-it-the-Middle attacks: An attacker could convince a sender to transfer an asset to an
- 3254 “incorrect” public-key

3255 **Setup scenario:** This system is essentially a simplified version of Zcash proof system, modified
3256 for indivisible assets. Each asset is represented by a unique AssetID, and for simplicity we assume
3257 that the entire set of assets has been distributed, and no assets are ever created or destroyed.

3258 At any given time, the public state of the system consists of a collection of “asset notes”. These notes
3259 are stored as leaves in a Merkle Tree, and each leaf represents a single indivisible asset represented
3260 by unique assetID. In more detail, a “note” is a commitment to {Nullifier, publicKey, assetID},
3261 indicating that publicKey “owns” assetID.

3262 **Main transaction type:** Sending an asset from Current Owner *A* to New Owner *B*

3263 **Security goals:**

- 3264 • Only the current owner can transfer the asset
- 3265 • Assets are never created or destroyed

3266 **Privacy goals:** Ideally, the system should hide all information about the ownership and transaction
3267 patterns of the users. The system sketched below does not attain that such a high-level of privacy,
3268 but instead achieves the following privacy-preserving features

- 3269 • Transactions are publicly visible, i.e., anyone can see that a transaction occurred

- 3270 • Transactions do not reveal which asset is being transferred
- 3271 • Transactions do not reveal the identities (public-keys) of the sender or receiver.
- 3272 – Limitation: Previous owner can tell when the asset is transferred. (Mitigation: after
- 3273 receiving asset, send it to yourself)

3274 **Details of a transfer:** Each transaction is intended to transfer ownership of an asset from a

3275 Current Owner to a New Owner. In this section, we outline the proofs used to ensure the validity

3276 of a transaction. Throughout this description, color blue denotes data that is globally and publicly

3277 visible in the protocol / statement. Color red denotes private data, e.g., a secret witness held by

3278 the prover or information shared between the Current Owner and New Owner.

3279 The Current Owner, A , has the following information

- 3280 • A **publicKey** and corresponding **secretKey**
- 3281 • An **assetID** corresponding to the asset being transferred
- 3282 • A **note** in the MerkleTree corresponding to the asset
- 3283 • Knows how to open the **commitment** (**Nullifier**, **assetID**, **publicKey**) **publicKeyOut** of the new
- 3284 Owner B

3285 The Current Owner, A , generates

- 3286 • A new **NullifierOut**
- 3287 • A new commitment **commitment** (**NullifierOut**, **assetID**, **publicKey**)

3288 The Current owner, A , sends

- 3289 • Privately to B : **NullifierOut**, **publicKeyOut**, **assetID**
- 3290 • Publicly to the blockchain: **Nullifier**, **comOut**, **ZKProof** (the structure of **ZKProof** is outlined
- 3291 below)

3292 If **Nullifier** does not exist in **MerkleTree** and **ZKProof** validates, then **comOut** is added to the

3293 merkleTree.

3294 **The structure of the Zero-Knowledge Proof:** We use a modification of Camenisch-Stadler nota-

3295 tion to describe the describe the structure of the proof.

3296 Public state: **MerkleTree** of Notes: Note = **Commitment** to { **Nullifier**, **publicKey**, **assetID** }

3297 **ZKProof** = $ZkPoK_{pp}\{$

3298 (witness: **publicKey**, **publicKeyOut**, **merkleProof**, **NullifierOut**, **com**, **assetID**, sig

3299 statement: **MerkleTree**, **Nullifier**, **comOut**) :

3300 predicate:

- 3301 – **com** is included in **MerkleTree** (using **merkleProof**)
- 3302 – **com** is a commitment to (**Nullifier**, **publicKey**, **assetID**)
- 3303 – **comOut** is a commitment to (**NullifierOut**, **publicKeyOut**, **assetID**)

3304 – sig is a signature on comOut for publicKey
 3305 }

3306 4.6.3 Zero-Knowledge proofs in the balance model

3307 In this section, we outline a simple system for privately transferring fungible assets, in the “balance
 3308 model.” This system is essentially a simplified version of zkLedger. The state of the system is an
 3309 (encrypted) account balance for each user. Each account balance is encrypted using an additively
 3310 homomorphic cryptosystem, under the account-holder’s key. A transaction decrements the sender’s
 3311 account balance, while incrementing the receiver’s account by a corresponding amount. If the
 3312 number of users is fixed, and known in advance, then a transaction can hide all information about
 3313 the sender and receiver by simultaneously updating all account balances. This provides a high-
 3314 degree of privacy, and is the approach taken by zkLedger. If the set of users is extremely large,
 3315 dynamically changing, or unknown to the sender, the sender must choose an “anonymity set” and
 3316 the transaction will reveal that it involved members of the anonymity set, but not the amount of the
 3317 transaction or which members of the set were involved. For simplicity of presentation, we assume
 3318 a model like zkLedger’s where the set of parties in the system is fixed, and known in advance, but
 3319 this assumption does not affect the details of the zero-knowledge proofs involved.

3320 **Motivation:** Each entity maintains a private account balance, and a transaction decrements the
 3321 sender’s balance and increments the receiver’s balance by a corresponding amount. We assume that
 3322 every transaction updates every account balance, thus all information the origin, destination and
 3323 value of a transaction will be completely hidden. The only information revealed by the protocol is
 3324 the fact that a transaction occurred.

3325 **Parties:**

- 3326 • A set of n stakeholders who wish to transfer fungible assets anonymously
- 3327 • The stakeholder who initiates the transaction is called the “prover” or the “sender”
- 3328 • The receiver, or receivers do not have a distinguished role in a transaction
- 3329 • A set of validators who maintain the (public) state of the system (e.g. using a blockchain or
 3330 other DLT).

3331 **What is being proved:** The sender must convince the validators that a proposed transaction is
 3332 “valid” and the state of the system should be updated to reflect the new transaction. A transaction
 3333 consists of a set of n ciphertexts, (c_1, \dots, c_n) , and where $c_i = \text{Enc}_{pk}(x_i)$, and a transaction is valid if:

- 3334 • The sum of all committed values is 0 (i.e., $x_1 + \dots + x_n = 0$)
- 3335 • The sender owns the private key corresponding to all negative x_i
- 3336 • After the update, all account balances remain positive

3337 What information is needed by the verifier:

- 3338 • The verifiers need access to the CRS used by the proof system

- 3339 • The verifiers need access to the current state of the system (i.e., the current vector of n
3340 encrypted account balances). This state can be stored on a distributed ledger

3341 Possible attacks:

- 3342 • CRS compromise: If an attacker learns the private randomness used to generate the CRS,
3343 the attacker can forge proofs in the underlying system
- 3344 • Ledger attacks: validating a transaction requires knowing the current state of the system
3345 (encrypted account balances), thus a validator with an incorrect view of the current state
3346 may be convinced to accept an incorrect transaction as valid.
- 3347 • Re-identification attacks: The purpose of incorporating ZKPs into this system is to facilitate
3348 transactions without revealing the identities of the sender and receiver. If anonymity is not
3349 required, ZKPs can be avoided altogether, as in Bitcoin. Although this system hides the
3350 sender and receiver of each transaction, the fact that a transaction occurred (and the time of
3351 its occurrence) is publicly recorded, and thus may be used to re-identify individual users.
- 3352 • IP-level attacks: by monitoring network traffic, an attacker could link transactions to specific
3353 senders or receivers (each transaction requires communication between the sender and the
3354 validators) or link public-keys (pseudonyms) to real-world identities
- 3355 • Man-it-the-Middle attacks: An attacker could convince a sender to transfer an asset to an
3356 “incorrect” public-key. This is perhaps less of a concern in the situation where the user-base
3357 is static, and all public-keys are known in advance.

3358 **Setup scenario:** There are fixed number of users, n . User i has a known public-key, pk_i . Each
3359 user has an account balance, maintained as an additively homomorphic encryption of their current
3360 balance under their pk . Each transaction is a list of n encryptions, corresponding to the amount
3361 each balance should be incremented or decremented by the transaction. To ensure money is never
3362 created or destroyed, the plaintexts in an encrypted transaction must sum to 0. We assume that
3363 all account balance are initialized to non-negative values.

3364 **Main transaction type:** Transferring funds from user i to user j

3365 **Security goals:**

- 3366 • An account balance can only be decremented by the owner of that account
- 3367 • Account balances always remain non-negative
- 3368 • The total amount of money in the system remains constant

3369 **Privacy goals:** Ideally, the system should hide all information about the ownership and transaction
3370 patterns of the users. The system sketched below does not attain that such a high-level of privacy,
3371 but instead achieves the following privacy-preserving features:

- 3372 • Transactions are publicly visible, i.e., anyone can see that a transaction occurred
- 3373 • Transactions do not reveal which asset is being transferred
- 3374 • Transactions do not reveal the identities (public-keys) of the sender or receiver.

3375 Limitation: transaction times are leaked

3376 **Details of a transfer:** Each transaction is intended to update the current account balances in the
 3377 system. Next, we outline the proofs used to ensure the validity of a transaction. Expressions in blue
 3378 denote information that is globally and publicly visible in the protocol / statement; expressions in
 3379 red denote private information, e.g. a secret witness held by the prover.

3380 The Sender, A , has the following information

- 3381 • Public keys pk_1, \dots, pk_n
- 3382 • $secretKey_i$ corresponding to $publicKey_i$, and a values x_j , to transfer to user j
- 3383 • The sender's own current account balance, y_i

3384 The Sender, A , generates

- 3385 • a vector of ciphertexts, C_1, \dots, C_n with $C_t = \text{Enc}_{pk_t}(x_t)$

3386 The Sender, A , sends

- 3387 • The vector of ciphertexts C_1, \dots, C_n and $ZKProof$ (described below) to the blockchain

3388 ZK Circuit:

3389 Public state: The current state of the system, i.e., a vector of (encrypted) account balances,
 3390 B_1, \dots, B_n .

3391 $ZKProof = \text{ZkPoK}_{pp}\{$ (witness: i, x_1, \dots, x_n, sk statement: C_1, \dots, C_n) :

3392 predicate:

- 3393 • C_t is an encryption to x_t under public key pk_t for $t = 1, \dots, n$
- 3394 • $x_1 + \dots + x_n = 0$
- 3395 • $x_t \geq 0$ OR sk corresponds to pk_t for $t = 1, \dots, n$
- 3396 • $x_t \geq 0$ OR current balance B_t encrypts a value no smaller than $|x_t|$ for $t = 1, \dots, n$

3397 }

3398 4.7 Regulation Compliance

3399 4.7.1 Overview

3400 An important pattern of applications in which zero-knowledge protocols are useful is within settings
 3401 in which a regulator wishes to monitor, or assess the risk related to some item managed by a
 3402 regulated party. One such example can be whether or not taxes are being paid correctly by an
 3403 account holder, or is a bank or some other financial entity solvent, or even stable.

3404 The regulator in such cases is interested in learning “the bottom line”, which is typically derived
 3405 from some aggregate measure on more detailed underlying data, but does not necessarily need to
 3406 know all the details. For example, the answer to the question of “did the bank take on too many
 3407 loans?” Is eventually answered by a single bit (Yes/No) and can be answered without detailing every
 3408 single loan provided by the bank and revealing recipients, their income, and other related data.

3409 Additional examples of such scenarios include checking that:

- 3410 • taxes have been properly paid by some company or person.
- 3411 • a given loan is not too risky.
- 3412 • data is retained by some record keeper (without revealing or transmitting the data)
- 3413 • an airplane has been properly maintained and is fit to fly

3414 The use of Zero knowledge proofs can then allow the generation of a proof that demonstrate the
 3415 correctness of the aggregate result. The idea is to show something like the following statement:
 3416 There is a commitment (possibly on a blockchain) to records that show that the result is correct.

3417 **Trusting data fed into the computation:** In order for a computation on hidden data to prove valu-
 3418 able, the data that is fed in must be grounded as well. Otherwise, proving the correctness of the com-
 3419 putation would be meaningless. To make this point concrete: A credit score that was computed from
 3420 some hidden data can be correctly computed from some financial records, but when these records are
 3421 not exposed to the recipient of the proof, how can the recipient trust that they are not fabricated?

3422 Data that is used for proofs should then generally be committed to by parties that are separate
 3423 from the prover, and that are not likely to be colluding with the prover. To continue our example
 3424 from before: an individual can prove that she has a high credit score based on data commitments
 3425 that were produced by her previous lenders (one might wonder if we can indeed trust previous
 3426 lenders to accurately report in this manner, but this is in fact an assumption implicitly made in
 3427 traditional credit scoring as well).

3428 The need to accumulate commitments regarding the operation and management of the processes
 3429 that are later audited using zero-knowledge often fits well together with blockchain systems, in which
 3430 commitments can be placed in an irreversible manner. Since commitments are hiding, such publicly
 3431 shared data does not breach privacy, but can be used to anchor trust in the veracity of the data.

3432 **4.7.2 An example in depth: Proof of compliance for aircraft**

3433 An operator is flying an aircraft, and holds a log of maintenance operations on the aircraft. These
 3434 records are on different parts that might be produced by different companies. Maintenance and
 3435 flight records are attested to by engineers at various locations around the world (who we assume
 3436 do not collude with the operator).

3437 The regulator wants to know that the aircraft is allowed to fly according to a certain set of rules.
 3438 (Think of the Volkswagen emissions cheating story.)

3439 The problem: Today, the regulator looks at the records (or has an auditor do so) only once in a
 3440 while. We would like to move to a system where compliance is enforced in “real time”, however,
 3441 this reveals the real-time operation of the aircraft if done naively.

3442 Why is zero-knowledge needed? We would like to prove that regulation is upheld, without revealing
 3443 the underlying operational data of the aircraft which is sensitive business operations. Regulators
 3444 themselves prefer not to hold the data (liability and risk from loss of records), prefer to have
 3445 companies self-regulate to the extent possible.

3446 What is the threat model beyond the engineers/operator not colluding? What about the parts

3447 manufacturers? Regulators? Is there an antagonistic relationship between the parts manufacturers?
3448 This scheme will work on regulation that isn't vague, such as aviation regulation. In some cases,
3449 the rules are vague on purpose and leave room for interpretation.

3450 4.7.3 Protocol high level

3451 **Parties:**

- 3452 • Operator / Party under regulation: performs operations that need to comply to a regulation.
3453 For example an airline operator that operates aircrafts
- 3454 • Risk bearer / Regulator : verifies that all regulated parties conform to the rules; updates the
3455 rules when risks evolve. For example, the FAA regulates and enforces that all aircrafts to
3456 be airworthy at all times. For an aircraft owner leasing their assets, they want to know that
3457 operation and maintenance does not degrade their asset. Same for a bank that financed an
3458 aircraft, where the aircraft is the collateral for the financing.
- 3459 • Issuer / 3rd party attesting to data: Technicians having examined parts, flight controllers
3460 attesting to plane arriving at various locations, embarked equipment providing signed readings
3461 of sensors.

3462 **What the operator is proving:**

- 3463 • proves to the regulator that the latest maintenance data indicates the aircraft is airworthy
- 3464 • proves to the bank that the aircraft maintenance status means it is worth a given value,
3465 according to a formula provided by that bank

3466 **What are the privacy requirements:**

- 3467 • Operator: does not reveal details of operations and assets maintenance status to competition.
- 3468 • Aircraft identity: must be kept anonymous from all parties, except regulators and technicians.
- 3469 • Technician's identity: must be kept anonymous from the regulator, but if needed the operator
3470 can be asked to open the commitments for the regulator to validate the reports.

3471 **The proof predicate:** "The operator is the owner of the aircraft, and knows some signed data
3472 attesting to the compliance with regulation rules: all the components are safe to fly".

- 3473 • The plane is made up of the components x_1, \dots, x_n and for each of the components:
 - 3474 – There is a legitimate attestation by an engineer who checked the component, and signed
3475 it's okay.
 - 3476 – The latest attestation by a technician is recent: the timestamp of the check was done
3477 before date D .

3478 **What is the public / private data:**

- 3479 • **Private:** (i) identity of the operator; (ii) airplane record; (iii) examination report of the
3480 technicians; (iv) identity of the technician who signed the report.
- 3481 • **Public:** Commitment to airplane record.

3482 The following is committed to a public ledger: a record for the airplane, including miles flown;
 3483 records attesting to repairs / inspections by mechanics. The decommitment is communicated to
 3484 the operator. These records reference the identifier of the plane.

3485 Whenever the plane flies, the old plane record needs to be invalidated, and a new one committed
 3486 with extra mileage. When a proof of “airworthiness” is required, the operator proves that for
 3487 each part, the mileage is below what requires replacement, or that an engineer replaced the part
 3488 (pointing to a record committed by a technician).

3489 **At the gadget level:**

- 3490 • The prover proves knowledge of a de-commitment of an airplane record (decommitment)
- 3491 • The record is in the set of records on the blockchain (set membership)
- 3492 • and knowledge of de-commitments for records for the parts (decommitment) that are also in
 3493 the set of commitments on the ledger (set membership)
- 3494 • The airplane record is not revoked, i.e., it is the most recent one (set non-membership for the
 3495 set of published nullifiers)
- 3496 • The plane id noted in the parts is the same as the plane id in the plane record (equality)
- 3497 • The mileage of the plane is lower than the mileage needed to replace each part (range proofs)
 3498 OR OTHERWISE there exists a record (set membership) stating that the part was replaced
 3499 by a technician; validate signature of the technician (maybe use ring signature outside of ZK?)

3500 **4.8 Conclusions**

- 3501 • The asset transfer and regulation can be used in the identity framework in a way that the
 3502 additions complete the framework.
- 3503 • External oracles such as blockchain used for storing reference to data commitments

Acknowledgments

The development of this community reference counts with the support of numerous individuals.

Version 0. The “proceedings” of the 1st ZKProof workshop (Boston, May 2018) formed the initial basis for this document. The contributions were organized in three tracks:

- **Implementation track.** Chairs: Sean Bowe, Kobi Gurkan, Eran Tromer. Participants: Benedikt Bünz, Konstantinos Chalkias, Daniel Genkin, Jack Grigg, Daira Hopwood, Jason Law, Andrew Poelstra, abhi shelat, Muthu Venkitasubramaniam, Madars Virza, Riad S. Wahby, Pieter Wuille.
- **Applications Track.** Chairs: Daniel Benarroch, Ran Canetti, Andrew Miller. Participants: Shashank Agrawal, Tony Arcieri, Vipin Bharathan, Josh Cincinnati, Joshua Daniel, Anuj Das Gupta, Angelo De Caro, Michael Dixon, Maria Dubovitskaya, Nathan George, Brett Hemenway Falk, Hugo Krawczyk, Jason Law, Anna Lysyanskaya, Zaki Manian, Eduardo Morais, Neha Narula, Gavin Pacini, Jonathan Rouach, Kartheek Solipuram, Mayank Varia, Douglas Wikstrom, Aviv Zohar.
- **Security track.** Chairs: Jens Groth, Yael Kalai, Muthu Venkitasubramaniam. Participants: Nir Bitansky, Ran Canetti, Henry Corrigan-Gibbs, Shafi Goldwasser, Charanjit Jutla, Yuval Ishai, Rafail Ostrovsky, Omer Paneth, Tal Rabin, Maryana Raykova, Ron Rothblum, Alessandra Scafuro, Eran Tromer, Douglas Wikström.

Version 0.1. Prior to the 2nd ZKProof workshop, the ZKProof organization team requested feedback from NIST about the developed documentation. The NIST PEC team (Luís Brandão, René Peralta, Angela Robinson) then elaborated the “NIST comments on the initial ZKProof documentation” [BPR19] with 28 comments/suggestions for subsequent development of a “Community Reference Document”. Luís Brandão ported to LaTeX the proceedings into a LaTeX version, along with inline comments, which became named as version 0.1.

Version 0.2. The contributions from version 0.1 to version 0.2 followed the editorial process initiated at the 2nd ZKProof Workshop (Berkeley, April 2019). Several suggested contributions stemmed from the breakout discussions in the workshop, which were possible by the collaboration of *scribes*, *moderators* and *participants*, as documented in the Workshop Notes [ZKP19]. The actual content contributions were developed thereafter by several *contributors*, including Yu Hang, Eduardo Morais, Justin Thaler, Ivan Visconti, Riad Wahby and Yupeng Zhang, besides the NIST PEC team (Luís Brandão, René Peralta, Angela Robinson) and the Editors team (Daniel Benarroch, Luís Brandão, Eran Tromer). The detailed description of the changes, contributions and contributors appears in the “diff” version of the community reference.

Version 0.3. The main update was the revision of the [paradigms](#) chapter. The old section “Taxonomy of Constructions” was replaced by a new section with “[background](#)” and then two sections on “[IT Proof Systems](#)” and “[Cryptographic Compilers](#)” were added. This change was inspired by, and mostly adapted from, two blogposts titled “Zero-Knowledge Proofs from Information-Theoretic Proof Systems” (parts 1 and 2) by Yuval Ishai [Ish20]. The content in the [IP based](#) (under [Linear IOP](#)) subsection was provided by Justin Thaler. The revised chapter was structured and edited by

3543 the ZKProof Editors team (Daniel Benarroch, Luís Brandão, Mary Maller, Eran Tromer). Some
3544 editorial corrections were externally suggested, including during the public review phase, as de-
3545 scribed in the “Call for contributions to the ZKProof Community Reference — Review cycle 2020:
3546 from version 0.2 to 0.3.” [ZKP-CC20]. Suggestions from Jens Groth improved Table 1.1 in Sec-
3547 tion 1.2. Thanks for several other comments from various readers. The integration of some received
3548 suggestions is left for future versions.

3549 **Miscellaneous.** A general “thank you” goes to all who have so far collaborated with the ZKProof
3550 initiative. This includes the workshop speakers, participants, organizers and sponsors, as well as the
3551 ZKProof steering committee and program committee members, and the participants in the online
3552 ZKProof forum. Detailed information about ZKProof is available on the zkproof.org website.

References

- 3554 [AHIKV17] B. Applebaum, N. Haramaty-Krasne, Y. Ishai, E. Kushilevitz, and V. Vaikuntana-
 3555 than. “Low-Complexity Cryptographic Hash Functions.” In: *8th Innovations in Theoretical Com-*
 3556 *puter Science Conference (ITCS 2017)*. Ed. by C. H. Papadimitriou. Vol. 67. Leibniz International
 3557 Proceedings in Informatics (LIPIcs). Pub. by *Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik*,
 3558 2017, 7:1–7:31. DOI: [10.4230/LIPIcs.ITCS.2017.7](https://doi.org/10.4230/LIPIcs.ITCS.2017.7). Also at ia.cr/2017/036 (Cited on p. 35).
- 3559 [AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. “Ligero: Lightweight Sublin-
 3560 ear Arguments Without a Trusted Setup.” In: *Proc. 2017 ACM SIGSAC Conference on Computer*
 3561 *and Communications Security*. CCS ’17. Pub. by ACM, 2017, pp. 2087–2104. DOI: [10.1145/3133956.](https://doi.org/10.1145/3133956.3134104)
 3562 [3134104](https://doi.org/10.1145/3133956.3134104) (Cited on p. 41).
- 3563 [AW09] S. Aaronson and A. Wigderson. “Algebrization: A New Barrier in Complexity Theory.”
 3564 In: *ACM Trans. Comput. Theory* 1.1 (February 2009). DOI: [10.1145/1490270.1490272](https://doi.org/10.1145/1490270.1490272). Also at
 3565 ECCC [TR08/005](https://eccc.weizmann.ac.il/report/2009/005) (Cited on p. 34).
- 3566 [BBBF18] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. “Verifiable Delay Functions.” In: *Ad-*
 3567 *vances in Cryptology — CRYPTO 2018*. Ed. by H. Shacham and A. Boldyreva. Pub. by *Springer*
 3568 *International Publishing*, 2018, pp. 757–788. Also at ia.cr/2018/601 (Cited on p. 60).
- 3569 [BBCGI19] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. “Zero-Knowledge
 3570 Proofs on Secret-Shared Data via Fully Linear PCPs.” In: *Advances in Cryptology — CRYPTO*
 3571 *2019*. Ed. by A. Boldyreva and D. Micciancio. Pub. by *Springer International Publishing*, 2019,
 3572 pp. 67–97. DOI: [10.1007/978-3-030-26954-8_3](https://doi.org/10.1007/978-3-030-26954-8_3). Also at ia.cr/2019/188 (Cited on pp. 32, 34, 40).
- 3573 [BBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. “Fast Reed-Solomon Interactive
 3574 Oracle Proofs of Proximity.” In: *45th International Colloquium on Automata, Languages, and Pro-*
 3575 *gramming (ICALP 2018)*. Ed. by I. Chatzigiannakis, C. Kaklamani, D. Marx, and D. Sannella.
 3576 Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Pub. by *Schloss Dagstuhl–*
 3577 *Leibniz-Zentrum fuer Informatik*, 2018, 14:1–14:17. DOI: [10.4230/LIPIcs.ICALP.2018.14](https://doi.org/10.4230/LIPIcs.ICALP.2018.14). Also at
 3578 ECCC [TR17/134](https://eccc.weizmann.ac.il/report/2018/134) (Cited on p. 40).
- 3579 [BBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. *Scalable, transparent, and post-*
 3580 *quantum secure computational integrity*. IACR Cryptology ePrint Archive, ia.cr/2018/046. 2018
 3581 (Cited on p. 40).
- 3582 [BCCGLRT17] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinfeld, and
 3583 E. Tromer. “The Hunting of the SNARK.” In: *Journal of Cryptology* 30 (2017), pp. 989–1066.
 3584 DOI: [10.1007/s00145-016-9241-9](https://doi.org/10.1007/s00145-016-9241-9). Also at ia.cr/2014/580. Merging of (i) “From extractable
 3585 collision resistance to succinct non-interactive arguments of knowledge, and back again”, ITCS ’12,
 3586 pp. 326–349, ACM, 2012, DOI:[10.1145/2090236.2090263](https://doi.org/10.1145/2090236.2090263), ia.cr/2011/443, and (ii) “Delegation of
 3587 Computation without Rejection Problem from Designated Verifier CS-Proofs”, 2011, ia.cr/2011/456
 3588 (Cited on p. 21).
- 3589 [BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. “Recursive Composition and Boot-
 3590 strapping for SNARKS and Proof-carrying Data.” In: *45th ACM Symposium on the Theory of*
 3591 *Computing*. STOC ’13. Pub. by ACM, 2013, pp. 111–120. DOI: [10.1145/2488608.2488623](https://doi.org/10.1145/2488608.2488623). Also at
 3592 ia.cr/2012/095 (Cited on pp. 42, 65).

- 3593 [BCDEK+14] P. Bichsel, J. Camenisch, M. Dubovitskaya, R. R. Enderlein, S. Krenn, I. Krontiris,
 3594 A. Lehmann, G. Neven, J. D. Nielsen, C. Paquin, F.-S. Preiss, K. Rannenberg, A. Sabouri, and
 3595 M. Stausholm. *D2.2 — Architecture for Attribute-based Credential Technologies — Final Version*.
 3596 Ed. by A. Sabour. August 2014. https://abc4trust.eu/download/Deliverable_D2.2.pdf (Cited on
 3597 p. 67).
- 3598 [BCDLRSY17] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K.
 3599 Samelin, and S. Yakubov. “Accumulators with Applications to Anonymity-Preserving Revocation.”
 3600 In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. April 2017, pp. 301–315.
 3601 DOI: [10.1109/EuroSP.2017.13](https://doi.org/10.1109/EuroSP.2017.13). Also at ia.cr/2017/043 (Cited on pp. 67, 75).
- 3602 [BCGGHJ17] J. Bootle, A. Cerulli, E. Ghadafi, J. Groth, M. Hajiabadi, and S. K. Jakobsen.
 3603 “Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability.” In: *Advances in Cryptology — ASIACRYPT 2017*. Ed. by T. Takagi and T. Peyrin. Pub. by Springer International
 3604 Publishing, 2017, pp. 336–365. DOI: [10.1007/978-3-319-70700-6_12](https://doi.org/10.1007/978-3-319-70700-6_12). Also at ia.cr/2017/872 (Cited
 3605 on pp. 32, 35, 41).
- 3607 [BCGGMTV14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and
 3608 M. Virza. “Zerocash: Decentralized Anonymous Payments from Bitcoin.” In: *Proc. 2014 IEEE
 3609 Symposium on Security and Privacy*. SP ’14. May 2014, pp. 459–474. DOI: [10.1109/SP.2014.36](https://doi.org/10.1109/SP.2014.36).
 3610 Also at ia.cr/2014/349 (Cited on pp. 56, 67).
- 3611 [BCGGRS19] E. Ben-Sasson, A. Chiesa, L. Goldberg, T. Gur, M. Riabzev, and N. Spooner.
 3612 “Linear-Size Constant-Query IOPs for Delegating Computation.” In: *Theory of Cryptography*. Ed.
 3613 by D. Hofheinz and A. Rosen. TCC 2019. Pub. by Springer International Publishing, 2019, pp. 494–
 3614 521. DOI: [10.1007/978-3-030-36033-7_19](https://doi.org/10.1007/978-3-030-36033-7_19). Also at ia.cr/2019/1230 (Cited on p. 40).
- 3615 [BCGT13] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. “On the Concrete Efficiency of
 3616 Probabilistically-checkable Proofs.” In: *Proc. 45th Annual ACM Symposium on Theory of Com-
 3617 puting*. STOC ’13. Pub. by ACM, 2013, pp. 585–594. DOI: [10.1145/2488608.2488681](https://doi.org/10.1145/2488608.2488681). Also at
 3618 [ECCC TR12/045](https://eccc.tr12/045) (Cited on p. 43).
- 3619 [BCGTV13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. “SNARKs for C:
 3620 Verifying Program Executions Succinctly and in Zero Knowledge.” In: *Advances in Cryptology —
 3621 CRYPTO 2013*. Ed. by R. Canetti and J. A. Garay. Pub. by Springer Berlin Heidelberg, 2013,
 3622 pp. 90–108. DOI: [10.1007/978-3-642-40084-1_6](https://doi.org/10.1007/978-3-642-40084-1_6). Also at ia.cr/2013/507 (Cited on pp. 28, 50, 51).
- 3623 [BCIKS20] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf. “Proximity Gaps
 3624 for Reed–Solomon Codes.” In: *2020 IEEE 61st Annual Symposium on Foundations of Computer
 3625 Science (FOCS)*. 2020, pp. 900–909. DOI: [10.1109/FOCS46700.2020.00088](https://doi.org/10.1109/FOCS46700.2020.00088). Also at ia.cr/2020/654
 3626 (Cited on p. 40).
- 3627 [BCIOP13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. “Succinct Non-
 3628 interactive Arguments via Linear Interactive Proofs.” In: *Theory of Cryptography*. Ed. by A. Sahai.
 3629 Pub. by Springer Berlin Heidelberg, 2013, pp. 315–333. DOI: [10.1007/978-3-642-36594-2_18](https://doi.org/10.1007/978-3-642-36594-2_18). Also
 3630 at ia.cr/2012718 (Cited on pp. 35, 38, 39).
- 3631 [BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. “Recursive Proof Composition from
 3632 Accumulation Schemes.” In: *TCC 2020: Theory of Cryptography*. Ed. by R. Pass and K. Pietrzak.
 3633 Vol. 12551. LNCS. Pub. by Springer, 2020, pp. 1–18. DOI: [10.1007/978-3-030-64378-2_1](https://doi.org/10.1007/978-3-030-64378-2_1) (Cited on
 3634 p. 42).

References

- 3635 [BCRSVW19] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward.
3636 “Aurora: Transparent Succinct Arguments for R1CS.” In: *Advances in Cryptology — EUROCRYPT*
3637 *2019*. Ed. by Y. Ishai and V. Rijmen. Pub. by *Springer International Publishing*, 2019, pp. 103–128.
3638 DOI: [10.1007/978-3-030-17653-2_4](https://doi.org/10.1007/978-3-030-17653-2_4). Also at ia.cr/2018/828 (Cited on pp. 33, 40).
- 3639 [BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. “Interactive Oracle Proofs.” In: *Theory of*
3640 *Cryptography*. Ed. by M. Hirt and A. Smith. Pub. by *Springer Berlin Heidelberg*, 2016, pp. 31–60.
3641 DOI: [10.1007/978-3-662-53644-5_2](https://doi.org/10.1007/978-3-662-53644-5_2). Also at ia.cr/2016/116 (Cited on pp. 32, 42, 43).
- 3642 [BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. “Scalable Zero Knowledge via
3643 Cycles of Elliptic Curves.” In: *Advances in Cryptology — CRYPTO 2014*. Ed. by J. A. Garay and R.
3644 Gennaro. Pub. by *Springer Berlin Heidelberg*, 2014, pp. 276–294. DOI: [10.1007/978-3-662-44381-1_](https://doi.org/10.1007/978-3-662-44381-1_16)
3645 [16](https://doi.org/10.1007/978-3-662-44381-1_16). Extended version at *Journal Algorithmica* (79), pp. 1102–1160, Dec-2017, doi:[10.1007/s00453-](https://doi.org/10.1007/s00453-016-0221-0)
3646 [016-0221-0](https://doi.org/10.1007/s00453-016-0221-0). Also at ia.cr/2014/595 (Cited on pp. 42, 63, 65).
- 3647 [BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. “Succinct Non-Interactive Zero
3648 Knowledge for a von Neumann Architecture.” In: *USENIX Security 2014*. 2014, pp. 781–796. Also
3649 at ia.cr/2013/879 (Cited on p. 28).
- 3650 [BDFG20] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. *Halo Infinite: Recursive zk-SNARKs*
3651 *from any Additive Polynomial Commitment Scheme*. IACR Cryptology ePrint Archive, [ia.cr/2020/](https://ia.cr/2020/1536)
3652 [1536](https://ia.cr/2020/1536). 2020 (Cited on p. 42).
- 3653 [BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. “Checking Computations in Poly-
3654 logarithmic Time.” In: *Proc. 23rd Annual ACM Symposium on Theory of Computing*. STOC ’91.
3655 Pub. by *Association for Computing Machinery*, 1991, pp. 21–32. DOI: [10.1145 / 103418. 103428](https://doi.org/10.1145/103418.103428)
3656 (Cited on p. 34).
- 3657 [BFS20] B. Bünz, B. Fisch, and A. Szepieniec. “Transparent SNARKs from DARK Compilers.”
3658 In: *Advances in Cryptology — EUROCRYPT 2020*. Ed. by A. Canteaut and Y. Ishai. Pub. by
3659 *Springer International Publishing*, 2020, pp. 677–706. DOI: [10.1007/978-3-030-45721-1_24](https://doi.org/10.1007/978-3-030-45721-1_24). Also at
3660 ia.cr/2019/1229 (Cited on pp. 33, 41).
- 3661 [BGGHKMR90] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and
3662 P. Rogaway. “Everything Provable is Provable in Zero-Knowledge.” In: *Advances in Cryptology —*
3663 *CRYPTO’ 88*. Ed. by S. Goldwasser. Vol. 403. LNCS. Pub. by *Springer New York*, 1990, pp. 37–56.
3664 DOI: [10.1007/0-387-34799-2_4](https://doi.org/10.1007/0-387-34799-2_4) (Cited on p. 39).
- 3665 [BGH19] S. Bowe, J. Grigg, and D. Hopwood. *Recursive Proof Composition without a Trusted*
3666 *Setup*. IACR Cryptology ePrint Archive, ia.cr/2019/1021. 2019 (Cited on p. 42).
- 3667 [BGHSV04] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. “Robust PCPs of
3668 Proximity, Shorter PCPs and Applications to Coding.” In: *Proc. 36th Annual ACM Symposium on*
3669 *Theory of Computing*. STOC ’04. Pub. by *Association for Computing Machinery*, 2004, pp. 1–10.
3670 DOI: [10.1145/1007352.1007361](https://doi.org/10.1145/1007352.1007361). Also at ECCO [TR04/021](https://ia.cr/2004/021) (Cited on p. 34).
- 3671 [BGKS20] E. Ben-Sasson, L. Goldberg, S. Kopparty, and S. Saraf. “DEEP-FRI: Sampling Outside
3672 the Box Improves Soundness.” In: *11th Innovations in Theoretical Computer Science Conference*
3673 *(ITCS 2020)*. Ed. by T. Vidick. Vol. 151. Leibniz International Proceedings in Informatics (LIPIcs).
3674 Pub. by *Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik*, 2020, 5:1–5:32. DOI: [10.4230/LIPIcs.](https://doi.org/10.4230/LIPIcs.ITCS.2020.5)
3675 [ITCS.2020.5](https://doi.org/10.4230/LIPIcs.ITCS.2020.5). Also at ia.cr/2019/336 (Cited on p. 40).

- 3676 [BGM17] S. Bowe, A. Gabizon, and I. Miers. *Scalable Multi-party Computation for zk-SNARK*
3677 *Parameters in the Random Beacon Model*. IACR Cryptology ePrint Archive, ia.cr/2017/1050. 2017.
3678 See also <https://github.com/ebfull/powersoftau> (Cited on p. 60).
- 3679 [BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. “Completeness Theorems for Non-
3680 Cryptographic Fault-Tolerant Distributed Computation.” In: *Proc. 20th Annual ACM Symposium*
3681 *on Theory of Computing*. STOC ’88. Pub. by *Association for Computing Machinery*, 1988, pp. 1–
3682 10. DOI: [10.1145/62212.62213](https://doi.org/10.1145/62212.62213) (Cited on p. 31).
- 3683 [BKS18] E. Ben-Sasson, S. Kopparty, and S. Saraf. “Worst-Case to Average Case Reductions for
3684 the Distance to a Code.” In: *Proc. 33rd Computational Complexity Conference*. CCC ’18. Pub.
3685 by *Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik*, 2018. Also at ECCO [TR18/090](https://eccc.tr18/090) (Cited on
3686 p. 40).
- 3687 [Blu87] M. Blum. “How to prove a theorem so no one else can claim it.” In: *Proc. International*
3688 *Congress of Mathematicians*. 1987, pp. 1444–1451 (Cited on p. 37).
- 3689 [BMRS20] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. *Coda: Decentralized Cryptocurrency*
3690 *at Scale*. IACR Cryptology ePrint Archive, ia.cr/2020/352. 2020 (Cited on p. 42).
- 3691 [BPR19] L. Brandão, R. Peralta, and A. Robinson. *NIST comments on the initial ZKProof doc-*
3692 *umentation*. <https://csrc.nist.gov/Projects/pec/zkproof>. Accessed in 2021. April 2019 (Cited on
3693 p. 89).
- 3694 [BTVW14] A. J. Blumberg, J. Thaler, V. Vu, and M. Walfish. *Verifiable computation using mul-*
3695 *tiiple provers*. IACR Cryptology ePrint Archive, ia.cr/2014/846. 2014 (Cited on p. 33).
- 3696 [CC06] H. Chen and R. Cramer. “Algebraic Geometric Secret Sharing Schemes and Secure Multi-
3697 Party Computations over Small Fields.” In: *Advances in Cryptology — CRYPTO 2006*. Ed. by C.
3698 Dwork. Pub. by *Springer Berlin Heidelberg*, 2006, pp. 521–536. DOI: [10.1007/11818175_31](https://doi.org/10.1007/11818175_31) (Cited
3699 on p. 31).
- 3700 [CCD88] D. Chaum, C. Crépeau, and I. Damgård. “Multiparty Unconditionally Secure Protocols.”
3701 In: *Proc. 20th Annual ACM Symposium on Theory of Computing*. STOC ’88. Pub. by *Association*
3702 *for Computing Machinery*, 1988, pp. 11–19. DOI: [10.1145/62212.62214](https://doi.org/10.1145/62212.62214) (Cited on p. 31).
- 3703 [CCHLRRW19] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Roth-
3704 blum, and D. Wichs. “Fiat-Shamir: From Practice to Theory.” In: *Proc. 51st Annual ACM SIGACT*
3705 *Symposium on Theory of Computing*. STOC 2019. Pub. by *ACM*, 2019, pp. 1082–1090. DOI: [10.](https://doi.org/10.1145/3313276.3316380)
3706 [1145/3313276.3316380](https://doi.org/10.1145/3313276.3316380) (Cited on p. 43).
- 3707 [CD01] R. Cramer and I. Damgård. “Secure Distributed Linear Algebra in a Constant Number of
3708 Rounds.” In: *Advances in Cryptology — CRYPTO 2001*. Ed. by J. Kilian. Pub. by *Springer Berlin*
3709 *Heidelberg*, 2001, pp. 119–136. DOI: [10.1007/3-540-44647-8_7](https://doi.org/10.1007/3-540-44647-8_7) (Cited on p. 31).
- 3710 [CD98] R. Cramer and I. Damgård. “Zero-knowledge proofs for finite field arithmetic, or: Can
3711 zero-knowledge be for free?” In: *Advances in Cryptology — CRYPTO ’98*. Ed. by H. Krawczyk.
3712 Pub. by *Springer Berlin Heidelberg*, 1998, pp. 424–441. DOI: [10.1007/BFb0055745](https://doi.org/10.1007/BFb0055745) (Cited on p. 27).
- 3713 [CDD17] J. Camenisch, M. Drijvers, and M. Dubovitskaya. “Practical UC-Secure Delegatable
3714 Credentials with Attributes and Their Application to Blockchain.” In: *Proc. 2017 ACM SIGSAC*

References

- 3715 *Conference on Computer and Communications Security*. CCS '17. Pub. by ACM, 2017, pp. 683–
3716 699. DOI: [10.1145/3133956.3134025](https://doi.org/10.1145/3133956.3134025) (Cited on p. 67).
- 3717 [CDIKLOV19] M. Chase, Y. Dodis, Y. Ishai, D. Kraschewski, T. Liu, R. Ostrovsky, and V.
3718 Vaikuntanathan. “Reusable Non-Interactive Secure Computation.” In: *Advances in Cryptology —*
3719 *CRYPTO 2019*. Ed. by A. Boldyreva and D. Micciancio. Pub. by Springer International Publishing,
3720 2019, pp. 462–488. DOI: [10.1007/978-3-030-26954-8_15](https://doi.org/10.1007/978-3-030-26954-8_15). Also at ia.cr/2018/940 (Cited on p. 27).
- 3721 [CDIKMRR13] G. Cohen, I. B. Damgård, Y. Ishai, J. Kölker, P. B. Miltersen, R. Raz, and
3722 R. D. Rothblum. “Efficient Multiparty Protocols via Log-Depth Threshold Formulae.” In: *Advances*
3723 *in Cryptology — CRYPTO 2013*. Ed. by R. Canetti and J. A. Garay. Pub. by Springer Berlin
3724 Heidelberg, 2013, pp. 185–202. DOI: [10.1007/978-3-642-40084-1_11](https://doi.org/10.1007/978-3-642-40084-1_11). Also at ia.cr/2013/480 (Cited
3725 on p. 31).
- 3726 [CFIK03] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. “Efficient Multi-party Computation
3727 over Rings.” In: *Advances in Cryptology — EUROCRYPT 2003*. Ed. by E. Biham. Pub. by Springer
3728 Berlin Heidelberg, 2003, pp. 596–613. DOI: [10.1007/3-540-39200-9_37](https://doi.org/10.1007/3-540-39200-9_37). Also at ia.cr/2003/030
3729 (Cited on p. 31).
- 3730 [CGH04] R. Canetti, O. Goldreich, and S. Halevi. “The Random Oracle Methodology, Revisited.”
3731 In: *J. ACM* 51.4 (July 2004), pp. 557–594. DOI: [10.1145/1008731.1008734](https://doi.org/10.1145/1008731.1008734). Also at ia.cr/1998/011
3732 (Cited on p. 37).
- 3733 [CHMMVW20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. “Marlin:
3734 Preprocessing zkSNARKs with Universal and Updatable SRS.” In: *Advances in Cryptology —*
3735 *EUROCRYPT 2020*. Ed. by A. Canteaut and Y. Ishai. Pub. by Springer International Publishing,
3736 2020, pp. 738–768. DOI: [10.1007/978-3-030-45721-1_26](https://doi.org/10.1007/978-3-030-45721-1_26). Also at ia.cr/2019/1047 (Cited on pp. 33,
3737 41).
- 3738 [CKS10] J. Camenisch, M. Kohlweiss, and C. Soriente. “Solving Revocation with Efficient Update
3739 of Anonymous Credentials.” In: *Security and Cryptography for Networks*. Ed. by J. A. Garay and
3740 R. De Prisco. Pub. by Springer Berlin Heidelberg, 2010, pp. 454–471. DOI: [10.1007/978-3-642-
3741 15317-4_28](https://doi.org/10.1007/978-3-642-15317-4_28) (Cited on p. 67).
- 3742 [CL18] R. Canetti and A. Lichtenberg. “Certifying Trapdoor Permutations, Revisited.” In: *Theory*
3743 *of Cryptography*. Ed. by A. Beimel and S. Dziembowski. TCC 2018. Pub. by Springer International
3744 Publishing, 2018, pp. 476–506. DOI: [10.1007/978-3-030-03807-6_18](https://doi.org/10.1007/978-3-030-03807-6_18). Also at ia.cr/2017/631 (Cited
3745 on p. 37).
- 3746 [CMS19] A. Chiesa, P. Manohar, and N. Spooner. “Succinct Arguments in the Quantum Random
3747 Oracle Model.” In: *Theory of Cryptography*. Ed. by D. Hofheinz and A. Rosen. TCC 2019. Pub.
3748 by Springer International Publishing, 2019, pp. 1–29. DOI: [10.1007/978-3-030-36033-7_1](https://doi.org/10.1007/978-3-030-36033-7_1). Also at
3749 ia.cr/2019/834 (Cited on p. 39).
- 3750 [CMT12] G. Cormode, M. Mitzenmacher, and J. Thaler. “Practical Verified Computation with
3751 Streaming Interactive Proofs.” In: *Proc. 3rd Innovations in Theoretical Computer Science Confer-*
3752 *ence*. ITCS '12. Pub. by Association for Computing Machinery, 2012, pp. 90–112. DOI: [10.1145/
3753 2090236.2090245](https://doi.org/10.1145/2090236.2090245). Also at [arXiv:cs/1105.2003](https://arxiv.org/abs/cs/1105.2003) (Cited on p. 33).
- 3754 [COS20] A. Chiesa, D. Ojha, and N. Spooner. “Fractal: Post-quantum and Transparent Recursive
3755 Proofs from Holography.” In: *Advances in Cryptology — EUROCRYPT 2020*. Ed. by A. Canteaut

- 3756 and Y. Ishai. Pub. by *Springer International Publishing*, 2020, pp. 769–793. DOI: [10.1007/978-3-](https://doi.org/10.1007/978-3-030-45721-1_27)
3757 [030-45721-1_27](https://doi.org/10.1007/978-3-030-45721-1_27). Also at ia.cr/2019/1076 (Cited on pp. 33, 40, 42).
- 3758 [CT10] A. Chiesa and E. Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature
3759 Cards.” In: *1st Conference on Innovations in Computer Science — ICS 2010*. Vol. 10. 2010, pp. 310–
3760 331. Also at <https://projects.csail.mit.edu/pcd/> (Cited on pp. 42, 63, 65).
- 3761 [CTV15] A. Chiesa, E. Tromer, and M. Virza. “Cluster Computing in Zero Knowledge.” In: *Ad-*
3762 *vances in Cryptology — EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Pub. by *Springer*
3763 *Berlin Heidelberg*, 2015, pp. 371–403. DOI: [10.1007/978-3-662-46803-6_13](https://doi.org/10.1007/978-3-662-46803-6_13). Also at ia.cr/2015/377
3764 (Cited on p. 42).
- 3765 [Dam10] I. Damgård. *On Σ -protocols*. CPT 2010, v.2. <https://www.cs.au.dk/~ivan/Sigma.pdf>.
3766 2010 (Cited on p. 37).
- 3767 [DFKP16] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. “Cinderella: Turning
3768 Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of Verifiable Com-
3769 putation.” In: *2016 IEEE Symposium on Security and Privacy (SP)*. May 2016, pp. 235–254. DOI:
3770 [10.1109/SP.2016.22](https://doi.org/10.1109/SP.2016.22) (Cited on p. 56).
- 3771 [ECFR] ECFR. *Code of Federal Regulations. eCFR – Title 17, Chapter II, §230.500*. [https://](https://www.ecfr.gov/current/title-17/section-230.501)
3772 www.ecfr.gov/current/title-17/section-230.501. Accessed in 2018 (Cited on p. 77).
- 3773 [EFKP20] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. “SPARKs: Succinct Parallelizable
3774 Arguments of Knowledge.” In: *Advances in Cryptology — EUROCRYPT 2020*. Ed. by A. Canteaut
3775 and Y. Ishai. Pub. by *Springer International Publishing*, 2020, pp. 707–737. DOI: [10.1007/978-3-](https://doi.org/10.1007/978-3-030-45721-1_25)
3776 [030-45721-1_25](https://doi.org/10.1007/978-3-030-45721-1_25). Also at ia.cr/2020/994 (Cited on p. 35).
- 3777 [FLS99] U. Feige, D. Lapidot, and A. Shamir. “Multiple NonInteractive Zero Knowledge Proofs
3778 Under General Assumptions.” In: *SIAM Journal on Computing* 29.1 (1999), pp. 1–28. DOI: [10.1137/](https://doi.org/10.1137/S0097539792230010)
3779 [S0097539792230010](https://doi.org/10.1137/S0097539792230010) (Cited on p. 37).
- 3780 [FS11] L. Fortnow and R. Santhanam. “Infeasibility of instance compression and succinct PCPs
3781 for NP.” In: *Journal of Computer and System Sciences* 77.1 (2011). Celebrating Karp’s Kyoto Prize,
3782 pp. 91–106. DOI: <https://doi.org/10.1016/j.jcss.2010.06.007>. Also at ECCC [TR07/096](https://eccc.weizmann.ac.uk/2010/096) (Cited on
3783 p. 31).
- 3784 [FS87] A. Fiat and A. Shamir. “How To Prove Yourself: Practical Solutions to Identification and
3785 Signature Problems.” In: *Advances in Cryptology — CRYPTO’ 86*. Ed. by A. M. Odlyzko. Vol. 263.
3786 LNCS. Pub. by *Springer Berlin Heidelberg*, 1987, pp. 186–194. DOI: [10.1007/3-540-47721-7_12](https://doi.org/10.1007/3-540-47721-7_12)
3787 (Cited on p. 37).
- 3788 [Gen09] C. Gentry. “A Fully Homomorphic Encryption Scheme.” AAI3382729. PhD thesis. 2009.
3789 DOI: [10.5555/1834954](https://doi.org/10.5555/1834954) (Cited on p. 38).
- 3790 [GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. “Quadratic Span Programs and
3791 Succinct NIZKs without PCPs.” In: *Advances in Cryptology — EUROCRYPT 2013*. Ed. by T.
3792 Johansson and P. Q. Nguyen. Pub. by *Springer Berlin Heidelberg*, 2013, pp. 626–645. DOI: [10.1007/](https://doi.org/10.1007/978-3-642-38348-9_37)
3793 [978-3-642-38348-9_37](https://doi.org/10.1007/978-3-642-38348-9_37). Also at ia.cr/2012/215 (Cited on pp. 28, 38, 39, 43, 50).
- 3794 [GH97] O. Goldreich and J. Håstad. *On the Complexity of Interactive Proof with Bounded Com-*
3795 *munication*. Tech. rep. April 1997. Also at ECCC [TR96/018](https://eccc.weizmann.ac.uk/2010/018) (Cited on p. 37).

References

- 3796 [GK96] O. Goldreich and A. Kahan. “How to Construct Constant-Round Zero-Knowledge Proof
3797 Systems for NP.” In: *J. Cryptol.* 9.3 (June 1996), pp. 167–189. DOI: [10.1007/BF00208001](https://doi.org/10.1007/BF00208001) (Cited on
3798 p. 36).
- 3799 [GKR08] S. Goldwasser, Y. Kalai, and G. Rothblum. “Delegating Computation: Interactive Proofs
3800 for Muggles.” In: vol. 62. May 2008, pp. 113–122. DOI: [10.1145/1374376.1374396](https://doi.org/10.1145/1374376.1374396). Also at Journal of
3801 the ACM, Vol. 62, Issue 4, No. 27, 2015, DOI:[10.1145/2699436](https://doi.org/10.1145/2699436). Also at ECCO TR07/108. (Cited
3802 on pp. 32–34).
- 3803 [GMR89] S. Goldwasser, S. Micali, and C. Rackoff. “The Knowledge Complexity of Interactive
3804 Proof Systems.” In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: [10.1137/0218012](https://doi.org/10.1137/0218012).
3805 Also later at: *SIAM Journal on Computing*, Vol. 18, Issue 1, ACM, 1989, doi:[10.1137/0218012](https://doi.org/10.1137/0218012)
3806 (Cited on p. 1).
- 3807 [GMW91] O. Goldreich, S. Micali, and A. Wigderson. “Proofs That Yield Nothing but Their
3808 Validity or All Languages in NP Have Zero-knowledge Proof Systems.” In: *J. ACM* 38.3 (July
3809 1991), pp. 690–728. DOI: [10.1145/116825.116852](https://doi.org/10.1145/116825.116852) (Cited on pp. 7, 24).
- 3810 [Gol01] O. Goldreich. *The Foundations of Cryptography — Volume 1, Basic Techniques*. <https://www.wisdom.weizmann.ac.il/~oded/foc-vol1.html>. June 2001 (Cited on p. 36).
- 3812 [Gol13] O. Goldreich. “A Short Tutorial of Zero-Knowledge.” In: *Secure Multi-Party Computation*.
3813 Ed. by M. M. Prabhakaran and A. Sahai. Vol. 10. Cryptology and Information Security Series. 2013,
3814 pp. 28–60. DOI: [10.3233/978-1-61499-169-4-28](https://doi.org/10.3233/978-1-61499-169-4-28) (Cited on p. 44).
- 3815 [Gol18] O. Goldreich. “On Doubly-Efficient Interactive Proof Systems.” In: 13.3 (2018), pp. 158–
3816 246. DOI: [10.1561/04000000084](https://doi.org/10.1561/04000000084) (Cited on p. 33).
- 3817 [GOS06] J. Groth, R. Ostrovsky, and A. Sahai. “Perfect Non-interactive Zero Knowledge for NP”
3818 In: *Advances in Cryptology — EUROCRYPT 2006*. Ed. by S. Vaudenay. LNC. Pub. by *Springer*
3819 *Berlin Heidelberg*, 2006, pp. 339–358. DOI: [10.1007/11761679_21](https://doi.org/10.1007/11761679_21). Also at ia.cr/2005/290 (Cited on
3820 p. 44).
- 3821 [Gro10] J. Groth. “Short Pairing-Based Non-interactive Zero-Knowledge Arguments.” In: *Ad-*
3822 *vances in Cryptology — ASIACRYPT 2010*. Ed. by M. Abe. Pub. by *Springer Berlin Heidelberg*,
3823 2010, pp. 321–340. DOI: [10.1007/978-3-642-17373-8_19](https://doi.org/10.1007/978-3-642-17373-8_19) (Cited on pp. 38, 39).
- 3824 [Gro16] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments.” In: *Advances in*
3825 *Cryptology — EUROCRYPT 2016*. Ed. by M. Fischlin and J.-S. Coron. Pub. by *Springer Berlin*
3826 *Heidelberg*, 2016, pp. 305–326. DOI: [10.1007/978-3-662-49896-5_11](https://doi.org/10.1007/978-3-662-49896-5_11). Also at ia.cr/2016/260 (Cited
3827 on pp. 39, 43, 59, 60).
- 3828 [GVW02] O. Goldreich, S. Vadhan, and A. Wigderson. “On Interactive Proofs with a Laconic
3829 Prover.” In: *Comput. Complex.* 11.1/2 (June 2002), pp. 1–53. DOI: [10.1007/s00037-002-0169-0](https://doi.org/10.1007/s00037-002-0169-0)
3830 (Cited on p. 37).
- 3831 [HGBNL21] D. Hopwood, J. Grigg, S. Bowe, K. Nuttycombe, and Y. T. Lai. *Zcash Improvement*
3832 *Proposal 224: Orchard Shielded Protocol*. <https://zips.z.cash/zip-0224>. 2021 (Cited on p. 42).
- 3833 [Hop18] D. Hopwood. *ZCon0 Circuit Optimisation handout*. <https://github.com/ZcashFounda>
3834 [tion/zcon0-workshop-notes](https://github.com/ZcashFoundation/zcon0-workshop-notes). 2018 (Cited on p. 49).

- 3835 **[IKO07]** Y. Ishai, E. Kushilevitz, and R. Ostrovsky. “Efficient Arguments without Short PCPs.” In:
 3836 *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC’07)*. 2007, pp. 278–
 3837 291. DOI: [10.1109/CCC.2007.10](https://doi.org/10.1109/CCC.2007.10) (Cited on p. 26).
- 3838 **[IKOS09]** Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. “Zero-Knowledge Proofs from
 3839 Secure Multiparty Computation.” In: *SIAM Journal on Computing* 39.3 (2009), pp. 1121–1152. DOI:
 3840 [10.1137/080725398](https://doi.org/10.1137/080725398). See also “ZK from SMPC” at Proc. STOC ’07, DOI:[10.1145/1250790.1250794](https://doi.org/10.1145/1250790.1250794)
 3841 (Cited on pp. 30, 31).
- 3842 **[IMSX15]** Y. Ishai, M. Mahmood, A. Sahai, and D. Xiao. *On Zero-Knowledge PCPs: Limitations,*
 3843 *Simplifications, and Applications*. 2015 (Cited on p. 37).
- 3844 **[Ish20]** Y. Ishai. *Zero-Knowledge Proofs from Information-Theoretic Proof Systems. Parts I and*
 3845 *II*. In: **The Art of Zero Knowledge** (ZKProof Blog) — [https://zkproof.org/2020/08/12/](https://zkproof.org/2020/08/12/information-theoretic-proof-systems)
 3846 [information-theoretic-proof-systems](https://zkproof.org/2020/08/12/information-theoretic-proof-systems). 2020 (Cited on p. 89).
- 3847 **[JSI96]** M. Jakobsson, K. Sako, and R. Impagliazzo. “Designated Verifier Proofs and Their Appli-
 3848 cations.” In: *Advances in Cryptology — EUROCRYPT ’96*. Ed. by U. Maurer. Pub. by *Springer*
 3849 *Berlin Heidelberg*, 1996, pp. 143–154. DOI: [10.1007/3-540-68339-9_13](https://doi.org/10.1007/3-540-68339-9_13) (Cited on p. 66).
- 3850 **[Kil92]** J. Kilian. “A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Ab-
 3851 stract).” In: *Proc. 24th Annual ACM Symposium on Theory of Computing*. STOC ’92. Pub. by
 3852 *Association for Computing Machinery*, 1992, pp. 723–732. DOI: [10.1145/129712.129782](https://doi.org/10.1145/129712.129782) (Cited on
 3853 pp. 25, 37).
- 3854 **[Kil95]** J. Kilian. “Improved Efficient Arguments.” In: *Advances in Cryptology — CRYPTO’ 95*.
 3855 Ed. by D. Coppersmith. Vol. 1070. LNCS. Pub. by *Springer Berlin Heidelberg*, 1995, pp. 311–324.
 3856 DOI: [10.1007/3-540-44750-4_25](https://doi.org/10.1007/3-540-44750-4_25) (Cited on p. 43).
- 3857 **[KKW18]** J. Katz, V. Kolesnikov, and X. Wang. “Improved Non-Interactive Zero Knowledge with
 3858 Applications to Post-Quantum Signatures.” In: *Proc. 2018 ACM SIGSAC Conference on Computer*
 3859 *and Communications Security*. CCS ’18. Pub. by *Association for Computing Machinery*, 2018,
 3860 pp. 525–537. DOI: [10.1145/3243734.3243805](https://doi.org/10.1145/3243734.3243805). Also at ia.cr/2018/475 (Cited on p. 31).
- 3861 **[Kla03]** H. Klauck. “Rectangle size bounds and threshold covers in communication complexity.”
 3862 In: *Proc. 18th IEEE Annual Conference on Computational Complexity*. 2003, pp. 118–134. DOI:
 3863 [10.1109/CCC.2003.1214415](https://doi.org/10.1109/CCC.2003.1214415). Also at [arXiv:cs/0208006](https://arxiv.org/abs/cs/0208006) (Cited on p. 34).
- 3864 **[KMSWP16]** A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. “Hawk: The Blockchain
 3865 Model of Cryptography and Privacy-Preserving Smart Contracts.” In: *2016 IEEE Symposium on*
 3866 *Security and Privacy (SP)*. May 2016, pp. 839–858. DOI: [10.1109/SP.2016.55](https://doi.org/10.1109/SP.2016.55) (Cited on p. 67).
- 3867 **[KPV19]** A. Kattis, K. Panarin, and A. Vlasov. *RedShift: Transparent SNARKs from List Poly-*
 3868 *nomial Commitment IOPs*. IACR Cryptology ePrint Archive, ia.cr/2019/1400. 2019 (Cited on
 3869 p. 41).
- 3870 **[KR08]** Y. T. Kalai and R. Raz. “Interactive PCP.” In: *Proc. 35th International Colloquium on*
 3871 *Automata, Languages and Programming, Part II*. Ed. by L. Aceto, I. Damgård, L. A. Goldberg,
 3872 M. M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz. *ICALP ’08*. Pub. by *Springer-Verlag*, 2008,
 3873 pp. 536–547. DOI: [10.1007/978-3-540-70583-3_44](https://doi.org/10.1007/978-3-540-70583-3_44). Also at ECCC [TR07/031](https://doi.org/10.1007/978-3-540-70583-3_44) (Cited on p. 31).

References

- 3874 [KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polyno-
3875 mials and Their Applications.” In: *Advances in Cryptology — ASIACRYPT 2010*. Ed. by M. Abe.
3876 Pub. by Springer Berlin Heidelberg, 2010, pp. 177–194. DOI: [10.1007/978-3-642-17373-8_11](https://doi.org/10.1007/978-3-642-17373-8_11) (Cited
3877 on p. 41).
- 3878 [Lee20] J. Lee. *Dory: Efficient, Transparent arguments for Generalised Inner Products and Poly-*
3879 *nomial Commitments*. IACR Cryptology ePrint Archive, ia.cr/2020/1274. 2020 (Cited on p. 41).
- 3880 [LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. “Algebraic Methods for Interactive
3881 Proof Systems.” In: *J. ACM* 39.4 (October 1992), pp. 859–868. DOI: [10.1145/146585.146605](https://doi.org/10.1145/146585.146605) (Cited
3882 on pp. 32, 33).
- 3883 [libsnark] libsnark. *libsnark: a C++ library for zkSNARK proofs*. [https://github.com/scipr-](https://github.com/scipr-lab/libsnark)
3884 [lab/libsnark](https://github.com/scipr-lab/libsnark). Accessed in 2022 (Cited on p. 28).
- 3885 [Lin20] Y. Lindell. “Secure Multiparty Computation.” In: *Commun. ACM* 64.1 (December 2020),
3886 pp. 86–96. DOI: [10.1145/3387108](https://doi.org/10.1145/3387108) (Cited on p. 30).
- 3887 [Lip12] H. Lipmaa. “Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-
3888 Knowledge Arguments.” In: *Theory of Cryptography*. Ed. by R. Cramer. Pub. by Springer Berlin
3889 Heidelberg, 2012, pp. 169–189. DOI: [10.1007/978-3-642-28914-9_10](https://doi.org/10.1007/978-3-642-28914-9_10). Also at ia.cr/2011/009 (Cited
3890 on p. 38).
- 3891 [LSTW21] J. Lee, S. T. V. Setty, J. Thaler, and R. S. Wahby. “Linear-time zero-knowledge
3892 SNARKs for R1CS.” In: *Cryptology ePrint Archive, Report ia.cr/2021/030* (2021) (Cited on p. 41).
- 3893 [Mau06] U. Maurer. “Secure multi-party computation made simple.” In: *Discrete Applied Mathe-*
3894 *matics* 154.2 (2006). Coding and Cryptography, pp. 370–381. DOI: [https://doi.org/10.1016/j.dam.](https://doi.org/10.1016/j.dam.2005.03.020)
3895 [2005.03.020](https://doi.org/10.1016/j.dam.2005.03.020) (Cited on p. 31).
- 3896 [Mic00] S. Micali. “Computationally Sound Proofs.” In: *SIAM J. Comput.* 30.4 (October 2000),
3897 pp. 1253–1298. DOI: [10.1137/S0097539795284959](https://doi.org/10.1137/S0097539795284959) (Cited on p. 25).
- 3898 [Mik19] Mikelodder7/Ursa. *Z-mix*. 2019. <https://github.com/mikelodder7/ursa/tree/master/libzmix> (Cited
3899 on p. 67).
- 3900 [Mos10] D. Moshkovitz. *An Alternative Proof of The Schwartz-Zippel Lemma*. TR10-096. 2010
3901 (Cited on p. 27).
- 3902 [Nao03] M. Naor. “On cryptographic assumptions and challenges.” In: *Advances in Cryptology —*
3903 *CRYPTO 2003*. Ed. by D. Boneh. Vol. 2729. Lecture Notes in Computer Science. Pub. by Springer,
3904 Berlin, Heidelberg, 2003. DOI: [10.1007/978-3-540-45146-4_6](https://doi.org/10.1007/978-3-540-45146-4_6) (Cited on p. 38).
- 3905 [NVV18] N. Narula, W. Vasquez, and M. Virza. “zkLedger: Privacy-Preserving Auditing for Dis-
3906 tributed Ledgers.” In: *15th USENIX Symposium on Networked Systems Design and Implementation*
3907 *(NSDI 18)*. Pub. by USENIX Association, 2018, pp. 65–80. Also at ia.cr/2018/241 (Cited on p. 67).
- 3908 [PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova. “Pinocchio: Nearly Practical Veri-
3909 fiable Computation.” In: *IEEE Symposium on Security and Privacy*. May 2013, pp. 238–252. DOI:
3910 [10.1109/SP.2013.47](https://doi.org/10.1109/SP.2013.47). Also at ia.cr/2013/279 (Cited on pp. 28, 50, 51, 59).
- 3911 [RR20] N. Ron-Zewi and R. D. Rothblum. “Local Proofs Approaching the Witness Length [Ex-
3912 tended Abstract].” In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science*

- 3913 (*FOCS*). 2020, pp. 846–857. DOI: [10.1109/FOCS46700.2020.00083](https://doi.org/10.1109/FOCS46700.2020.00083). Also at ia.cr/2019/1062 (Cited
3914 on p. 40).
- 3915 [RRR16] O. Reingold, G. N. Rothblum, and R. D. Rothblum. “Constant-round Interactive Proofs
3916 for Delegating Computation.” In: *Proc. 48th Annual ACM Symposium on Theory of Computing*.
3917 STOC ’16. Pub. by ACM, 2016, pp. 49–62. DOI: [10.1145/2897518.2897652](https://doi.org/10.1145/2897518.2897652). Also at SIAM Journal
3918 on Computing (50-3, 2021), doi:[10.1137/16M1096773](https://doi.org/10.1137/16M1096773). Also at ECCO [TR16/061](https://arxiv.org/abs/1606.06101) (Cited on pp. 32–
3919 34, 42).
- 3920 [RTV04] O. Reingold, L. Trevisan, and S. Vadhan. “Notions of Reducibility between Crypto-
3921 graphic Primitives.” In: *Theory of Cryptography*. Ed. by M. Naor. Pub. by Springer Berlin Heidel-
3922 berg, 2004, pp. 1–20. DOI: [10.1007/978-3-540-24638-1_1](https://doi.org/10.1007/978-3-540-24638-1_1) (Cited on p. 37).
- 3923 [Sch90] C. P. Schnorr. “Efficient Identification and Signatures for Smart Cards.” In: *Advances in*
3924 *Cryptology — EUROCRYPT ’89*. Ed. by J.-J. Quisquater and J. Vandewalle. Vol. 434. LNCS. Pub.
3925 by Springer Berlin Heidelberg, 1990, pp. 688–689. DOI: [10.1007/3-540-46885-4_68](https://doi.org/10.1007/3-540-46885-4_68) (Cited on p. 6).
- 3926 [Set20] S. Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup.”
3927 In: *Advances in Cryptology — CRYPTO 2020*. Ed. by D. Micciancio and T. Ristenpart. Pub. by
3928 Springer International Publishing, 2020, pp. 704–737. DOI: [10.1007/978-3-030-56877-1_25](https://doi.org/10.1007/978-3-030-56877-1_25). Also at
3929 ia.cr/2019/550 (Cited on pp. 33, 41).
- 3930 [SL20] S. T. V. Setty and J. Lee. *Quarks: Quadruple-efficient transparent zkSNARKs*. IACR Cryp-
3931 tology ePrint Archive, ia.cr/2020/1275. 2020 (Cited on p. 41).
- 3932 [Sov18] F. Sovrin. *SovrinTM: A Protocol and Token for Self-Sovereign Identity and Decentralized*
3933 *Trust*. January 2018. [https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-](https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf)
3934 [Token-White-Paper.pdf](https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf) (Cited on p. 67).
- 3935 [Spi96] D. A. Spielman. “Linear-time encodable and decodable error-correcting codes.” In: *IEEE*
3936 *Transactions on Information Theory* 42.6 (1996), pp. 1723–1731. DOI: [10.1109/18.556668](https://doi.org/10.1109/18.556668) (Cited
3937 on p. 35).
- 3938 [Tha13] J. Thaler. “Time-Optimal Interactive Proofs for Circuit Evaluation.” In: *Advances in*
3939 *Cryptology — CRYPTO 2013*. Ed. by R. Canetti and J. A. Garay. Pub. by Springer Berlin Hei-
3940 delberg, 2013, pp. 71–89. DOI: [10.1007/978-3-642-40084-1_5](https://doi.org/10.1007/978-3-642-40084-1_5). Also at ia.cr/2013/351 (Cited on
3941 p. 33).
- 3942 [Tha20] J. Thale. *Blog series: In The Art of Zero Knowledge*. Ed. by ZKProof. [https://zkproof.](https://zkproof.org/2020/03/16/sum-checkprotocol)
3943 [org/2020/03/16/sum-checkprotocol](https://zkproof.org/2020/03/16/sum-checkprotocol). March 2020 (Cited on p. 32).
- 3944 [Val08] P. Valiant. “Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/
3945 Space Efficiency.” In: *5th Theory of Cryptography Conference*. Ed. by R. Canetti. TCC ’08. Pub.
3946 by Springer Berlin Heidelberg, 2008. DOI: [10.1007/978-3-540-78524-8_1](https://doi.org/10.1007/978-3-540-78524-8_1) (Cited on p. 42).
- 3947 [VSBW13] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. “A Hybrid Architecture for Interac-
3948 tive Verifiable Computation.” In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 223–
3949 237. DOI: [10.1109/SP.2013.48](https://doi.org/10.1109/SP.2013.48) (Cited on p. 33).
- 3950 [WJBSTWW17] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T.
3951 Wies. “Full Accounting for Verifiable Outsourcing.” In: *Proc. 2017 ACM SIGSAC Conference on*

References

- 3952 *Computer and Communications Security*. CCS '17. Pub. by *Association for Computing Machinery*,
3953 2017, pp. 2071–2086. DOI: [10.1145/3133956.3133984](https://doi.org/10.1145/3133956.3133984). Also at ia.cr/2017/242 (Cited on p. 33).
- 3954 [WTTW18] R. S. Wahby, I. Tzialla, a. shelat abhi, J. Thaler, and M. Walfish. “Doubly-efficient
3955 zkSNARKs without trusted setup.” In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE.
3956 2018, pp. 926–943. DOI: [10.1109/SP.2018.00060](https://doi.org/10.1109/SP.2018.00060). Also at ia.cr/2017/1132 (Cited on pp. 33, 41, 43).
- 3957 [WZCPS18] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. “DIZK: A Distributed Zero
3958 Knowledge Proof System.” In: *27th USENIX Security Symposium (USENIX Security 18)*. Pub. by
3959 *USENIX Association*, August 2018, pp. 675–692. Also at ia.cr/2018/691 (Cited on p. 34).
- 3960 [XZZPS19] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. “Libra: Succinct Zero-
3961 Knowledge Proofs with Optimal Prover Computation.” In: *Advances in Cryptology — CRYPTO*
3962 *2019 Proceedings, Part III*. Ed. by A. Boldyreva and D. Micciancio. Pub. by *Springer International*
3963 *Publishing*, 2019, pp. 733–764. DOI: [10.1007/978-3-030-26954-8_24](https://doi.org/10.1007/978-3-030-26954-8_24). Also at ia.cr/2019/317 (Cited
3964 on pp. 33, 41, 43).
- 3965 [zca18] zcash-hackworks/babyzoe. *Baby ZoE — first step towards Zerocash over Ethereum*. 2018.
3966 <https://github.com/zcash-hackworks/babyzoe> (Cited on p. 67).
- 3967 [ZGKPP17] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. “vSQL:
3968 Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases.” In: *2017 IEEE Symposium*
3969 *on Security and Privacy (SP)*. May 2017, pp. 863–880. DOI: [10.1109/SP.2017.43](https://doi.org/10.1109/SP.2017.43). Also at [ia.cr/](https://ia.cr/2017/1145)
3970 [2017/1145](https://ia.cr/2017/1145) (Cited on pp. 33, 41).
- 3971 [ZKP-CC20] ZKProof. *Call for contributions to the ZKProof Community Reference — Review cy-*
3972 *cle 2020: from version 0.2 to 0.3*. Ed. by Z. Editors. [https://github.com/zkpstandard/zkreference/](https://github.com/zkpstandard/zkreference/blob/master/Call-2020-for-contribs-to-ZCRef.pdf)
3973 [blob/master/Call-2020-for-contribs-to-ZCRef.pdf](https://github.com/zkpstandard/zkreference/blob/master/Call-2020-for-contribs-to-ZCRef.pdf). August 2020 (Cited on p. 90).
- 3974 [ZKP-FF] ZKProof. *ZKProof file formats*. https://github.com/zkpstandard/file_formats. Ac-
3975 cessed in 2022 (Cited on p. 51).
- 3976 [ZKP.Science] ZKP.Science. *Zero-Knowledge Proofs — What are they, how do they work, and are*
3977 *they fast yet?* <https://zkp.science>. Sourced from <https://github.com/ZKProofs/ZKProofs.github.io>.
3978 Accessed in 2022 (Cited on pp. 48, 49).
- 3979 [ZKP19] ZKProof. *Notes of the 2nd ZKProof Workshop*. Ed. by D. Benarroch, L. T. A. N. Brandão,
3980 and E. Tromer. Pub. by *zkproof.org*, December 2019. (Workshop held in Berkeley, USA, in April
3981 2019) (Cited on p. 89).
- 3982 [ZLWZSXZ21] J. Zhang, T. Liu, W. Wang, Y. Zhang, D. Song, X. Xie, and Y. Zhang. “Doubly
3983 Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time.” In: *Proc.*
3984 *2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Pub. by
3985 *Association for Computing Machinery*, 2021, pp. 159–177. DOI: [10.1145/3460120.3484767](https://doi.org/10.1145/3460120.3484767). Also at
3986 ia.cr/2020/1247 (Cited on p. 33).
- 3987 [ZXZS20] J. Zhang, T. Xie, Y. Zhang, and D. Song. “Transparent Polynomial Delegation and Its
3988 Applications to Zero Knowledge Proof.” In: *2020 IEEE Symposium on Security and Privacy (SP)*.
3989 2020, pp. 859–876. DOI: [10.1109/SP40000.2020.00052](https://doi.org/10.1109/SP40000.2020.00052). Also at ia.cr/2019/1482 (Cited on pp. 33,
3990 41).

Page intentionally blank

3991 Appendix A. Acronyms and glossary

3992 A.1 Acronyms

- | | | | |
|------|--|------|---|
| 3993 | • 3SAT: 3-satisfiability | 4009 | • PKI: public-key infrastructure |
| 3994 | • AND: AND gate (Boolean gate) | 4010 | • QAP: quadratic arithmetic program |
| 3995 | • API: application program interface | 4011 | • R1CS: rank-1 constraint system |
| 3996 | • CRH: collision-resistant hash (function) | 4012 | • RAM: random access memory |
| 3997 | • CRS: common-reference string | 4013 | • RSA: Rivest–Shamir–Adleman |
| 3998 | • DAG: directed acyclic graph | 4014 | • SHA: secure hash algorithm |
| 3999 | • DSL: domain specific languages | 4015 | • SMPC: secure multiparty computation |
| 4000 | • FFT: fast-Fourier transform | 4016 | • SNARG: succinct non-interactive argument |
| 4001 | • ILC: ideal linear commitment | 4017 | • SNARK: SNARG of knowledge |
| 4002 | • IOP: interactive oracle proofs | 4018 | • SRS: structured reference string |
| 4003 | • LIP: linear interactive proofs | 4019 | • UC: universal composability or universally composable |
| 4004 | • MA: Merlin–Arthur | 4020 | |
| 4005 | • NIZK: non-interactive zero-knowledge | 4021 | • URS: uniform random string |
| 4006 | • NP: non-deterministic polynomial | 4022 | • XOR: eXclusive OR (Boolean gate) |
| 4007 | • PCD: proof-carrying data | 4023 | • ZK: zero knowledge |
| 4008 | • PCP: probabilistic checkable proof | 4024 | • ZKP: zero-knowledge proof |

4025 A.2 Glossary

- 4026 • **NIZK:** Non-Interactive Zero-Knowledge. Proof system, where the prover sends a single message
4027 to the verifier, who then decides to accept or reject. Usually set in the common reference string
4028 model, although it is also possible to have designated verifier NIZK proofs.
- 4029 • **SNARK:** Succinct Non-interactive ARGument of Knowledge. A special type of non-interactive
4030 proof system where the proof size is small and verification is fast.
- 4031 • **Instance:** Public input that is known to both prover and verifier. Notation: x . (Some scientific
4032 articles use “instance” and “statement” interchangeably, but we distinguish between the two.)
- 4033 • **Witness:** Private input to the prover. Others may or may not know something about the
4034 witness. Notation: w .
- 4035 • **Application Inputs:** Parts of the witness interpreted as inputs to an application, coming from
4036 an external data source. The complete witness and the instance can be computed by the prover
4037 from application inputs.
- 4038 • **Relation:** Specification of relationship between instances and witness. A relation can be viewed
4039 as a set of permissible pairs (instance, witness). Notation: R .
- 4040 • **Language:** Set of instances that have a witness in R . Notation: L .

- 4041 • **Statement:** Defined by instance and relation. Claims the instance has a witness in the relation,
4042 which is either true or false. Notation: $x \in L$.
- 4043 • **Constraint System:** a language for specifying relations.
- 4044 • **Proof System:** A zero-knowledge proof system is a specification of how a prover and verifier
4045 can interact for the prover to convince the verifier that the statement is true. The proof system
4046 must be complete, sound and zero-knowledge.
 - 4047 – *Complete:* If the statement is true and both prover and verifier follow the protocol; the verifier
4048 will accept.
 - 4049 – *Sound:* If the statement is false, and the verifier follows the protocol; he will not be convinced.
 - 4050 – *Zero-knowledge:* If the statement is true and the prover follows the protocol; the verifier will
4051 not learn any confidential information from the interaction with the prover apart from the fact
4052 the statement is true.
- 4053 • **Backend:** an implementation of the ZK proof system’s low-level cryptographic protocol.
- 4054 • **Frontend:** means to express ZK statements in a convenient language and to prove such state-
4055 ments in zero knowledge by compiling them into a low-level representation and invoking a suitable
4056 ZK backend.
- 4057 • **Instance reduction:** conversion of the instance in a high-level statement to an instance for a
4058 low-level statement (suitable for consumption by the backend), by a frontend.
- 4059 • **Witness reduction:** conversion of the witness to a high-level statement to witness for a low-level
4060 statement (suitable for consumption by the backend), by a frontend.
- 4061 • **R1CS (Rank 1 Constraint Systems):** an NP-complete language for specifying relations, as
4062 system of bilinear constraints (i.e., a rank 1 quadratic constraint system). This is a more intuitive
4063 reformulation of QAP.
- 4064 • **QAP (Quadratic Arithmetic Program):** An NP-complete language for specifying relations
4065 via a quadratic system in polynomials. See R1CS for an equivalent formulation.

4066 Reference strings:

- 4067 • **CRS (Common Reference String):** A string output by the NIZK’s Generator algorithm,
4068 and available to both the prover and verifier. Consists of proving parameters and verification
4069 parameters. May be a URS or an SRS.
- 4070 • **URS (Uniform Random String):** A common reference string created by uniformly sampling
4071 from some space, and in particular involving no secrets in its creation. (Also called Common
4072 Random String in prior literature; we avoid this term due to the acronym clash with Common
4073 Reference String).
- 4074 • **SRS (Structured Reference String):** A common reference string created by sampling from
4075 some complex distribution, often involving a sampling algorithm with internal randomness that
4076 must not be revealed, since it would create a trapdoor that enables creation of convincing proofs
4077 for false statements. The SRS may be non-universal (depend on the specific relation) or universal
4078 (independent of the relation, i.e., serve for proving all of NP).
- 4079 • **PP (Prover Parameters) or Proving Key:** The portion of the Common Reference String
4080 that is used by the prover.
- 4081 • **VP (Verifier Parameters) or Verification Key:** The portion of the Common Reference
4082 String that is used by the verifier.

Appendix B. Version history

The ZKProof Community Reference (ZkpComRef) development follows a sequence of main versions:

- **Version 0 [2018-08-01]: Baseline documents.** The proceedings of the 1st ZKProof Workshop (May 2018), with contributions settled by 2018-08-01 and available at [ZKProof.org](https://zkproof.org), along with the [ZKProof Charter](#), constitute the starting point of the ZKProof Community reference. Each of the three Workshop tracks — security, applications, implementation — led to a corresponding proceedings document, titled “ZKProof Standards (*track name*) Track Proceedings”. The ZKProof charter is also part of the baseline documents.
- **Version 0.1 [2019-04-11]: LaTeX/PDF compilation.** The several proceedings composing version 0 were ported to LaTeX code and compiled into a single PDF document entitled “ZKProof Community Reference” (version 0.1) for presentation and discussion at the 2nd ZKProof workshop. That version included editorial adjustments for easier indexation and consistent style.
- **Version 0.2 [2019-12-31]: Consolidated draft.** The process of consolidating the draft community reference document started at the 2nd ZKProof workshop (April 2019), where an editorial process was introduced and several “breakout sessions” were held for discussion on focused topics, including the “NIST comments on the initial ZKProof documentation”. The discussions yielded suggestions of topics to develop and incorporate in a new version of the document. Several concrete items of “proposed contributions” were then defined as GitHub issues. Subsequently, various submitted contributions provided content improvements, such as: distinguish [ZKPs of knowledge vs. of membership](#); recommend [security parameters for benchmarks](#); clarify some [terminology](#) related to ZKP systems (e.g., statements, CRS, RICS); discuss [interactivity vs. non-interactivity](#), and [transferability vs. deniability](#); clarify the scope of use-cases and applications; update the “gadgets” table; add new [references](#). The new version also includes numerous editorial improvements towards a consolidated document, namely a substantially reformulated frontmatter with several new sections ([abstract](#), [about this community reference](#), [change log](#), [acknowledgments](#), [intellectual property](#), [executive summary](#)), a reorganized structure with a new chapter (still to be completed) on construction [paradigms](#). The changes are tracked in a “diff” version of the document.
- **Version 0.3 [2022-07-17]: Paradigms add-on.** The main update was in the [Paradigms](#) chapter, replacing the old “Taxonomy of Constructions” section by three new sections: “[Background](#)” (§2.1), “[Information-Theoretic \(IT\) Proof Systems](#)” (§2.2) and “[Cryptographic Compilers \(CC\)](#)” (§2.3). New placeholder sections (contributions needed) were also included for “[Specialized ZK Proofs](#)” (§2.4) and “[Proof Composition](#)” (§2.5). Other small changes include an improvement of Table 1.1 in Section 1.2, a new paragraph in the [executive summary](#).

Additional documentation covering the development history of the ZkpComRef can be found in the ZKProof GitHub repository (<https://github.com/zkpstandard/zkreference>) and the ZKProof webpage (<https://docs.zkproof.org/reference>) and forum (<https://community.zkproof.org>).

Page intentionally blank