

Annotated changes

in the ZKProof Community Reference version 0.2

2019-12-31

Compared with the clean version, this version contains:

- in the left margins: line numbers
- in the right margins: indices of **edits** (**Ex**) and references to **contribution items** (**Cy.z**)
- in the end of the document: [tables of contribution items](#) since version 0.1

Check the “diff” version (another document) for better detail on the deleted and added content.

List of Contributions

C1: Implement editorial structural changes	86
C2: Set expectations on intellectual property disclosure	90
C3: Add an executive summary	91
C4: Clarify proofs of knowledge	91
C5: Explain the computational security parameter	93
C6: Clarify the public vs. non-public aspect of “common” in CRS enhancement	93
C7: Discuss transferability and deniability	94
C8: Explain the statistical security parameter	95
C9: Clarify the (implicit) scope of some use-cases	96
C10: Compare circuits vs. R1CS	97
C11: Add introduction to interactive zero-knowledge proofs	98
C12: Improve description of applications and predicates	98
C13: Improve motivation in the application chapter	99
C14: Improve the table of gadgets	100
C15: Include references in Application chapter	100

ZKProof Community Reference

Version 0.2

E1: C1.2

December 31, 2019

This document is an ongoing work.

Feedback and contributions are encouraged.

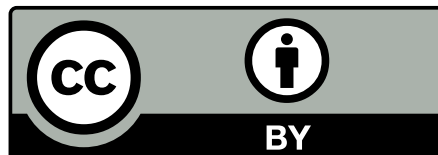
Find the latest version at <https://zkproof.org>.

E2: C1.2

Send your comments to editors@zkproof.org.



ZKPROOF



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

Abstract


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

E3: C1.5

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

19 **About this version.** This is the version 0.2 of the ZKProof Community Reference. It results
20 from the help of many contributors, as described in the [Acknowledgments](#), in the [Version history](#),
21 and in the documentation of previous ZKProof workshops. At a 0.x version, this document should
22 be considered as being in an incomplete state, serving as a basis for further development. Reaching
23 a future stable version requires additional revision and substantial contributions.

E4: C1.3

24 **Citing this version:**  ZKProof. *ZKProof Community Reference. Version 0.2*. Ed. by D. Benarroch,
25 L. T. A. N. Brandão, E. Tromer. Pub. by zkproof.org. Dec. 2019. Updated versions at <https://zkproof.org>

E5: C1.4

About this community reference

E6: C1.6

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

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

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

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

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

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

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

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

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

63 The team of editors coordinates the process, promoting transparency by means of public calls for
 64 contributions and feedback, using editorial discretion towards the improvement of the document
 65 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.

Contents

109

Table of Contents

E13: C1.10

111	Abstract	B
112	About this version	B
113	About this community reference	i
114	ZKProof charter	ii
115	Intellectual property — expectations on disclosure and licensing	ii
116	Contents	iii
117	Executive summary	vii
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	11
139	1.6.4 Zero knowledge	12
140	1.6.5 Advanced security properties	13
141	1.6.6 Transferability vs. deniability	13

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	Construction paradigms	19
149	2.1	Taxonomy of Constructions	19
150	2.1.1	Proof Systems	20
151	2.1.2	Compilers: Cryptographic	21
152	2.1.3	Compilers: Information-theoretic	22
153	2.2	Interactivity	22
154	2.2.1	Advantages of Interactive Proof and Argument Systems	23
155	2.2.2	Disadvantages of Interactive Proof and Argument Systems	24
156	2.2.3	Nuances on transferability vs. interactivity	25
157		(Non)-Transferability/Deniability of Zero-Knowledge Proofs	26
158	2.3	Several construction paradigms	27
159	3	Implementation	29
160	3.1	Overview	29
161	3.1.1	What this document is NOT about:	29
162	3.2	Backends: Cryptographic System Implementations	29
163	3.3	Frontends: Constraint-System Construction	30
164	3.4	APIs and File Formats	31
165	3.4.1	Generic API	31
166	3.4.2	R1CS File Format	33
167	3.5	Benchmarks	35
168	3.5.1	What metrics and components to measure	35
169	3.5.2	How to run the benchmarks	36
170	3.5.3	What benchmarks to run	37
171	3.6	Correctness and Trust	38
172	3.6.1	Considerations	38
173	3.6.2	SRS Generation	41
174	3.6.3	Contingency plans	42
175	3.7	Extended Constraint-System Interoperability	43
176	3.7.1	Statement and witness formats	43
177	3.7.2	Statement semantics, variable representation & mapping	43
178	3.7.3	Witness reduction	44

179	3.7.4	Gadgets interoperability	44
180	3.7.5	Procedural interoperability	44
181	3.7.6	Proof interoperability	45
182	3.7.7	Common reference strings	45
183	3.8	Future goals	46
184	3.8.1	Interoperability	46
185	3.8.2	Frontends and DSLs	46
186	3.8.3	Verification of implementations	46
187	4	Applications	47
188	4.1	Introduction	47
189	4.2	Types of verifiability	48
190	4.3	Previous works	49
191	4.4	Gadgets within predicates	49
192	4.5	Identity framework	53
193	4.5.1	Overview	53
194	4.5.2	Motivation for Identity and Zero Knowledge	53
195	4.5.3	Terminology / Definitions	53
196	4.5.4	The Protocol Description	54
197	4.5.5	A use-case example of credential aggregation	59
198	4.6	Asset Transfer	62
199	4.6.1	Privacy-preserving asset transfers and balance updates	62
200	4.6.2	Zero-Knowledge Proofs in the asset-tracking model	63
201	4.6.3	Zero-Knowledge proofs in the balance model	65
202	4.7	Regulation Compliance	68
203	4.7.1	Overview	68
204	4.7.2	An example in depth: Proof of compliance for aircraft	69
205	4.7.3	Protocol high level	70
206	4.8	Conclusions	71
207		Acknowledgments	73
208		References	75
209	A	Acronyms and glossary	81
210	A.1	Acronyms	81
211	A.2	Glossary	81
212	B	Version history	83

213 **List of Figures**

214 Figure 3.1: Abstract parties and objects in a NIZK 32

215 **List of Tables**

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

217 Table 2.1: Different types of PCPs 20

218 Table 3.1: APIs and interfaces by types of universality and preprocessing 32

219 Table 4.1: List of gadgets 50

220 Table 4.2: Commitment gadget 50

221 Table 4.3: Signature gadget 51

222 Table 4.4: Encryption gadget 51

223 Table 4.5: Distributed-decryption gadget 51

224 Table 4.6: Random-function gadget 51

225 Table 4.7: Set-membership gadget 52

226 Table 4.8: Mix-net gadget 52

227 Table 4.9: Generic-computation gadget 52

228 Table 4.10: Holder identification 56

229 Table 4.11: Issuer identification 57

230 Table 4.12: Credential Issuance 57

231 Table 4.13: Credential Revocation 58

Executive summary

E14: C3.1

233 Zero-knowledge proofs (ZKPs) are an important privacy-enhancing tool from cryptography. They
234 allow proving the veracity of a statement, related to confidential data, without revealing any in-
235 formation beyond the validity of the statement. ZKPs were initially developed by the academic
236 community in the 1980s, and have seen tremendous improvements since then. They are now of
237 practical feasibility in multiple domains of interest to the industry, and to a large community of
238 developers and researchers. ZKPs can have a positive impact in industries, agencies, and for per-
239 sonal use, by allowing privacy-preserving applications where designated private data can be made
240 useful to third parties, despite not being disclosed to them.

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

247 ZKP systems involve at least two parties: a prover and a verifier. The goal of the prover is to
248 convince the verifier that a statement is true, without revealing any additional information. For
249 example, suppose the prover holds a birth certificate digitally signed by an authority. In order
250 to access some service, the prover may have to prove being at least 18 years old, that is, that
251 there exists a birth certificate, tied to the identify of the prover and digitally signed by a trusted
252 certification authority, stating a birthdate consistent with the age claim. A ZKP allows this, without
253 the prover having to reveal the birthdate.

254 This document describes important aspects of the current state of the art in ZKP security, im-
255 plementation, and applications. There are several use-cases and applications where ZKPs can add
256 value. To better assess this it is useful to benchmark implementations under several metrics, evalu-
257 ate tradeoffs between security and efficiency, and develop an interoperability basis. The security of
258 a proof system is paramount for the system users, but efficiency is also essential for user experience.

259 The “Security” chapter introduces the theory and terminology of ZKP systems. A ZKP system can
260 be described with three components: `setup`, `prove`, `verify`. The `setup`, which can be implemented
261 with various techniques, determines the initial state of the prover and the verifier, including private
262 and common elements. The `prove` and `verify` components are the algorithms followed by the prover
263 and verifier, respectively, possibly in an interactive manner. These algorithms are defined so as to
264 ensure three main security requirements: completeness, soundness, and zero-knowledge.

265 Completeness requires that if both `prove` and `verify` are correct, and if the statement is true, then
266 at the end of the interaction the prover is convinced of this fact. Soundness requires that not even
267 a malicious prover can convince the verifier of a false statement. Zero knowledge requires that even
268 a malicious verifier cannot extract any information beyond the truthfulness of the given statement.

269 The “Implementation” chapter focuses on devising a framework for the implementation of ZKPs,
270 which is important for interoperability. One important aspect to consider upfront is the represen-
271 tation of statements. In a ZKP protocol, the statement needs to be converted into a mathematical
272 object. For example, in the case of proving that an age is at least 18, the statement is equivalent to
273 proving that the private birthdate $Y_1-M_1-D_1$ (year-month-day) satisfies a relation with the present

274 date $Y_2 - M_2 - D_2$, namely that their distance is greater than or equal to 18 years. This simple example
275 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 $Y_2 = Y_1 + 18 \wedge$
276 $M_2 = M_1 \wedge D_2 \geq D_1$. An actual conversion suitable for ZKPs, namely for more complex statements, can
277 pose an implementation challenge. There are nonetheless various techniques that enable converting
278 a statement into a mathematical object, such as a circuit. This document gives special attention to
279 representations based on a Rank-1 constraint system (R1CS) and quadratic arithmetic programs
280 (QAP), which are adopted by several ZKP solutions in use today. Also, the document gives special
281 emphasis to implementations of non-interactive proof systems.

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

295 The interplay between security concepts and implementation guidelines must be balanced in the
296 development of secure, practical, and interoperable ZKP applications. Solutions provided by ZKP
297 technology must be ensured by careful security practices and realistic assumptions. This document
298 aims 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.

A zero-knowledge proof 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 zero-knowledge.

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

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

338 1.1.2 Requirements for a zero-knowledge proof system specification

339 A full proof system specification MUST include:

- 340 1. Precise specification of the type of statements the proof system is designed to handle
- 341 2. Construction including algorithms used by the prover and verifier
- 342 3. If applicable, description of setup the prover and verifier use
- 343 4. Precise definitions of security the proof system is intended to provide
- 344 5. A security analysis that proves the zero-knowledge proof system satisfies the security defini-
 345 tions and a full list of any unproven assumptions that underpin security

346 Efficiency claims about a zero-knowledge proof system should include all relevant performance
 347 parameters for the intended usage. Efficiency claims must be reported fairly and accurately, and if
 348 a comparison is made to other zero-knowledge proof systems a best effort must be made to compare
 349 apples to apples.

350 The remainder of the document will outline common approaches to specifying a zero-knowledge
 351 proof system, outline some construction paradigms, and give guidelines for how to present efficiency
 352 claims.

353 1.2 Terminology

354 **Instance:** Input commonly known to both prover (P) and verifier (V), and used to support the E20: C6.1
 355 statement of what needs to be proven. This common input may either be local to the prover–verifier
 356 interaction, or public in the sense of being known by external parties. Notation: x . (Some scientific
 357 articles use “instance” and “statement” interchangeably, but we distinguish between the two.)

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

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

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

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

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

367 between computational security parameter and statistical security parameter. Notation: k (com-
 368 putational) or s (statistical).

369 **Setup:** The inputs given to the prover and to the verifier, apart from the instance x and the wit-
 370 ness w . The setup of each party can be decomposed into a private component (“PrivateSetup $_P$ ” or
 371 “PrivateSetup $_V$ ”, respectively not known to the other party) and a common component “Common-
 372 Setup = CRS” (known by both parties), where CRS denotes a “common reference string” (required
 373 by some zero-knowledge proof systems). Notation: $\text{setup}_P = (\text{PrivateSetup}_P, \text{CRS})$ and $\text{setup}_V =$
 374 $(\text{PrivateSetup}_V, \text{CRS})$.”

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

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

383 **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

391 1.3 Specifying Statements for ZK

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

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

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

407 1.3.1 Circuit representation

408 A circuit is a directed acyclic graph (DAG) comprised of nodes and labels for nodes, which satisfy
 409 the following constraints:

E24: C10.2

- 410 • Nodes with in-degree 0 are referred to as the **input nodes** and are labeled with some constant
 411 (e.g., 0, 1, ...) or with input variable names (e.g., v_1, v_2, \dots)
- 412 • There is a single node with out-degree 0 that is referred to as the **output node**.
- 413 • Internal nodes are referred to as **gate nodes** and describe a computation performed at the
 414 node.

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

418 **Boolean Circuit satisfiability.** The relation R has instances of the form $x = (C, v_1, \dots, v_{n_x})$
 419 and 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
 420 gate nodes that are labeled with Boolean operations, e.g., XOR or AND, v_1, \dots, v_{n_x} must specify truth
 421 values for some of the input nodes, and w_1, \dots, w_{n_w} must specify truth values for the remaining
 422 input variables, such that when evaluating the circuit the output node becomes 1 (true).

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

429 1.3.2 R1CS representation

430 A rank-1 constraint system (R1CS) is a system of equations represented by a list of triplets $(\vec{a}, \vec{b}, \vec{c})$
 431 of vectors of elements of some field. Each triplet defines a “constraint” as an equation of the form
 432 $(A) \cdot (B) - (C) = 0$. Each of the three elements — (A), (B), (C) — in such equation is a linear
 433 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.

E25: C10.2

434 **R1CS satisfiability.** For all triplets $(\vec{a}, \vec{b}, \vec{c})$ of vectors in the R1CS, the solution vector \vec{s} must
 435 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
 436 element of \vec{s} is fixed to the constant 1 (instead of a variable), to enable encoding constants in the
 437 constraints. The remaining elements represent several kinds of variables:

- 438 • **Witness variables:** known only to the prover; represent external inputs to the constraint
 439 system — the witness of the ZK proof system.
- 440 • **Internal variables:** known only to the prover; internal to the constraint system (represent
 441 the inputs and outputs of multiplication gates);
- 442 • **Instance variables:** known by both prover and verifier.

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

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

- 454 • **Boolean circuits operations:**
 - 455 – **NOT operation:** If x is a Boolean variable, then $1 - x$ is the negation of x . Put differently,
 456 if x is 0 or 1, then $1 - x$ is respectively 1 or 0.
 - 457 – **AND operation:** can be implemented as $(A) \times (B)$
 - 458 – **XOR operation** ($c = a \text{ XOR } b$): can be implemented as $(2 \cdot a) \times (b) = (a + b - c)$, or
 459 equivalently as $c = a + b - (a \text{ AND } b) * 2$
- 460 • **Arithmetic circuit operations:**
 - 461 – Multiplication gates are directly represented as equations of the form $a * b = c$.
 - 462 – Linear constraints are used to keep track of inputs and outputs across these gates, and
 463 to represent addition and multiplication-by-constants.

464 1.3.3 Types of relations

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

E26: C10.2

471 **Setup-dependent relations:** Sometimes it is convenient to let the relation R take an additional
 472 input setup_R , i.e., let the relation contain triples (setup_R, x, w) . The input setup_R can be used
 473 to specify persistent information. For example, for arithmetic circuit satisfiability, if the same
 474 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
 475 input setup_R can also be used to capture trusted input the relation does not check, e.g., a trusted
 476 Rivest–Shamir–Adleman (RSA) modulus.

477 1.4 ZKPs of knowledge vs. ZKPs of membership

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

E27: C4.9

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

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

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

499 1.4.1 Example: ZKP of knowledge of a discrete logarithm (discrete-log)

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

E28: C4.10

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

516 1.4.2 Example: ZKP of knowledge of a hash pre-image

517 Consider a cryptographic hash function $H : \{0, 1\}^{512} \rightarrow \{0, 1\}^{256}$, restricted to binary inputs of
 518 length 512. In this definition of H , the set of all 256-bit strings is the *co-domain*, which might be
 519 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
 520 pre-image), known by the prover and unpredictable to the verifier, for some instance $x = H(w)$
 521 that the prover presents to the verifier. Since a cryptographic hash function is one-way, there is
 522 significance in providing a ZKPoK of a pre-image, which proves knowledge of a witness in the re-
 523 lation $R = \{(x, w) : H(w) = x\}$. Such proof also constitutes directly a proof of membership in the
 524 language L , i.e., that the instance x is a member of the image of H . However, interestingly depend-
 525 ing on the known properties of H , this membership predicate might or might not be self-evident
 526 from the instance x .

E29: C4.11

- 527 • If H is known to have as image the set of all bit-strings of length 256 (i.e., if $L = \{0, 1\}^{256}$),
 528 then membership is self-evident. In this case a ZKP of membership is superfluous, since it is
 529 trivial to verify the property of a bit-string having 256 bits.
- 530 • H may instead have the property that an element x uniformly selected from the co-domain
 531 $\{0, 1\}^{256}$ is not in the image of H , with some noticeable probability (e.g., ≈ 0.368 , if H
 532 is modeled as a random function), and with the membership predicate being difficult to
 533 determine. In this setting it can be useful to have the ability to perform a ZKP of membership.

534 1.4.3 Example: ZKP of membership for graph non-isomorphism

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

E30: C4.12

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

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

552 1.5 Syntax

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

557 1.5.1 Prove($state, m$) \rightarrow ($state, p$)

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

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

565 1.5.2 Verify($state, p$) \rightarrow ($state, m$)

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

- 567 • The initial state of Verify must include an instance x .
- 568 • The initial state of Verify may also include additional setup information $setup_V$, e.g., $state =$
 569 $(setup_V, x)$.
- 570 • If receiving a special initialization message $p = \mathbf{start}$, it means the verifier is to initiate the
 571 protocol.
- 572 • If Verify outputs a special symbol $m = \mathbf{accept}$, it means the verifier accepts the proof of the
 573 statement $x \in L$. In this case, Verify must return $m = \mathbf{accept}$ on all future calls.
- 574 • If Verify outputs a special symbol $m = \mathbf{reject}$, it means the verifier rejects the proof of the
 575 statement $x \in L$. In this case, Verify must return $m = \mathbf{reject}$ on all future calls.

576 The setup information $setup_P$ and $setup_V$ can take many forms. A common example found in the
 577 cryptographic literature is that $setup_P = setup_V = k$, where k is a security parameter indicating
 578 the desired security level of the proof system. It is also conceivable that $setup_P$ and $setup_V$ contain
 579 descriptions of particular choices of primitives to instantiate the proof system with, e.g., to use
 580 the SHA-256 hash function or to use a particular elliptic curve. The setup information may also

581 be generated by a probabilistic process. For example: it may be that setup_P and setup_V include
 582 a common reference string; or, in the case of designated-verifier proofs, setup_P and setup_V may
 583 be correlated in a particular way. When we want to specifically refer to this process, we use a
 584 probabilistic setup algorithm **Setup**.

585 1.5.3 Setup(parameters) \rightarrow (setup_R , setup_P , setup_V , auxiliary output)

586 The setup algorithm may take input parameters, which could for instance be computational or
 587 statistical security parameters indicating the desired security level of the proof system, or size
 588 parameters specifying the size of the statements the proof system should work for, or choices of
 589 cryptographic primitives e.g. the SHA-256 hash function or an elliptic curve.

- 590 • The setup algorithm returns an input setup_R for the relation the proof system is for. An
 591 important special case is where the setup_R is just the empty string, i.e., the relation is
 592 independent of any setup.
- 593 • The setup algorithm returns setup_P for the prover and setup_V for the verifier.
- 594 • There may potentially be additional auxiliary outputs.
- 595 • If the inputs are malformed or any error occurs, the Setup algorithm may output an error
 596 symbol.

597 Some examples of possible setups.

- 598 • NIZK proof system for 3SAT in the uniform reference string model based on trapdoor per-
 599 mutations
 - 600 – $\text{setup}_R = n$, where n specifies the maximal number of clauses
 - 601 – $\text{setup}_P = \text{setup}_V =$ uniform random string of length $N = \text{size}(n, k)$ for some function
 602 $\text{size}(n, k)$ of n and security parameter k
- 603 • Groth-Sahai proofs for pairing-product equations
 - 604 – $\text{setup}_R =$ description of bilinear group defining the language
 - 605 – $\text{setup}_P = \text{setup}_V =$ common reference string including description of the bilinear group
 606 in setup_R plus additional group elements
- 607 • SNARK for QAP such as e.g. Pinocchio
 - 608 – $\text{setup}_R =$ QAP specification including finite field F and polynomials
 - 609 – $\text{setup}_P = \text{setup}_V =$ common reference string including a bilinear group defined over the
 610 same finite field and some group elements
 611 The prover and verifier do not use the same group elements in the common reference
 612 string. For efficiency reasons, one may let setup_P be the subset of the group elements the
 613 prover uses, and setup_V another (much smaller) subset of group elements the verifier uses.
- 614 • Cramer-Shoup hash proof systems
 - 615 – $\text{setup}_R =$ specifies finite cyclic group of prime order
 - 616 – $\text{setup}_P =$ the cyclic group and some group elements
 - 617 – $\text{setup}_V =$ the cyclic group and some discrete logarithms

618 It depends on the concrete setting how Setup runs. In some cases, a trusted third party runs an
 619 algorithm to generate the setup. In other cases, Setup may be a multi-party computation offering
 620 resilience against a subset of corrupt and dishonest parties (and the auxiliary output may represent
 621 side-information the adversarial parties learn from the MPC protocol). Yet, another possibility
 622 is to work in the plain model, where the setup does nothing but copy a security parameter, e.g.,
 623 $\text{setup}_P = \text{setup}_V = k$.

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

626 **Common reference string:** If the setup information is public and known to everybody, we say
 627 the proof system is in the common reference string model. The setup may for instance specify
 628 $\text{setup}_R = \text{setup}_P = \text{setup}_V$, which we then refer to as a common reference string CRS.

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

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

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

639 1.6 Definition and Properties

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

642 1.6.1 Completeness

643 Intuitively, a proof system is complete if an honest prover with a valid witness w for a statement
 644 $x \in L$ can convince an honest verifier that the statement is true. A full specification of a proof
 645 system **must** include a precise definition of completeness that captures this intuition. We give an
 646 example of a definition below for a proof system where the prover initiates.

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

- 648 1. Run **Setup**($parameters$) \rightarrow ($\text{setup}_R, \text{setup}_P, \text{setup}_V, aux$)
- 649 2. Let the adversary choose a worst case instance and witness:
 650 **Adversary**($parameters, \text{setup}_R, \text{setup}_P, \text{setup}_V, aux$) \rightarrow (x, w)
- 651 3. Run the interaction between Prove and Verify until the prover returns **error** or the verifier
 652 accepts or rejects. Let $result$ be the outcome, with the convention that $result = \mathbf{error}$ if the
 653 protocol does not terminate. $\langle \mathbf{Prove}(\text{setup}_P, x, w, \mathbf{start}) ; \mathbf{Verify}(\text{setup}_V, x) \rangle \rightarrow result$

654 • **Adversary** wins if $(\text{setup}_R, x, w) \in R$ and result is not **accept**.

655 We define the adversary’s advantage as a function of parameters to be $\text{Advantage}(\text{parameters}) =$
 656 $\text{Pr}[\mathbf{Adversary} \text{ wins}]$

657 A proof system for R running on parameters is complete if nobody ever constructs an efficient
 658 adversary with significant advantage.

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

664 1.6.2 Soundness

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

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

- 669 1. Run **Setup**(parameters) \rightarrow $(\text{setup}_R, \text{setup}_P, \text{setup}_V, \text{aux})$
- 670 2. Let the (stateful) adversary choose an instance
 671 **Adversary**($\text{parameters}, \text{setup}_R, \text{setup}_P, \text{setup}_V, \text{aux}$) $\rightarrow x$
- 672 3. Let the adversary interact with the verifier and result be the verifier’s output (letting $\text{result} =$
 673 **reject** if the protocol does not terminate). $\langle \mathbf{Adversary} ; \mathbf{Verify}(\text{setup}_V, x) \rangle \rightarrow \text{result}$

674 • **Adversary** wins if $(\text{setup}_R, x) \notin L$ and result is **accept**.

675 We define the adversary’s advantage as a function of parameters to be
 676 $\text{Advantage}(\text{parameters}) = \text{Pr}[\mathbf{Adversary} \text{ wins}]$

677 A proof system for R running on parameters is sound if nobody ever constructs an efficient adversary
 678 with significant advantage.

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

684 1.6.3 Proof of knowledge

685 Intuitively, a proof system is a proof of knowledge if it is not just sound, but that the ability to
 686 convince an honest verifier implies that the prover must “know” a witness. To “know” a witness
 687 can be defined as it being possible to extract a witness from a successful prover. If a proof system

688 is claimed to be a proof of knowledge, then the full specification **must** include a precise definition
 689 of knowledge soundness that captures this intuition, but we do not define proofs of knowledge here.

690 **To improve.** A future version of this document should include here a game definition for the
 691 extractor required by the formal notion of proof of knowledge. This security property also arises
 692 naturally in the ideal/real simulation paradigm, in the context of an *ideal ZKP functionality* that,
 693 in the ideal world, receives the witness directly from the prover. E31: C4.13

694 1.6.4 Zero knowledge

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

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

- 704 1. Choose a bit uniformly at random $0,1 \rightarrow b$
 - 705 2. If $b = 0$ run **Setup(parameters)** \rightarrow (setup_R, setup_P, setup_V, aux)
 - 706 3. Else if $b = 1$ run **SimSetup(parameters)** \rightarrow (setup_R, setup_P, setup_V, aux, trapdoor)
 - 707 4. Let the (stateful) adversary choose an instance and witness
 708 **Adversary(parameters, setup_R, setup_P, setup_V, aux)** \rightarrow (x, w)
 - 709 5. If (setup_R, x, w) $\notin R$ return *guess* = 0
 - 710 6. If $b = 0$ let the adversary interact with the prover and output a guess (letting *guess* = 0 if
 711 the protocol does not terminate). \langle **Prove**(setup_P, x, w) ; **Adversary** $\rangle \rightarrow$ *guess*
 - 712 7. Else if $b = 1$ let the adversary interact with a simulated prover and output a guess (letting
 713 *guess* = 0 if the protocol does not terminate)
 714 \langle **SimProve**(setup_P, x, trapdoor) ; **Adversary** $\rangle \rightarrow$ *guess*
- 715 • **Adversary** wins if *guess* = b

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

$$717 \text{Advantage(parameters)} = | \Pr[\mathbf{Adversary} \text{ wins}] - 1/2 |$$

718 A proof system for R running on parameters is zero knowledge if nobody ever constructs an efficient
 719 adversary with significant advantage.

720 It depends on the application what is considered an efficient adversary (computing equipment,
 721 running time, memory consumption, usage lifetime, etc.) and how large an advantage can be toler-
 722 ated. Special strong notions include statistical zero knowledge (aka unconditional zero knowledge)
 723 where any adversary has small advantage, and perfect zero knowledge, where for any adversary the
 724 advantage is exactly 0.

725 multi-theorem zero knowledge. In the zero-knowledge definition, the adversary interacts with the
 726 prover or simulator on a single instance. It is possible to strengthen the zero-knowledge definition
 727 to guard also against an adversary that sees proofs for multiple instances.

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

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

737 1.6.5 Advanced security properties

738 The literature describes many advanced security notions a proof system may have. These include
 739 security under concurrent composition and nonmalleability to guard against man-in-the-middle
 740 attacks, security against reset attacks in settings where the adversary has physical access, simula-
 741 tion soundness and simulation extractability to assist sophisticated security proofs, and universal
 742 composability.

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

752 1.6.6 Transferability vs. deniability

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

759 Despite *deniability* being often a desired property, the dual property of *transferability* can also be
 760 considered a feature, and such a setting is also of interest in this document. *Transferability* means
 761 that the verifier in a legitimate proof execution becomes able to convince an external party that
 762 the corresponding statement is true. In the case of a statement of knowledge, this means being
 763 convinced that some prover did indeed have the claimed knowledge. In some cases this can be done

E32: C7.1

764 by simply sending the transcript (the verifier’s view) of the interaction (messages exchanged and
765 the internal state of the verifier).

766 For a proper security analysis of an application, it is important to characterize whether deniability
767 of transferability (or a nuanced version of them) is intended. This may be an important aspect of
768 composability with other applications.

769 1.6.7 Examples of setup and trust

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

- 772 • No setup or trustless setup. This is when no trust is required, for instance because the setup
773 is just a copy of a security parameter k , or because everybody can verify the setup is correct
774 directly.
- 775 • Uniform random string. All parties have access to a uniform random string $URS = \text{setup}_R =$
776 $\text{setup}_P = \text{setup}_V$. We can distinguish between the lighter trust case where the parties just need
777 to get a uniformly sampled string, which they may for instance get from a trusted common
778 source of randomness e.g. sunspot activity, and the stronger trust case where zero-knowledge
779 relies on the ability to simulate the URS generation together with a simulation trapdoor.
- 780 • Common reference string. The URS model is a special case of the CRS model. But in the CRS
781 model it is also possible that the common setup is sampled with a non-uniform distribution,
782 which may exclude easy access to a trusted common source. A distinction can be made
783 whether the CRS has a verifiable structure, i.e., it is easy to verify it is well-formed, or
784 whether full trust is required.
- 785 • Designated verifier setup. If we have a setup that generates correlated setup_P and setup_V ,
786 where setup_V is intended only for a designated verifier, we also need to place trust in the
787 setup algorithm. This is for instance the case in Cramer-Shoup public-key encryption where
788 a designated verifier NIZK proof is used to provide security under chosen-ciphertext attack.
789 Here the setup is generated as part of the key generation process, and the recipient can be
790 trusted to do this honestly because it is the recipient’s own interest to make the encryption
791 scheme secure.
- 792 • Random oracle model. The common setup describes a cryptographic hash function, e.g.,
793 SHA256. In the random oracle model, the hash function is heuristically assumed to act
794 like a random oracle that returns a random value whenever it is queried on an input not seen
795 before. There are theoretical examples where the random oracle model fails, exploiting the
796 fact that in real life the hash function is a deterministic function, but in practice the heuristic
797 gives good efficiency and currently no weaknesses are known for ‘natural’ proof systems.
- 798 • There are several proposals to reduce the trust in the setup such as using secure multi-party
799 computation to generate a CRS, using a multi-string model where there are many CRSs and
800 security only relies on a majority being honestly generated, and subversion resistant CRS
801 where zero-knowledge holds even against a maliciously generated CRS.

1.7 Assumptions

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

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

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

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

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

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

Plausibility. At all costs, an intractability assumption that has been broken should not be used. We recommend designing flexible and modular proof systems such that they can be easily updated if an underpinning cryptographic assumption is shown to be false.

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

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

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

848 **System assumptions:** Besides cryptographic assumptions, a proof system may rely on assump-
 849 tions about the equipment or environment it works in. As an example, if the proof system relies
 850 on a trusted setup it should be clearly stated what kind of trust is placed in.

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

855 1.8 Efficiency

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

- 858 • **Round complexity:** Number of transmissions between prover and verifier. Usually mea-
 859 sured in the number of moves, where a move is a message from one party to the other. An
 860 important special case is that of 1-move proof systems, aka non-interactive proof systems,
 861 where the verifier receives a proof from the prover and directly decides whether to accept or
 862 not. Non-interactive proofs may be transferable, i.e., they can be copied, forwarded and used
 863 to convince several verifiers.
- 864 • **Communication:** Total size of communication between prover and verifier. Usually mea-
 865 sured in bits.
- 866 • **Prover computation:** Computational effort the prover expends over the duration of the
 867 protocol. Sometimes measured as a count of the dominant cryptographic operations (to avoid
 868 system dependence) and sometimes measured in seconds on a particular system (when making
 869 concrete measurements).
- 870 • Depending on the intended usage, many other metrics may be important: memory consump-
 871 tion, energy consumption, entropy consumption, potential for parallelisation to reduce time,
 872 and offline/online computation trade-offs.
- 873 • **Verifier computation:** Computational effort the verifier expends over the duration of the
 874 protocol.
- 875 • **Setup cost:** Size of setup parameters, e.g. a common reference string, and computational
 876 cost of creating the setup.

877 Readers of a proof system specification may differ in the granularity they need in the efficiency
 878 measurements. Take as an example a proof system consisting of an information theoretic core that
 879 is then compiled with cryptographic primitives to yield the full system. An implementer will likely
 880 want to have a detailed performance analysis of the information theoretic core as well as the cryp-
 881 tographic compilation, since this will guide her choice of trade-offs and optimizations. A consumer

882 on the other hand will likely want to have a high-level performance analysis and an apples-to-apples
 883 comparison to competing proof systems. We therefore recommend to provide both a detailed anal-
 884 ysis that quantifies all the dominant efficiency costs, and a bottom-line analysis that summarizes
 885 performance for reasonable choices of parameters and identifies the optimal performance region.

886 1.8.1 Characterization of security properties

E33: C5.3

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

898 1.8.2 Computational security levels for benchmarking

E34: C5.4

899 The benchmarks for each technique shall include at least one parametrization achieving a con-
 900 jectured computational security level κ approximately equal to, or greater than, 128 bits. Each
 901 technique should also be benchmarked for at least one additional higher computational security
 902 level, such as 192 or 256 bits. (If only one, the latter is preferred.) The benchmarking at more
 903 than one level aids the understanding of how the efficiency varies with the security level. The
 904 interest in a security level as high as 256 bits can be considered a precautionous (and heuristic) safety
 905 margin, compared for example with intended 128 bits. This is intended to handle the possibility
 906 that the conjectured level of security is later found to have been over-estimated. The evaluation
 907 at computational security below 128 bits may be justified for the purpose of clarifying how the
 908 execution complexity or time varies with the security parameter, but should not be construed as a
 909 recommendation for practical security.

910 **An exception allowing lower computational security parameter.** With utmost care, a
 911 computational security level may be justified below 128 bits, including for benchmarking. The
 912 following text describes as exception. In some interactive ZKPs (see Section 2.2), there may be
 913 cryptographic properties that only need to be held during a portion of a protocol execution, which
 914 in turn may be required to take less than a fixed amount of time, say, one minute. For example, a
 915 commitment scheme used to enable temporary hiding during a coin-flipping protocol may only need
 916 to hold until the other party reveals a secret value. In such case the property may be implemented
 917 with less than 128 bits of security, under special care (namely with respect to composition in a
 918 concurrent setting) and if the difference in efficiency is substantial. Such decreased security level
 919 of a component of a protocol may also be useful for example to enable properties of deniability
 920 (non-transferability).

E35: C5.4

921 Depending on the application, other exceptions may be acceptable, upon careful analysis, when

922 the witness whose knowledge is being proven is itself discoverable from the ZK instance with less
 923 computational resources than those corresponding to 128 bits of security.

924 1.8.3 Statistical security levels for benchmarking

E36: C8.1

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

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

938 For interactive techniques where the efficiency upon using 64 bits of statistical security is similar to
 939 that of using a higher parameter similar to the computation security parameter (at least 128 bits),
 940 then the benchmark **should** use at least one higher statistical parameter that enables retaining high
 941 computational security (at least 128 bits) even if the protocol is transformed into a non-interactive
 942 version via a Fiat-Shamir transformation or similar. In the resulting non-interactive protocols, the
 943 prover is the sole generator of the proof, and so a malicious prover can rewind and restart an at-
 944 tempt to generate a forged proof whenever a non-interactively produced challenge is unsuitable to
 945 complete the forgery. Computational security remains if the expected number of needed attempts
 946 is of the order of 2^κ .

Chapter 2. Construction paradigms

E37: C1.11

2.1 Taxonomy of Constructions

E38: C1.12

There are many different types of zero-knowledge proof systems in the literature that offer different tradeoffs between communication cost, computational cost, and underlying cryptographic assumptions. Most of these proofs can be decomposed into an “information-theoretic” zero-knowledge proof system, sometimes referred to as a zero-knowledge *probabilistically checkable proof* (PCP), and a *cryptographic compiler*, or crypto compiler for short, that compiles such a PCP into a zero-knowledge proof. (Here and in the following, we will sometimes omit the term “zero-knowledge” for brevity even though we focus on zero-knowledge proof systems by default.)

Different kinds of PCPs require different crypto compilers. The crypto compilers are needed because PCPs make unrealistic independence assumptions between values contributed by the prover and queries made by the verifier, and also do not take into account the cost of communicating a long proof. The main advantage of this separation is modularity: PCPs can be designed, analyzed and optimized independently of the crypto compilers, and their security properties (soundness and zero-knowledge) do not depend on any cryptographic assumptions. It may be beneficial to apply different crypto compilers to the same PCP, as different crypto compilers may have incomparable efficiency and security features (e.g., trade succinctness for better computational complexity or post-quantum security).

PCPs can be divided into two broad categories: ones in which the verifier makes point queries, namely reads individual symbols from a proof string, and ones where the verifier makes linear queries that request linear combinations of field elements included in the proof string. Crypto compilers for the former types of PCPs typically only use symmetric cryptography (a collision-resistant hash function in their interactive variants and a random oracle in their non-interactive variants) whereas crypto compilers for the latter type of PCPs typically use homomorphic public-key cryptographic primitives (such as SNARK-friendly pairings).

Table 2.1 summarizes different types of PCPs and corresponding crypto compilers. The efficiency and security features of the resulting zero-knowledge proofs depend on both the parameters of the PCP and the features of the crypto compiler.

Table 2.1: Different types of PCPs

975
976
977
978
979
980
981
982
983
984
985
986
987

Proof System	Inter-action	Queries to Proof	Crypto Compilers	Features
Classical proof (no zk)	No	All	GMW, ...,	1,2,3e
			Cramer-Damgård 98, ...	1,3e
Classical PCP	No	Point Queries	Kilian, Micali, IMS	1,2,3b
Linear PCP	No	Inner-product Queries	IKO,[Gro10],GGPR,BCIOP	3a
IOP	Yes	Point Queries	BCS16+ZKStarks	1,2,3b
			BCS16+Ligero	1,2,3d
Linear IOP	Yes	Inner-product Queries	Hyrax	1,3b/3c
			vSQL	3c
			vRAM [ZGKPP18]	3b
ILC	Yes	Matrix-vector Queries	Bootle 16,[BCGJM18]	1,3b
			Bootle 17	1,2,3d

988 **Notation:** We say that a verifier makes “point queries” to the proof Π if the verifier has access
989 to a proof oracle O^Π that takes as input an index i and outputs the i -th symbol $\Pi(i)$ of the proof.
990 We say that a verifier makes “inner-product queries” to the proof $\Pi \in \mathbb{F}^m$ (for some finite field \mathbb{F})
991 if the proof oracle takes as input a vector $q \in \mathbb{F}^m$ and returns the value $\langle \Pi, q \rangle \in \mathbb{F}$. We say that
992 a verifier makes “matrix-vector queries” to the proof $\Pi \in \mathbb{F}^{m \times k}$ if the proof oracle takes as input a
993 vector $q \in \mathbb{F}^k$ and returns the matrix-vector product $(\Pi \cdot q) \in \mathbb{F}^m$.

- 994 1. No trusted setup
- 995 2. Relies only on symmetric-key cryptography (e.g., collision-resistant hash functions and/or
996 random oracles)
- 997 3. Succinct proofs
 - 998 (a) Fully succinct: Proof length independent of statement size. $O(1)$ crypto elements (fully)
 - 999 (b) Polylog succinct: Polylogarithmic number of crypto elements
 - 1000 (c) Depth-succinct: Depends on depth of a verification circuit representing the statement.
 - 1001 (d) Sqrt succinct: Proportional to square root of circuit size
 - 1002 (e) Non succinct: Proof length is larger than circuit size.

1003 2.1.1 Proof Systems

1004 *Note:* For all of the applications we consider, the prover must run in polynomial time, given a
1005 statement-witness pair, and the verifier must run in (possibly randomized) polynomial time.

- 1006 a. Classical Proofs: In a classical NP/MA proof, the prover sends the verifier a proof string π ,
1007 the verifier reads the entire proof π and the entire statement x , and accepts or rejects.
- 1008 b. PCP (Probabilistically Checkable Proofs): In a PCP proof, the prover sends the verifier a
1009 (possibly very long) proof string π , the verifier makes “point queries” to the proof, reads the

- 1010 entire statement x , and accepts or rejects. Relevant complexity measures for a PCP include
 1011 the verifier's query complexity, the proof length, and the alphabet size.
- 1012 c. Linear PCPs: In a linear PCP proof, the prover sends the verifier a (possibly very long)
 1013 proof string π , which lies in some vector space \mathbb{F}^m . The verifier makes some number of linear
 1014 queries to the proof, reads the entire statement x , and accepts or rejects. Relevant complexity
 1015 measures for linear PCPs include the proof length, query complexity, field size, and the
 1016 complexity of the verifier's decision predicate (when expressed as an arithmetic circuit).
- 1017 d. IOP (Interactive Oracle Proofs): An IOP is a generalization of a PCP to the interactive set-
 1018 ting. In each round of communication, the verifier sends a challenge string c_i to the prover and
 1019 the prover responds with a PCP proof π_i that the verifier may query via point queries. After
 1020 several rounds of interactions, the verifier accepts or rejects. Relevant complexity measures
 1021 for IOPs are the round complexity, query complexity, and alphabet size. IOP generalizes
 1022 the notion of Interactive PCP [KR08], and coincides with the notion of Probabilistically
 1023 Checkable Interactive Proof [RRR16].
- 1024 e. Linear IOP: A linear IOP is a generalization of a linear PCP to the interactive setting. (See
 1025 IOP above.) Here the prover sends in each round a proof vector π_i that the verifier may query
 1026 via linear (inner-product) queries.
- 1027 f. ILC (Ideal Linear Commitment): The ILC model is similar to linear IOP, except that the
 1028 prover sends in each round a proof matrix rather than proof vector, and the verifier learns the
 1029 product of the proof matrix and the query vector. This model relaxes the Linear Interactive
 1030 Proofs (LIP) model from [BCIOP13]. (That is, each ILC proof matrix may be the output of
 1031 an arbitrary function of the input and the verifier's messages. In contrast, each LIP proof
 1032 matrix must be a linear function of the verifier's messages.) Important complexity measures
 1033 for ILCs are the round complexity, query complexity, and dimensions of matrices.

E39: C1.17

1034 2.1.2 Compilers: Cryptographic

- 1035 a. Cramer-Damgård [CD98]: Compiles an NP proof into a zero-knowledge proof. The prover
 1036 evaluates the circuit C recognizing the relation on its statement-witness pair (x, w) . The
 1037 prover commits to every wire value in the circuit and sends these commitments to the verifiers.
 1038 The prover then convinces the verifier using sigma protocols that the wire values are all
 1039 consistent with each other. The prover opens the input wires to x and thus convinces the
 1040 verifier that the circuit $C(x, \cdot)$ is satisfied on some witness w . The compiler uses additively
 1041 homomorphic commitments (instantiated using the discrete-log assumption, for example) and
 1042 generating or verifying the proof requires a number of public-key operations that is linear in
 1043 the size of the circuit C .
- 1044 b. Kilian [Kil95] / Micali [Mic00] / IMS [IMS12]: Compiles a PCP with a small number of
 1045 queries into a succinct proof. The prover produces a PCP proof that x is in L . The prover
 1046 commits to the entire PCP proof using a Merkle tree. The verifier asks the prover to open
 1047 a few positions in the proof. The prover opens these positions and uses Merkle proofs to
 1048 convince the verifier that the openings are consistent with the Merkle commitment. The
 1049 verifier accepts iff the PCP verifier accepts. The compiler can be made non-interactive in the
 1050 random oracle model via the Fiat-Shamir heuristic.

- 1051 c. GGPR [GGPR13a] / BCIOP [BCIOP13]: Compiles a linear PCP into a SNARG via a trans-
 1052 formation to LIPs. The public parameters of the SNARG are as long as the linear PCP
 1053 proof and the SNARG proof consists of a constant number of ciphertexts/commitments (if
 1054 the linear PCP has constant query complexity). In the public verification setting, this com-
 1055 piler relies on “SNARG-friendly” bilinear maps and is thus not post-quantum secure. In
 1056 the designated verifier setting, it can be made post-quantum secure via linear-only encryp-
 1057 tion [BISW17]. Generating the proof requires a number of public-key operations that grows
 1058 linearly (or quasi-linearly) in the size of the circuit recognizing the relation.
- 1059 d. BCS16 [BCS16]: A generalization of the Fiat-Shamir compiler that is useful for collapsing
 1060 many-round public-coin proofs (such as IOPs) into NIZKs in the random-oracle model.
- 1061 e. Hyrax [WTSTW18] and vSQL [ZGKPP17]: Give mechanisms for compiling the GKR proto-
 1062 col [GKR15] into NIZKs in the random oracle model. The techniques in these works generalize
 1063 to compile any public-coin linear IOP (without zero knowledge) into a non-interactive zero-
 1064 knowledge proof in the random-oracle model, that additionally relies on algebraic commitment
 1065 schemes. The latter are typically implemented using homomorphic public-key cryptography.
- 1066 f. Bootle16 [BCCGP16]: Compiler for converting an ILC proof into a many-round succinct proof
 1067 under the discrete-log assumption. Generating and verifying the proof requires a number of
 1068 public-key operations that grows linearly with the size of the circuit recognizing the NP
 1069 relation in question.

1070 Note: In addition to the crypto compilers described above, there are information-theoretic compilers
 1071 that convert between different types of information-theoretic objects.

1072 2.1.3 Compilers: Information-theoretic

- 1073 a. MPC-in-the-Head (IKOS [IKOS07], ZKboo [GMO16], Ligero [AHIV17]): Compiles secure
 1074 multi-party computation protocols into either (zero-knowledge) PCPs or IOPs.
- 1075 b. BCIOP [BCIOP13]: Compiles quadratic arithmetic programs (QAPs) or quadratic span pro-
 1076 grams (QSPs) into linear PCPs such that resulting linear PCP has a degree-two decision
 1077 predicate. The BCIOP paper also gives a compiler for converting linear PCP into 1-round
 1078 LIP/ILC and adding zero-knowledge to linear PCP.
- 1079 c. Bootle17 [BCGGHJ17]: Compiles a proof in the ILC model into an IOP. They also give an
 1080 example instantiation of the ILC proof that yields an IOP proof system with square-root
 1081 complexity.

1082 2.2 Interactivity

1083 Several of the proof systems described in the Taxonomy of Constructions given in Section 2.1 are
 1084 interactive, including classical interactive proofs (IPs), IOPs, and linear IOPs. This means that
 1085 the verifier sends multiple challenge messages to the prover, with the prover replying to challenge
 1086 i before receiving challenge $i + 1$; soundness relies on the prover being unable to predict challenge
 1087 $i + 1$ when it responds to challenge i . The other proof systems from the Taxonomy of Constructions

E40: C11.1

1088 are non-interactive, namely classical PCPs and linear PCPs. All of these proof systems can be com-
 1089 bined with cryptographic compilers to yield argument systems that may or may not be interactive,
 1090 depending on the compiler.

1091 2.2.1 Advantages of Interactive Proof and Argument Systems

1092 a. Efficiency and Simplicity. Interactive proof systems can be simpler or more efficient than non-
 1093 interactive ones. As an example, researchers introduced the IOP model [BCS16; RRR16],
 1094 which is interactive, in part because interactivity allowed for circumventing efficiency bottle-
 1095 necks arising in state of the art PCP constructions [BCGT13]. As another example, some
 1096 argument systems derived from IPs [WTSTW18; XZZPS19] have substantially better space
 1097 complexity for the prover (a key scalability bottleneck) than state of the art PCPs [BCGT13]
 1098 or linear PCPs [GGPR13a; Gro16].

1099 Yet, if an interactive protocol is public coin, it can be rendered non-interactive and publicly
 1100 verifiable in most settings via the Fiat-Shamir transformation (see Section 2.1.2), often with
 1101 little loss in efficiency. This means that protocol designers have the freedom to leverage
 1102 interactivity as a “resource” to simplify protocol design, improve efficiency, weaken or remove
 1103 trusted setup, etc., and still have the option of obtaining a non-interactive argument using
 1104 the Fiat-Shamir transformation.

1105 (Applying the Fiat-Shamir heuristic to an interactive protocol to obtain a non-interactive
 1106 argument may increase soundness error, and may transform statistical security to computa-
 1107 tional security — see Section 1.8.3. However, recent works [BCS16; CCHL+19] show that
 1108 when the transformation is applied to specific IP, IOP, and linear IOP protocols of both
 1109 practical and theoretical interest, the blowup in soundness error is only polynomial in the
 1110 number of rounds of interaction.)

1111 b. Setup. Cryptographic compilers for linear PCPs currently require a structured reference string
 1112 (SRS) (see Section 3.6.2). Here, an SRS is a structured string that must be generated by
 1113 a trusted third party during a setup phase, and soundness requires that any trapdoor used
 1114 during this trusted setup must not be revealed. In contrast, some compilers that apply to IPs,
 1115 IOPs (as well as PCPs), and linear IPs yields arguments in which the prover and the verifier
 1116 need only access a uniform random string (URS), which can be obtained from a common
 1117 source of randomness. Such a setup is referred as *transparent* setup in the literature.

1118 c. Cryptographic Primitives. Argument systems derived from IPs, IOPs, or linear IOPs also
 1119 sometimes rely on more desirable cryptographic primitives. For example, IPs themselves
 1120 are information-theoretically secure, relying on no cryptographic assumptions at all. And
 1121 in contrast to arguments derived from linear PCPs, those derived from IOPs rely only on
 1122 symmetric-key cryptographic primitives (see, e.g., [BCS16]). Finally, it has long been known
 1123 how to obtain succinct *interactive* arguments in the plain model based on falsifiable as-
 1124 sumptions like collision-resistant hash families [Kil95], but this is not the case for succinct
 1125 *non-interactive* arguments.

1126 d. Non-transferability. In some applications, it is essential that proofs be deniable or *non-*
 1127 *transferable* (i.e., it must be impossible for a verifier to convince a third party of the validity
 1128 of the statement — see Sections 1.6.6). While these properties are not unique to interactive
 1129 protocols, interaction offers a natural way to make proofs non-transferable (for details, see
 1130 Section 2.2.3).

- 1131 e. Interactivity May Limit Adversaries' Abilities. Interactive protocols can potentially be run
 1132 with fewer bits of security and hence be more efficient. For example, interactive settings
 1133 may allow for the enforcement of a time limit for the protocol to terminate, limiting the
 1134 runtime of attackers. Alternatively, in an interactive setting it may be possible to ensure
 1135 that adversaries only have one attempt to attack a protocol, while this will not be possible
 1136 in many non-interactive settings. See Section 1.8.2 for details.
- 1137 f. Interactivity May Be Inherent to Applications. Many applications are inherently interactive.
 1138 For example, real-world networking protocols involve multiple messages just to initiate a con-
 1139 nection. In addition, zero-knowledge protocols are often combined with other cryptographic
 1140 primitives in applications (e.g., oblivious transfer). If the other primitives are interactive, then
 1141 the final cryptographic protocol will be interactive regardless of whether the zero-knowledge
 1142 protocol is non-interactive. If an application is inherently interactive, it may be reasonable to
 1143 leverage the interaction as a resource if it can render a protocol simpler, more efficient, etc.

1144 2.2.2 Disadvantages of Interactive Proof and Argument Systems

- 1145 1. Interactive protocols must occur online. In an interactive protocol, the proof cannot simply
 1146 be published or posted and checked later at the verifier's convenience, as can be done with
 1147 non-interactive protocols.
- 1148 2. Public Verifiability. Many applications require that proofs be verifiable by any party at
 1149 any time. Public verifiability may be difficult to achieve for interactive protocols. This is
 1150 because soundness of interactive protocols relies on the prover being unable to predict the
 1151 next challenge it will receive in the protocol. Unless there is a publicly trusted source of
 1152 unpredictable randomness (e.g., a randomness beacon) and a way for provers to timestamp
 1153 messages, it is not clear how any party other than the one sending the challenges can be
 1154 convinced that the challenges were properly generated, and the prover replied to challenge i
 1155 before learning challenge $i + 1$. See Section 2.2.3 below for further details.
- 1156 3. Network latency can make interactive protocols slow. If an interactive protocol consists of
 1157 many messages sent over a network, network latency may contribute significantly to the
 1158 total execution time of the protocol.
- 1159 4. Timing or Side Channel Attacks. Because interactive protocols require the prover to send
 1160 multiple messages, there may be more vulnerability to side channel or timing attacks compared
 1161 to non-interactive protocols. Timing attacks will only affect zero-knowledge, not soundness,
 1162 for public-coin protocols, because the verifier's messages are simply random coins, and timing
 1163 attacks should not leak information to the prover in this case. In private coin protocols, both
 1164 zero-knowledge and soundness may be affected by these attacks.
- 1165 5. Concurrent Security. If an interactive protocol is not used in isolation, but is instead used
 1166 in an environment where multiple interactive protocols may be executed concurrently, then
 1167 considerable care should be taken to ensure that the protocol remains secure. See for example
 1168 [Gol13, Section 2.1] and the references therein. Issues of concurrent execution security are
 1169 greatly mitigated for non-interactive protocols [GOS06].
- 1170 6. Proof Length. Currently, the zero-knowledge protocols with the shortest known proofs are
 1171 based on linear PCPs, which are non-interactive. These proofs are just a few group elements
 1172 (see Table 2.1). While (public-coin) zero-knowledge protocols based on IPs or IOPs can

1173 be rendered non-interactive with the Fiat-Shamir heuristic, they currently produce longer
 1174 proofs. The longer proofs may render these protocols unsuitable for some applications (e.g.,
 1175 public blockchain), but they may still be suitable for other applications (even related ones,
 1176 like enterprise blockchain applications).

1177 2.2.3 Nuances on transferability vs. interactivity

1178 The relation between interactivity and transferability/deniability is not without nuances. The
 1179 following paragraphs show several possible combinations. E41: C7.1

1180 **Non-interactive and deniable.** A non-interactive ZKP may be non-transferable. This may be
 1181 based for example on a setup assumption such as a local CRS that is itself deniable. In that case,
 1182 a malicious verifier cannot prove to an external party that the CRS was the one used in a real
 1183 protocol execution, leading the external party to have reasonable suspicion that the verifier may
 1184 have simulated the CRS so as to become able to simulate a protocol execution transcript, without
 1185 actual participation of a legitimate prover. Another example of non-transferability is when a ZKP
 1186 intended to prove (i) an assertion (of membership or knowledge) actually proves its disjunction
 1187 with (ii) the knowledge of the secret key of a designated verifier, for example assuming a public key
 1188 infrastructure (PKI). This suffices to convince the original verifier the initial statement (i) is true,
 1189 since the verifier knows that the prover does not actually know the secret key (ii). In other words,
 1190 a success in the interactive proof stems from the initial assertion (i) being truthful. However, for
 1191 any external party, the transcript of the proof may conceivably have been produced by the original
 1192 designated verifier, who can simply do it with the knowledge of the secret key (ii). In that sense,
 1193 the designated verifier would be unable to convince others that the transcript of a legitimate proof
 1194 was not simulated by the verifier.

1195 **Non-interactive and transferable.** If transferability is intended as a feature, then a non-
 1196 interactive protocol can be achieved for example with a public (undeniable) CRS. For example,
 1197 if a CRS is generated by a trusted randomness beacon, and if soundness follows from the inability
 1198 of the prover to control the CRS, then any external party (even one not involved with the prover
 1199 at the time of proof generation) can at a later time verify that a proof transcript could have only
 1200 been generated by a legitimate prover.

1201 **Interactive and deniable.** A classical example (in a standalone setting, without concurrent exe-
 1202 cutions) for obtaining the deniability property comes from interactive ZKP protocols proven secure
 1203 based on the use of rewinding. Here, deniability follows from the simulatability of transcripts for
 1204 any malicious verifier. For each interactive step, the simulator learns the challenge issued by the
 1205 possibly malicious verifier, and then rewinds to reselect the preceding message of the prover, so as
 1206 to be able to answer the subsequent challenge. Some techniques require the use of commitments
 1207 and/or trapdoors, and may enable this property even for straight-line simulation (i.e., without
 1208 rewinding), provided there is an appropriate trusted setup.

1209 **Interactive and transferable.** In certain settings it is possible, even from an interactive ZKP
 1210 protocol execution, to produce a transcript that constitutes a transferable proof. Usually, trans-
 1211 ferability can be achieved when the (possibly malicious) verifier can convincingly show to external
 1212 parties that the challenges selected during a protocol execution were unpredictable at the time of
 1213 the determination of the preceding messages of the prover. The transferable proof transcript is then
 1214 composed of the messages sent by the prover and additional information from the internal state of

1215 a malicious verifier, including details about the generation of challenges. For example, a challenge
 1216 produced (by the verifier) as a cryptographic hash output (or as a keyed pseudo-random function)
 1217 of the previous messages may later be used to provide assurance that only a legitimate prover would
 1218 have been able to generate a valid subsequent message (response). As another example, if the inter-
 1219 active ZKP protocol is composed with a communication protocol where the prover authenticates all
 1220 sent messages (e.g., signed within a PKI, and timestamped by a trusted service), then the overall
 1221 sequence of those certified messages becomes, in the hands of the verifier, a transferable proof. Fur-
 1222 thermore, from a transferable transcript, the actual transfer can also be performed in an interactive
 1223 way: the verifier (in possession of the transcript) acts as prover in a transferable ZKP of knowledge of
 1224 a transferable transcript, thereby transferring to the external verifier a new transferable transcript.

1225 (Non)-Transferability/Deniability of Zero-Knowledge Proofs

1226 **Off-line non-transferability (deniability) of ZK proofs.** Zero-knowledge proofs are in gen- E42: C7.2
 1227 eral interactive. Interaction is inherent without a setup. Indeed, Goldreich and Oren showed that
 1228 for non-trivial languages zero-knowledge proofs require at least 3 rounds.

1229 The zero-knowledge property in absence of setup guarantees a property called off-line non-transfer-
 1230 ability, also known as deniability — note that a verifier could always compute an equivalent tran-
 1231 script by running the simulator. This property means that the verifier gets no evidence of having
 1232 received an accepting proof from a prover and thus has no advantage in transferring the received
 1233 proof to others.

1234 **On-line non-transferability of ZK proofs.** The situation is more complicated in case of on-
 1235 line non-transferability. Indeed, in this case a malicious verifier plays with a honest prover in
 1236 a zero-knowledge proof system and at the same time the malicious verifier plays with others in
 1237 the attempt of transferring the proof that he his receiving from the prover. Non-transferability
 1238 is therefore a form of security against man-in-the-middle attacks. Security against such attacks
 1239 is typically referred to as non-malleability when the same zero-knowledge proof system is used by
 1240 the adversary to try to transfer the proof to a honest verifier. When instead different protocols
 1241 are involved as part of the activities of the adversary, some stronger notions are required to model
 1242 security under such attacks (e.g., universal composability).

1243 **Transferability of a NIZK proof: publicly verifiable ZK.** The transferability of a zero-
 1244 knowledge proof could become unavoidable when some forms of setups are considered and the zero-
 1245 knowledge proof makes some crucial use of it. Indeed, notice that both in the common reference
 1246 string model and in the programmable random oracle model one can construct non-interactive
 1247 zero-knowledge proofs. Such proofs cannot be simulated by the verifier with the same setup or the
 1248 same instantiation of the random oracle. More specifically, non-interactive zero-knowledge proofs
 1249 are constructed without the contribution of any verifier, therefore they are publicly verifiable proofs
 1250 that can naturally be transferred among verifiers.

1251 **Designated-verifier NIZK proofs.** With more sophisticated setups other options become pos-
 1252 sible. Consider for instance a verifier possessing a public identity implemented through a public key.
 1253 In this case the prover can compute a non-interactive zero-knowledge proof that makes crucially
 1254 use of the public key of the verifier at the point that the verifier using the corresponding secret key

1255 could compute an indistinguishable proof. In this case we have that the proof is a non-interactive
1256 designated-verifier zero-knowledge proof and is non-transferable since the verifier that receives the
1257 proof could have computed an equivalent proof by herself, therefore there is no evidence to share
1258 with others about the fact that the proof comes from a honest prover.

1259 **Transferability of interactive ZK proofs.** The use of identities implemented through public
1260 keys can also have impact in the interactive case. Consider the case where there is no trusted
1261 setup. In this case one can design an interactive zero-knowledge proof system that can have a
1262 transferability flavor by exploiting the public keys of prover and verifier. Indeed, if the prover signs
1263 the transcript, then the proof is transferable by the verifier to whoever believes that the prover is
1264 honest.

1265 2.3 Several construction paradigms

1266 Zero-knowledge proof protocols can be devised within several paradigms, such as:

E43: C1.13

- 1267 • Specialized protocols for specialized proofs of membership or proofs of knowledge
- 1268 • Proofs based on discrete-log and/or pairings
- 1269 • Probabilistic checkable proofs
- 1270 • Quadratic arithmetic programs
- 1271 • GKR
- 1272 • Interactive oracle proofs
- 1273 • MPC in the head
- 1274 • Using garbled circuits

Page intentionally blank

1275

Chapter 3. Implementation

1276

3.1 Overview

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

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.

1291

3.1.1 What this document is NOT about:

1292

1293

1294

1295

- 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

1296

3.2 Backends: Cryptographic System Implementations

1297

1298

1299

1300

1301

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).

1302

1303

1304

1305

The backend typically consists of a concrete implementation of the ZK proof system(s) given as pseudocode in a corresponding publication (see the [Security Track](#) document for extensive discussion of these), along with supporting code for the requisite arithmetic operations, serialization formats, tests, benchmarking etc.

1306

1307

There are numerous such backends, including implementations of many of the schemes discussed in the [Security Track](#). Most have originated as academic research prototypes, and are available

1308 as open-source projects. Since the offerings and features of backends evolve rapidly, we refer the
 1309 reader to the curated taxonomy at <https://zkp.science> for the latest information.

1310 Considerations for the choice of backends include:

- 1311 • ZK proof system(s) implemented by the backend, and their associated security, assumptions
 1312 and asymptotic performance (as discussed in the Security Track document)
- 1313 • Concrete performance (see Benchmarks section)
- 1314 • Programming language and API style (this consideration may be satisfied by adherence to
 1315 prospective ZK proof standards; see the the API and File Formats section)
- 1316 • Platform support
- 1317 • Availability as open source
- 1318 • Active community of maintainers and users
- 1319 • Correctness and robustness of the implementation (as determined, e.g., by auditing and formal
 1320 verification)
- 1321 • Applications (as evidence of usability and scrutiny).

1322 3.3 Frontends: Constraint-System Construction

1323 The frontend of a ZK proof system implementation provides means to express statements in a
 1324 convenient language and to prove such statements in zero knowledge by compiling them into a
 1325 low-level representation and invoking a suitable ZK backend.

1326 A frontend consists of:

- 1327 • The specification of a high-level language for expressing statements.
- 1328 • A compiler that converts relations expressed in the high-level language into the low-level
 1329 relations suitable for some backend(s). For example, this may produce an R1CS constraint
 1330 system.
- 1331 • Instance reduction: conversion of the instance in a high-level statement to a low-level instance
 1332 (e.g., assignment to R1CS instance variables).
- 1333 • Witness reduction: conversion of the witness to a high-level statement to a low-level witness
 1334 (e.g., assignment to witness variables).
- 1335 • Typically, a library of "gadgets" consisting of useful and hand-optimized building blocks for
 1336 statements.

1337 Languages for expressing statements, which have been implemented in frontends to date include:
 1338 code library for general-purpose languages, domain-specific language, suitably-adapted general-
 1339 purpose high-level language, and assembly language for a virtual CPU.

1340 Frontends' compilers, as well as gadget libraries, often implement various optimizations aiming to
 1341 reduce the cost of the constraint systems (e.g., the number of constraints and variables). This in-
 1342 cludes techniques such as making use of "free linear combinations" in R1CS, using nondeterministic

1343 advice given in witness variables (e.g., for integer arithmetic or random-access memory), removing
1344 redundancies, using cryptographic schemes tailored for the given algebraic settings (e.g., Pedersen
1345 hashing on the Jubjub curve or MiMC for hash functions, RSA verification for digital signatures),
1346 and many other techniques. See the [Zcon0 Circuit Optimisation handout](#) for further discussion.

1347 There are many implemented frontends, including some that provide alternative ways to invoke
1348 the same underlying backends. Most have originated as academic research prototypes, and are
1349 available as open-source projects. Since the offerings and features of frontends evolve rapidly, we
1350 refer the reader to the curated taxonomy at <https://zkp.science> for the latest information.

1351 3.4 APIs and File Formats

1352 Our primary goal is to improve interoperability between proving systems and frontend consumers
1353 of proving system implementations. We focused on two approaches for building standard interfaces
1354 for implementations:

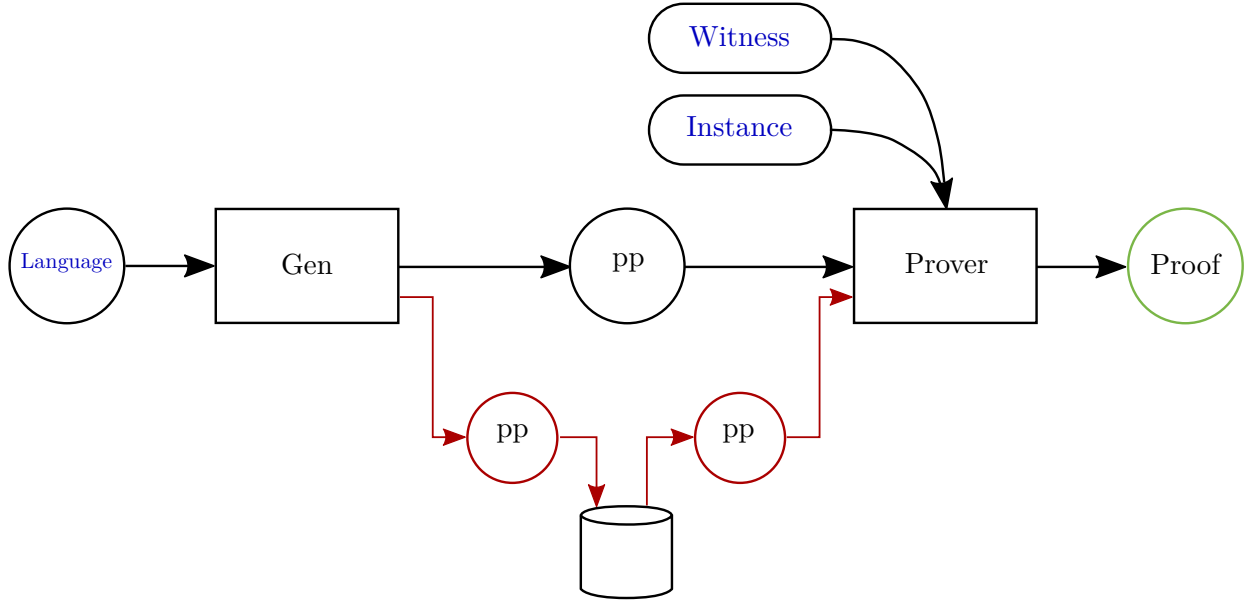
- 1355 1. We aim to develop a common API for proving systems to expose their capabilities to frontends
1356 in a way that is maximally agnostic to the underlying implementation details.
- 1357 2. We aim to develop a file format for encoding a popular form of constraint systems (namely
1358 R1CS), and its assignments, so that proving system implementations and frontends can in-
1359 teract across language and API barriers.

1360 We did not aim to develop standards for interoperability between backends implementing the same
1361 (abstract) scheme, such as serialization formats for proofs (see the Extended Constraint-System
1362 Interoperability section for further discussion).

1363 3.4.1 Generic API

1364 In order to help compare the performance and usability tradeoffs of proving system implemen-
1365 tations, frontend application developers may wish to interact with the underlying proof systems
1366 via a generic interface, so that proving systems can be swapped out and the tradeoffs observed in
1367 practice. This also helps in an academic pursuit of analysis and comparison.

1368 The abstract parties and objects in a NIZK are depicted in Figure 3.1.



1369

1370

Figure 3.1. Abstract parties and objects in a NIZK

1371 We did not complete a generic API design for proving systems, but we did survey numerous tradeoffs
 1372 and design approaches for such an API that may be of future value.

1373 We separate the APIs and interfaces between the universal and non-universal NIZK setting. In
 1374 the universal setting, the NIZK’s CRS generation is independent of the relation (i.e., one CRS
 1375 enables proving any NP statement). In the non-universal settings, the CRS generation depends on
 1376 the relation (represented as a constraint system), and a given CRS enables proving the statements
 1377 corresponding to any instance with respect to the specific relation.

1378

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

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

	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], [GGPR13b], [BCGTV13]	?
Universal (GENERATE needs just a size bound)	vnTinyRAM, vRAM, Bulletproofs (with explicit CRH)	Bulletproofs (with PRG-based CRH generation)

1390 1391 1392	Universal and scalable (GENERATE needs nothing but security parameter)	(impossible)	“Fully scalable” SNARKs based on PCD (recursive composition)
----------------------	---	--------------	---

1393 In any case, we identified several capabilities that proving systems may need to express via a generic
1394 interface:

- 1395 1. The creation of CRS objects in the form of proving and verifying parameters, given the input
1396 language or size bound.
- 1397 2. The serialization of CRS objects into concrete encodings.
- 1398 3. Metadata about the proving system such as the size and characteristic of the field (for arith-
1399 metic constraints).
- 1400 4. Witness objects containing private inputs known only to the prover, and Instance objects
1401 containing public inputs known to the prover and verifier.
- 1402 5. The creation of Proof objects when supplied proving parameters, an Instance, and a Witness.
- 1403 6. The verification of Proof objects given verifying parameters and an Instance.

1404 **Future work:** We would like to see a concrete API design which leverages our tentative model,
1405 with additional work to encode concepts such as recursive composition and the batching of proving
1406 and verification operations.

1407 3.4.2 R1CS File Format

1408 There are many frontends for constructing constraint systems, and many backends which consume
1409 constraint systems (and variable assignments) to create or verify proofs. We focused on creating a
1410 file format that frontends and backends can use to communicate constraint systems and variable
1411 assignments. Goals include simplicity, ease of implementation, compactness and avoiding hard-
1412 coded limits.

1413 Our initial work focuses on R1CS due to its popularity and familiarity. Refer to the [Security](#)
1414 [Track](#) document for more information about constraint systems. The design we arrived at is
1415 tentative and requires further iteration. Implementation and specification work will appear at
1416 https://github.com/zkpstandard/file_formats.

1417 *R1CS (Rank 1 Constraint Systems)* is an NP-complete language for specifying relations as a sys-
1418 tem of bilinear constraints (i.e., a rank 1 quadratic constraint system), as defined in [BCGTV13,
1419 Appendix E in extended version]; this is a more intuitive reformulation of QAP *QAP (Quadratic*
1420 *Arithmetic Program)*, defined in [PHGR13]. R1CS is the native constraint system language of many
1421 ZK proof constructions (see the [Security Track](#) document), including many ZK proof applications
1422 in operational deployment.

1423 Our proposed format makes heavy use of variable-length integers which are prevalent in the (space-
1424 efficient) encoding of an R1CS. We refer to VarInt as a variable-length unsigned integer, and
1425 SignedVarInt as a variable-length signed integer. We typically use VarInt for lengths or version

1426 numbers, and `SignedVarInt` for field element constants. The actual description of a `VarInt` is not
 1427 yet specified.

1428 We'll be working with primitive variable indices of the following form:

```
1429 ConstantVar ← SignedVarInt(0)
1430 InstanceVar(i) ← SignedVarInt(-(i + 1))
1431 WitnessVar(i) ← SignedVarInt(i + 1)
1432 VariableIndex ← ConstantVar / InstanceVar(i) / WitnessVar(i)
```

1433 *ConstantVar* represents an indexed constant in the field, usually assigned to one. *InstanceVar*
 1434 represents an indexed variable of the instance, or the public input, serialized with negative indices.
 1435 *WitnessVar* represents an indexed variable of the witness, or the private/auxiliary input, serialized
 1436 with positive indices. *VariableIndex* represents one of any of these possible variable indices.

1437 We'll also be working with primitive expressions of the following form:

```
1438 Coefficient ← SignedVarInt
1439 Sequence(Entry) ← | length: VarInt | length * Entry |
1440 LinearCombination ← Sequence(| VariableIndex | Coefficient |)
```

- 1441 • Coefficients must be non-zero.
- 1442 • Entries should be sorted by type, then by index:
 - 1443 – | ConstantVar | sorted(InstanceVar) | sorted(WitnessVar) |

```
1444 Constraint ←
1445 | A: LinearCombination | B: LinearCombination | C: LinearCombination |
```

1446 We represent a *Coefficient* (a constant in a linear combination) with a *SignedVarInt*. (TODO: there
 1447 is no constraint on its canonical form.) These should never be zero. We express a *LinearCombi-*
 1448 *nation* as sequences of *VariableIndex* and *Coefficient* pairs. Linear combinations should be sorted
 1449 by type and then by index of the *VariableIndex*; i.e., *ConstantVar* should appear first, *InstanceVar*
 1450 should appear second (ascending) and *WitnessVar* should appear last (ascending).

1451 We express constraints as three *LinearCombination* objects A, B, C, where the encoded constraint
 1452 represents $A * B = C$.

1453 The file format will contain a header with details about the constraint system that are important
 1454 for the backend implementation or for parsing.

```
1455 Header(version, vals) ←
1456 | version: VarInt | vals: Sequence(SignedVarInt) |
```

1457 The *vals* component of the *Header* will contain information such as:

- 1458 • P ← Field characteristic
- 1459 • D ← Degree of extension
- 1460 • N_X ← Number of instance variables
- 1461 • N_W ← Number of witness variables

1462 The representation of elements of extension fields is not currently specified, so D should be 1.

Implementation

1463 The file format contains a magic byte sequence “R1CSstmt”, a header, and a sequence of constraints,
1464 as follows:

```
1465 R1CSFile ←  
1466 | "R1CSstmt" | Header(0, [ P, D, N_X, N_W, ... ]) | Sequence(Constraint) |
```

1467 Further values in the header are undefined in this specification for version 0, and should be ignored.
1468 The file extension “.r1cs” is used for R1CS circuits.

1469 **Further work:** We wish to have a format for expressing the assignments for use by the backend
1470 in generating the proof. We reserve the magic “R1CSasig” and the file extension “.assignments”
1471 for this purpose. We also wish to have a format for expressing symbol tables for debugging. We
1472 reserve the magic “R1CSsymb” and the file extension “.r1cssym” for this purpose.

1473 In the future we also wish to specify other kinds of constraint systems and languages that some
1474 proving systems can more naturally consume.

1475 3.5 Benchmarks

1476 As the variety of zero-knowledge proof systems and the complexity of applications has grown, it
1477 has become more and more difficult for users to understand which proof system is the best for their
1478 application. Part of the reason is that the tradeoff space is high-dimensional. Another reason is
1479 the lack of good, unified benchmarking guidelines. We aim to define benchmarking procedures that
1480 both allow fair and unbiased comparisons to prior work and also aim to give enough freedom such
1481 that scientists are incentivized to explore the whole tradeoff space and set nuanced benchmarks in
1482 new scenarios and thus enable more applications.

1483 The benchmark standardisation is meant to document best practices, not hard requirements. They
1484 are especially recommended for new general-purpose proof systems as well as implementations
1485 of existing schemes. Additionally the long-term goal is to enable independent benchmarking on
1486 standardized hardware.

1487 3.5.1 What metrics and components to measure

1488 We recommend that as the primary metrics the **running time (single-threaded)** and the **com-**
1489 **munication complexity** (proof size, in the case of non-interactive proof systems) of all compo-
1490 nents should be measured and reported for any benchmark. The measured components should
1491 at least include the **prover** and the **verifier**. If the setup is significant then this should also be
1492 measured, further system components like parameter loading and number of rounds (for interactive
1493 proof systems) are suggested.

1494 The following metrics are additionally suggested:

- 1495 • Parallelizability
- 1496 • Batching
- 1497 • Memory consumption (either as a precise measurement or as an upper bound)
- 1498 • Operation counts (e.g., number of field operations, multi-exponentiations, FFTs and their

- 1499 sizes)
- 1500 • Disk usage/Storage requirement
 - 1501 • Crossover point: point where verifying is faster than running the computation
 - 1502 • Largest instance that can be handled on a given system
 - 1503 • Witness generation (this depends on the higher-level compiler and application)
 - 1504 • Tradeoffs between any of the metrics.

1505 3.5.2 How to run the benchmarks

1506 Benchmarks can be both of analytical and computational nature. Depending on the system either
 1507 may be more appropriate or they can supplement each other. An analytical benchmark consists of
 1508 asymptotic analysis as well as concrete formulas for certain metrics (e.g. the proof size). Ideally
 1509 analytical benchmarks are parameterized by a security level or otherwise they should report the
 1510 security level for which the benchmark is done, along with the assumptions that are being used.

1511 Computational benchmarks should be run on a consistent and commercially available machine.
 1512 The use of cloud providers is encouraged, as this allows for cheap reproducibility. The machine
 1513 specification should be reported along with additional restrictions that are put on it (e.g. throt-
 1514 tling, number of threads, memory supplied). Benchmarking machines should generally fall into
 1515 one of the following categories and the machine description should indicate the category. If the
 1516 software implementation makes certain architectural assumptions (such as use of special hardware
 1517 instructions) then this should be clearly indicated.

- 1518 • Battery powered mobile devices
- 1519 • Personal computers such as laptops
- 1520 • Server style machines with many cores and large memories
- 1521 • Server clusters using multiple machines
- 1522 • Custom hardware (should not be used to compare to software implementations)

1523 We recommend that most runs are executed on a single-threaded machine, with parallelizability
 1524 being an optional metric to measure. The benchmarks should be obtained preferably for more than
 1525 one security level, following the recommendations stated in Sections 1.8.2 and 1.8.3.

1526 In order to enable better comparisons we recommend that the metrics of other proof systems/
 1527 implementations are also run on the same machine and reported. The onus is on the library
 1528 developer to provide a simple way to run any instance on which a benchmark is reported. This
 1529 will additionally aid the reproducibility of results. Links to implementations will be gathered at
 1530 zkp.science and library developers are encouraged to ensure that their library is properly referenced.
 1531 Further we encourage scientific publishing venues to require the submission of source code if an
 1532 implementation is reported. Ideally these venues even test the reproducibility and indicate whether
 1533 results could be reproduced.

E44: C5.1

1534 **3.5.3 What benchmarks to run**

1535 We propose a set of benchmarks that is informed by current applications of zero-knowledge proofs,
 1536 as well as by differences in proving systems. This list in no way complete and should be amended
 1537 and updated as new applications emerge and new systems with novel properties are developed.
 1538 Zero-knowledge proof systems can be used in a black-box manner on an existing application, but
 1539 often designing the application with a proof system in mind can yield large efficiency gains. To
 1540 cover both scenarios we suggest a set of benchmarks that include commonly used primitives (e.g.
 1541 SHA-256) and one where only the functionality is specified but not the primitives (e.g. a collision-
 1542 resistant hash function at 128-bit classical security).

E45: C5.2

1543 **Commonly used primitives.** Here we list a set of primitives that both serve as microbench-
 1544 marks and are of separate interest. Library developers are free to choose how their library runs a
 1545 given primitive, but we will aid the process by providing circuit descriptions in commonly used file
 1546 formats (e.g. R1CS).

1547 **Recommended:**

- 1548 1. SHA-256
- 1549 2. AES
- 1550 3. A simple vector or matrix product at different sizes

1551 **Further suggestions:**

- 1552 - Zcash Sapling “spend” relation
- 1553 - RC4 (for RAM memory access)
- 1554 - Scrypt
- 1555 - TinyRAM running for n steps with memory size s
- 1556 - Number theoretic transform (coefficients to points): Small fields; Big fields; Pattern matching.

1557 **Repetition:**

- 1558 • The above relations, parallelized by putting n copies in parallel.

1559 **Functionalities.** The following are examples of cryptographic functionalities that are especially
 1560 interesting to application developers. The realization of the primitive may be secondary, as long
 1561 as it achieves the security properties. It is helpful to provide benchmarks for a constraint-system
 1562 implementation of a realization of these primitives that is tailored for the NIZK backend.

1563 In all of the following, the primitive underlying the ZKP statement should be given at a level of
 1564 128 bits or higher and match the security of the NIZK proof system.

E46: C5.2

- 1565 • Asymmetric cryptography
 - 1566 - Signature verification
 - 1567 - Public key encryption
 - 1568 - Diffie Hellman key exchange over any group with 128 bit security

- 1569 • Symmetric & Hash
 - 1570 - Collision-resistant hash function on a 1024-byte message
 - 1571 - Set membership in a set of size 2^{20} (e.g., using Merkle authentication tree)
 - 1572 - MAC
 - 1573 - AEAD
- 1574 • The scheme’s own verification circuit, with matching parameters, for recursive composition
1575 (Proof-Carrying Data)
- 1576 • Range proofs [Freely chosen commitment scheme]
 - 1577 - Proof that number is in $[0, 2^{64})$
 - 1578 - Proof that number is positive
- 1579 • Proof of permutation (proving that two committed lists contain the same elements)

1580 3.6 Correctness and Trust

1581 In this section we explore the requirements for making the implementation of the proof system
1582 trustworthy. Even if the mathematical scheme fulfills the claimed properties (e.g., it is proven
1583 secure in the requisite sense, its assumptions hold and security parameters are chosen judiciously),
1584 many things can go wrong in the subsequent implementation: code bugs, structured reference
1585 string subversion, compromise during deployment, side channels, tampering attacks, etc. This
1586 section aims to highlight such risks and offer considerations for practitioners.

1587 3.6.1 Considerations

1588 **Design of high-level protocol and statement.** The specification of the high-level protocol
1589 that invokes the ZK proof system (and in particular, the NP statement to be proven in zero
1590 knowledge) may fail to achieve the intended domain-specific security properties.

1591 Methodology for specifying and verifying these protocols is at its infancy, and in practice often relies
1592 on manual review and proof sketches. Possible methods for attaining assurance include reliance on
1593 peer-reviewed academic publications (e.g., Zerocash [BCGG+14] and Cinderella [DFKP16]) reuse of
1594 high-level gadgets as discussed in the [Applications Track](#), careful manual specification and proving
1595 of protocol properties by trained cryptographers, and emerging tools for formal verification.

1596 Whenever nontrivial optimizations are applied to a statement, such as algebraic simplification, or
1597 replacement of an algorithm used in the original intended statement with a more efficient alternative,
1598 those optimizations should be supported by proofs at an appropriate level of formality.

1599 See the [Applications Track](#) document for further discussion.

1600 **Choice of cryptographic primitives.** Traditional cryptographic primitives (hash functions,
1601 PRFs, etc.) in common use are generally not designed for efficiency when implemented in circuits
1602 for ZK proof systems. Within the past few years, alternative “circuit-friendly” primitives have

Implementation

1603 been proposed that may have efficiency advantages in this setting (e.g., LowMC and MiMC). We
1604 recommend a conservative approach to assessing the security of such primitives, and advise that
1605 the criteria for accepting them need to be as stringent as for the more traditional primitives.

1606 **Implementation of statement.** The concrete implementation of the statement to be proven
1607 by the ZK proof system (e.g., as a Boolean circuit or an R1CS) may fail to capture the high-level
1608 specification. This risk increases if the statement is implemented in a low abstraction level, which
1609 is more prone to errors and harder to reason about.

1610 The use of higher-level specifications and domain-specific languages (see the Front Ends section)
1611 can decrease the risk of this error, though errors may still occur in the higher-level specifications
1612 or in the compilation process.

1613 Additionally, risk of errors often arises in the context of optimizations that aim to reduce the size
1614 of the statement (e.g., circuit size or number of R1CS constraints).

1615 Note that correct statement semantics is crucial for security. Two implementations that use the
1616 same high-level protocol, same constraint system and compatible backends may still fail to correctly
1617 interoperate if their instance reductions (from high-level statement to the low-level input required
1618 by the backend) are incompatible – both in completeness (proofs don’t verify) or soundness (causing
1619 false but convincing proofs, implying a security vulnerability).

1620 **Side channels.** Developers should be aware of the different processes in which side channel
1621 attacks can be detrimental and take measure to minimize the side channels. These include:

- 1622 - SRS generation — in some schemes, randomly sampled elements which are discarded can be
1623 used, if exposed, to subvert the soundness of the system.
- 1624 - Assignment generation / proving — the private auxiliary data can be exposed, which allows
1625 the attacker to understand the secret data used for the proof.

1626 **Auditing.** First of all, circuit designers should provide a high-level description of their circuit
1627 and statement alongside the low-level circuit, and explain the connections between them.

1628 The high-level description should facilitate auditing of the security properties of the protocol being
1629 implemented, and whether these match the properties intended by the designers or that are likely
1630 to be expected by users.

1631 If the low-level description is not expressed directly in code, then the correspondence between
1632 the code and the description should be clear enough to be checked in the auditing process, either
1633 manually or with tool support.

1634 A major focus of auditing the correctness and security of a circuit implementation will be in verifying
1635 that the low-level description matches the high-level one. This has several aspects, corresponding
1636 to the security properties of a ZK proof system:

- 1637 • An instance for the low-level circuit must reveal no more information than an instance for the
1638 high-level statement. This is most easily achieved by ensuring that it is a canonical encoding
1639 of the high-level instance.

- 1640 • It must not be possible to find an instance and witness for the low-level circuit that does not
1641 correspond to an instance and witness for the high-level statement.

1642 At all levels of abstraction, it is beneficial to use types to clarify the domains and representations
1643 of the values being manipulated. Typically, a given proving system will not be able to *directly*
1644 represent all of the types of value needed for a given high-level statement; instead, the values will
1645 be encoded, for example as field elements in the case of R1CS-based proof systems. The available
1646 operations on these elements may differ from those on the values they are representing; for instance,
1647 field addition does not correspond to integer addition in the case of overflow.

1648 An adversary who is attempting to prove an instance of the statement that was not intended to be
1649 provable, is not necessarily constrained to using instance and witness variables that correspond to
1650 these intended representations. Therefore, close attention is needed to ensuring that the constraint
1651 system explicitly excludes unintended representations.

1652 There is a wide space of design tradeoffs in how the frontend to a proof system can help to address
1653 this issue. The frontend may provide a rich set of types suitable for directly expressing high-level
1654 statements; it may provide only field elements, leaving representation issues to the frontend user;
1655 it may provide abstraction mechanisms by which users can define new types; etc. Auditability of
1656 statements expressed using the frontend should be a major consideration in this design choice.

1657 If the frontend takes a "gadget" approach to composition of statement elements, then it must be
1658 clear whether each gadget is responsible for constraining the input and/or output variables to their
1659 required types.

1660 **Testing.** Methods to test constraint systems include:

- 1661 - Testing for failure - does the implementation accept an assignment that should not be ac-
1662 cepted?
- 1663 - Fuzzing the circuit inputs.
- 1664 - Finding missing constraints - e.g., missing boolean constraints on variables that represent
1665 bits, or other missing type constraints.
- 1666 - Finding dead constraints, and reporting them (instead of optimising out).
- 1667 - Detection of unintended nondeterminism. For instance, given a partial fixed assignment, solve
1668 for the remainder and check that there is only one solution.

1669 A proof system implementation can support testing by providing access, for test and debugging
1670 purposes, to the reason why a given assignment failed to satisfy the constraints. It should also
1671 support injection of values for instance and witness variables that would not occur in normal use
1672 (e.g. because they do not represent a value of the correct type). These features facilitate "white
1673 box testing", i.e. testing that the circuit implementation rejects an instance and witness *for the*
1674 *intended reason*, rather than incidentally. Without this support, it is difficult to write correct tests
1675 with adequate coverage of failure modes.

1676 **3.6.2 SRS Generation**

1677 A prominent trust issue arises in proving systems which require a parameter setup process (struc-
 1678 tured reference string) that involves secret randomness. These may have to deal with scenarios
 1679 where the process is vulnerable or expensive to perform security. We explore the real world so-
 1680 cial and technical problems that these setups must confront, such as air gaps, public verifiability,
 1681 scalability, handing aborts, and the reputation of participants, and randomness beacons.

1682 ZKP schemes require a URS (*uniform* reference string) or SRS (*structured* reference string) for their
 1683 soundness and/or ZK properties. This necessitates suitable randomness sources and, in the case of
 1684 a common reference string, a securely-executed setup algorithm. Moreover, some of the protocols
 1685 create reference strings that can be reused across applications. We thus seek considerations for
 1686 executing the setup phase of the leading ZKP scheme families, and for sharing of common resources.
 1687 This section summarizes an open discussion made by the participants of the Implementation Track,
 1688 aiming to provide considerations for practitioners to securely generate a CRS.

1689 **SRS subversion and failure modes.** Constructing the SRS in a single machine might fit some
 1690 scenarios. For example, this includes a scenario where the verifier is a single entity — the one
 1691 who generates the SRS. In that scenario, an aspect that should be considered is subversion zero-
 1692 knowledge — a property of proving schemes allowing to maintain zero-knowledge, even if the SRS
 1693 is chosen maliciously by the verifier.

1694 Strategies for subversion zero knowledge include:

- 1695 - Using a multi-party computation to generate the SRS
- 1696 - Adaptation of either [Gro16] [PHGR13]
- 1697 - Updatable SRS - the SRS is generated once in a secure manner, and can then be specialized
 1698 to many different circuits, without the need to re-generate the SRS

1699 There are other subversion considerations which are discussed in the ZKProof [Security Track](#).

1700 **SRS generation using MPC** In order to reduce the need of trust in a single entity generating
 1701 the SRS, it is possible to use a multi-party computation to generate the SRS. This method should
 1702 ideally be secure as long as one participant is honest (per independent computation phase). Some
 1703 considerations to strengthen the security of the MPC include:

- 1704 - Have as many participants as possible
 - 1705 - Diversity of participants; reduce the chance they will collude
 - 1706 - Diversity of implementations (curve, MPC code, compiler, operating system, language)
 - 1707 - Diversity of hardware (CPU architecture, peripherals, RAM)
 - 1708 - One-time-use computers
 - 1709 - GCP / EC2 (leveraging enterprise security)
 - 1710 - If you are concerned about your hardware being compromised, then avoid side channels
 1711 (power, audio/radio, surveillance)
 - 1712 - Hardware removal:

- 1713 - Remove WiFi/Bluetooth chip
- 1714 - Disconnect webcam / microphone / speakers
- 1715 - Remove hard disks if not needed, or disable swap
- 1716 - Air gaps
- 1717 - Deterministic compilation
- 1718 - Append-only logs
- 1719 - Public verifiability of transcripts
- 1720 - Scalability
- 1721 - Handling aborts
- 1722 - Reputation
- 1723 - Information extraction from the hardware is difficult
- 1724 - Flash drives with hardware read-only toggle

1725 Some protocols (e.g., Powers of Tau) also require sampling unpredictable public randomness. Such
 1726 randomness can be harnessed from proof of work blockchains or other sources of entropy such
 1727 as stock markets. Verifiable Delay Functions can further reduce the ability to bias these sources
 1728 [BBBF18]

1729 **SRS reusability** For schemes that require an SRS, it may be possible to design an SRS generation
 1730 process that allows the re-usability of a part of the SRS, thus reducing the attack surface. A good
 1731 example of it is the [Powers of Tau](#) method for the [Groth16](#) construction, where most of the SRS
 1732 can be reused before specializing to a specific constraint system.

1733 **Designated-verifier setting** There are cases where the verifier is a known-in-advance single
 1734 entity. There are schemes that excel in this setting. Moreover, schemes with public verifiability
 1735 can be specialized to this setting as well.

1736 3.6.3 Contingency plans

1737 We would like to explore in future workshops the notion of contingency plans. For example, how
 1738 do we cope:

- 1739 - With our proof system being compromised?
- 1740 - With our specific circuit having a bug?
- 1741 - When our ZKP protocol has been breached (identifying proofs with invalid witness, etc)

1742 Some ideas that were discussed and can be expanded on are:

- 1743 - Scheme-agility and protocol-agility in protocols - when designing the system, allow flexibility
 1744 for the primitives used
- 1745 - Combiners (using multiple proof systems in parallel) - to reduce the reliance on a single proof
 1746 system, use multiple

- 1747 - Discuss ways to identify when ZKP protocol has been breached (identifying proofs with invalid
1748 witness, etc)

1749 **3.7 Extended Constraint-System Interoperability**

1750 The following are stronger forms of interoperability which have been identified as desirable by
1751 practitioners, and are to be addressed by the ongoing standardization effort.

1752 **3.7.1 Statement and witness formats**

1753 In the R1CS File Format section and associated resources, we define a file format for R1CS con-
1754 straint systems. There remains to finalize this specification, including instances and witnesses. This
1755 will enable users to have their choice of frameworks (frontends and backends) and streaming for
1756 storage and communication, and facilitate creation of benchmark test cases that could be executed
1757 by any backend accepting these formats.

1758 Crucially, analogous formats are desired for constraint system languages other than R1CS.

1759 **3.7.2 Statement semantics, variable representation & mapping**

1760 Beyond the above, there's a need for different implementations to coordinate the semantics of the
1761 statement (instance) representation of constraint systems. For example, a high-level protocol may
1762 have an RSA signature as part of the statement, leaving ambiguity on how big integers modulo a
1763 constant are represented as a sequence of variables over a smaller field, and at what indices these
1764 variables are placed in the actual R1CS instance.

1765 Precise specification of statement semantics, in terms of higher-level abstraction, is needed for
1766 interoperability of constraint systems that are invoked by several different implementations of the
1767 instance reduction (from high-level statement to the actual input required by the ZKP prover and
1768 verifier). One may go further and try to reuse the actual implementation of the instance reduction,
1769 taking a high-level and possibly domain-specific representation of values (e.g., big integers) and
1770 converting it into low-level variables. This raises questions of language and platform incompatibility,
1771 as well as proper modularization and packaging.

1772 Note that correct statement semantics is crucial for security. Two implementations that use the
1773 same high-level protocol, same constraint system and compatible backends may still fail to cor-
1774 rectly interoperate if their instance reductions are incompatible – both in completeness (proofs
1775 don't verify) or soundness (causing false but convincing proofs, implying a security vulnerability).
1776 Moreover, semantics are a requisite for verification and helpful for debugging.

1777 Some backends can exploit uniformity or regularity in the constraint system (e.g., repeating patterns
1778 or algebraic structure), and could thus take advantage of formats and semantics that convey the
1779 requisite information.

1780 At the typical complexity level of today's constraint systems, it is often acceptable to handle all of
1781 the above manually, by fresh re-implementation based on informal specifications and inspection of

1782 prior implementation. We expect this to become less tenable and more error prone as application
 1783 complexity grows.

1784 3.7.3 Witness reduction

1785 Similar considerations arise for the witness reduction, converting a high-level witness representation
 1786 (for a given statement) into the assignment to witness variables. For example, a high-level protocol
 1787 may use Merkle trees of particular depth with a particular hash function, and a high-level instance
 1788 may include a Merkle authentication path. The witness reduction would need to convert these
 1789 into witness variables, that contain all of the Merkle authentication path data (encoded by some
 1790 particular convention into field elements and assigned in some particular order) and moreover the
 1791 numerous additional witness variables that occur in the constraints that evaluate the hash function,
 1792 ensure consistency and Booleanity, etc.

1793 The witness reduction is highly dependent on the particular implementation of the constraint
 1794 system. Possible approaches to interoperability are, as above: formal specifications, code reuse and
 1795 manual ad hoc compatibility.

1796 3.7.4 Gadgets interoperability

1797 At a finer grain than monolithic constraint systems and their assignments, there is need for sharing
 1798 subcircuits and gadgets. For example, libsnark offers a rich library of highly optimized R1CS
 1799 gadgets, which developers of several front-end compilers would like to reuse in the context of their
 1800 own constraint-system construction framework.

1801 While porting chunks of constraints across frameworks is relatively straightforward, there are chal-
 1802 lenges in coordinating the semantics of the externally-visible variables of the gadget, analogous
 1803 to but more difficult than those mentioned above for full constraint systems: there is a need to
 1804 coordinate or reuse the semantics of a gadget's externally-visible variables, as well as to coordinate
 1805 or reuse the witness reduction function of imported gadgets in order to converts a witness into an
 1806 assignment to the internal variables.

1807 As for instance semantics, well-defined gadget semantics is crucial for soundness, completeness and
 1808 verification, and is helpful for debugging.

1809 3.7.5 Procedural interoperability

1810 An attractive approach to the aforementioned needs for instance and witness reductions (both at
 1811 the level of whole constraint systems and at the gadget level) is to enable one implementation
 1812 to invoke the instance/witness reductions of another, even across frameworks and programming
 1813 languages.

1814 This requires communication not of mere data, but invocation of procedural code. Suggested ap-
 1815 proaches to this include linking against executable code (e.g., .so files or .dll), using some elegant
 1816 and portable high-level language with its associated portable, or using a low-level portable exe-
 1817 cutable format such as WebAssembly. All of these require suitable calling conventions (e.g., how
 1818 are field elements represented?), usage guidelines and examples.

1819 Beyond interoperability, some low-level building blocks (e.g., finite field and elliptic curve arith-
1820 metic) are needed by many or all implementations, and suitable libraries can be reused. To a large
1821 extent this is already happening, using the standard practices for code reuse using native libraries.
1822 Such reused libraries may offer a convenient common ground for consistent calling conventions as
1823 well.

1824 **3.7.6 Proof interoperability**

1825 Another desired goal is interoperability between provers and verifiers that come from different
1826 implementations, i.e., being able to independently write verifiers that make consistent decisions
1827 and being able to re-implement provers while still producing proofs that convince the old verifier.

1828 This is especially pertinent in applications where proofs are posted publicly, such as in the context
1829 of blockchains (see the Applications Track document), and multiple independent implementations
1830 are desired for both provers and verifiers.

1831 To achieve such interoperability, provers and verifiers must agree on all of the following:

- 1832 • ZK proof system (including fixing all degrees of freedom, such as choice of finite fields and
1833 elliptic curves)
- 1834 • Instance and witness formats (see above subsection)
- 1835 • Prover parameters formats
- 1836 • Verifier parameters formats
- 1837 • Proof formats
- 1838 • A precise specification of the constraint system (e.g., R1CS) and corresponding instance and
1839 witness reductions (see above subsection).

1840 Alternatively: a precise high-level specification along with a precisely-specified, deterministic fron-
1841 tend compilation.

1842 **3.7.7 Common reference strings**

1843 There is also a need for standardization regarding Common Reference String (CRS), i.e., prover
1844 parameters and verifier parameters. First, interoperability is needed for streaming formats (com-
1845 munication and storage), and would allow application developers to easily switch between different
1846 implementations, with different security and performance properties, to suit their need. Moreover,
1847 for Structured Reference Strings (SRS), there are nontrivial semantics that depend on the ZK proof
1848 system and its concrete realization by backends, as well as potential for partial reuse of SRS across
1849 different circuits in some schemes (e.g., the Powers of Tau protocol).

1850 3.8 Future goals

1851 3.8.1 Interoperability

1852 Many additional aspects of interoperability remain to be analyzed and supported by standards,
 1853 to support additional ZK proof system backends as well as additional communication and reuse
 1854 scenarios. Work has begun on multiple fronts both, and a dedicated public [mailing list](#) is established.

1855 **Additional forms of interoperability.** As discussed in the Extended Constraint-System Inter-
 1856 operability section above, even within the R1CS realm, there are numerous additional needs beyond
 1857 plain constraint systems and assignment representations. These affect security, functionality and
 1858 ease of development and reuse.

1859 **Additional relation styles.** The R1CS-style constraint system has been given the most focus
 1860 in the Implementation Track discussions in the first workshop, leading to a file format and an
 1861 API specification suitable for it. It is an important goal to discuss other styles of constraint
 1862 systems, which are used by other ZK proof systems and their corresponding backends. This includes
 1863 arithmetic and Boolean circuits, variants thereof which can exploit regular/repeating elements, as
 1864 well as arithmetic constraint satisfaction problems.

1865 **Recursive composition.** The technique of recursive composition of proofs, and its abstraction as
 1866 Proof-Carrying Data (PCD) [CT10; BCTV14], can improve the performance and functionality of
 1867 ZK proof systems in applications that deal with multi-stage computation or large amounts of data.
 1868 This introduces additional objects and corresponding interoperability considerations. For example,
 1869 PCD compliance predicates are constraint systems with additional conventions that determine their
 1870 semantics, and for interoperability these conventions require precise specification.

1871 **Benchmarks.** We strive to create concrete reference benchmarks and reference platforms, to
 1872 enable cross-paper milliseconds comparisons and competitions.

1873 We seek to create an open competition with well-specified evaluation criteria, to evaluate different
 1874 proof schemes in various well-defined scenarios.

1875 3.8.2 Frontends and DSLs

1876 We would like to expand the discussion on the areas of domain-specific languages, specifically in
 1877 aspects of interoperability, correctness and efficiency (even enabling source-to-source optimisation).

1878 The goal of Gadget Interoperability, in the Extended Constraint-System Interoperability section,
 1879 is also pertinent to frontends.

1880 3.8.3 Verification of implementations

1881 We would to discuss the following subjects in future workshops, to assist in guiding towards best
 1882 practices: formal verification, auditing, consistency tests, etc.

1883

Chapter 4. Applications

1884

4.1 Introduction

1885

1886

1887

1888

1889

1890

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*:

E48: C12.2

1891

1892

- **Predicate:** The relation or condition that the statement and witness must satisfy. Can be represented as a circuit.

1893

1894

1895

- **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).

E49: C1.14

1896

1897

1898

1899

1900

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.

E50: C1.14

1901

1902

1903

1904

1905

1906

1907

1908

1909

1910

1911

1912

1913

1914

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. [BCTV17] and [BCCT13].

E51: C13.1

E52: C15.1

E53: C12.3

1915

1916

1917

1918

1919

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*.

1920 4.2 Types of verifiability

1921 **Verifiability type.** When designing ZK based applications, one needs to keep in mind which of
 1922 the following three models (that define the functionality of the ZKP) is needed:

E54: C1.14

1923 1. **Public.** Publicly verifiable as a requirement: a scheme / use-case where there is a system
 1924 requirement that the proofs are transferable.

E55: C7.3

1925 2. **Designated.** Designated verifier as a security feature: only the intended receiver of the proof
 1926 can verify it, making the proof non-transferable. This property can apply to both interactive
 1927 and non-interactive ZKPs.

1928 3. **Optional.** There is no need to be able to transfer but also no non-transferability requirement.
 1929 This property is applicable both in the interactive and in the non-interactive model.

1930 Section 2.2.3 discusses transferability vs. deniability, which is strongly related to aspects of public
 1931 verifiability vs. designated verifiability, both in the interactive and in the non-interactive settings.
 1932 As a use-case example, consider some application related to blockchain currency, where aspects of
 1933 user-privacy and regulatory-control are relevant.

E56: C9.1

1934 Publicly-verifiable ZKPs can be appropriate when the validity of a transaction should be public
 1935 (e.g., so that everyone knows that some asset changed owner), while some supporting data needs to
 1936 remain private (e.g., the secret key of a blockchain address, controlling the ownership of the asset).
 1937 However, sometimes even the statement being proven should remain private beyond the scope of
 1938 the verifier, and therefore a non-transferable proof should be used. This may apply for example
 1939 to a proof of having enough funds available for a purchase, or also of knowing the secret key of a
 1940 certain blockchain address. Alice wants to prevent Bob from using the received proof to convince
 1941 Charley of the claims made by Alice. For that purpose, Alice can perform a deniability interactive
 1942 proof with Bob. Alternatively, Alice can send to Bob a (non-interactive) proof transcript built for
 1943 Bob as a *designated verifier*. Depending on the use case, both public-verifiability and designated-
 1944 verifiability may make sense as an application goal, and it is important to distinguish between
 1945 both.

1946 The “designation of verifiers” allows resolving possible conflicts between authenticity and privacy
 1947 [JSI96]. For example, a voting center wants only Bob to be convinced that the vote he cast was
 1948 counted; the voting center designates Bob to be the one convinced by the validity of the proof, in
 1949 order to prevent a malicious coercer to force him to prove how he voted. Since the designated-verifier
 1950 proofs are non-transferable, Bob cannot transfer the proof even if he wants to.

E57: C9.2

1951 Suppose Alice wants to convince *only* Bob that a statement θ is true. For that purpose, Alice can
 1952 prove the disjunction “Either θ is true or I know the secret key of Bob”. Given that Bob knows his
 1953 own secret key, Bob could have produced such proof by himself. Therefore, a third party Charlie
 1954 will not be convinced that θ is true after seeing such proof transcript sent from Bob. This holds
 1955 even if Bob shares his secret key to Charlie, or if the key has been publicly leaked.

1956 Designated proofs are possible both in the interactive and non-interactive settings. In the interac-
 1957 tive setting (e.g., proving being the signer of an undeniable signature) the prover has the ability
 1958 to control when the verification takes place. However, in general (without a designated-verifier
 1959 approach) the prover may be unable to control who is able to verify the proof, namely if the verifier
 1960 is acting as a relay to another controlling party. The use of a designated proof has the potential

1961 to solve this problem.

1962 4.3 Previous works

1963 This section includes an overview of some of the works and applications existing in the zero-
1964 knowledge world. [Contribution needed: add more references.]

1965 ZKP protocols for anonymous credentials have been studied extensively in academic spaces [CKS10;
1966 BCDE+14; CDD17; BCDL+17; NVV18]. Products such as Miracl, Val:ID, Sovrin [Sov18], and
1967 LibZmix [Mik19] offer practical solutions to achieve privacy-preserving identity frameworks.

E58: C12.5

1968 Zerocash began as an academic work and was later developed into a product ensuring anonymous
1969 transactions [BCGG+14]. Baby ZoE enables Zerocash over Ethereum [zca18]. HAWK also uses
1970 zk-SNARKS to enable smart-contracts with transactional privacy [KMSWP16].

1971 4.4 Gadgets within predicates

1972 Formalizing the security of these protocols is a very difficult task, especially since there is no
1973 predetermined set of requirements, making it an ad-hoc process. Use-cases must be sure to dis-
1974 tinguish between privacy requirements and security guarantees. We discuss the use-case case of
1975 privacy-preserving asset transfer to illustrate the difference.

E59: C12.6

1976 Secure asset transfer is possible at several financial institutions, provided that the institution has
1977 knowledge of the identities of the sender, recipient, asset, and amount. In a privacy-preserving asset
1978 transfer, the identities of sender and recipient may be concealed even from the entity administering
1979 the transfer. It is important to note that a successful transfer must meet privacy requirements as
1980 well as provide security guarantees.

1981 Privacy requirements might include the anonymity of sender and recipient, concealment of asset
1982 type and asset amount. Security guarantees might include the inability of anyone besides the sender
1983 to initiate a transfer on the sender's behalf or the inability of a sender to execute a transfer of asset
1984 type without sufficient holdings of the asset.

1985 Here we outline a set of initial gadgets to be taken into account. See Table 4.1 for a simple list
1986 of gadgets — this list should be expanded continuously and on a case by case basis. For each of
1987 the gadgets we write the following representations, specifying what is the secret / witness, what is
1988 public / statement:

1989 NP statements for non-technical people:

**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 .**

1991 General form (Camenisch-Stadler): $\mathbf{Zk} \{ (\text{wit}): \mathbf{P}(\text{wit}, \text{statement}) \}$

1992 Example of ring signature: $\mathbf{Zk} \{ (\text{sig}): \mathbf{VerifySignature}(\mathbf{P1}, \text{sig}) \text{ or } \mathbf{VerifySignature}(\mathbf{P2},$

1993 sig) }

1994

Table 4.1: List of gadgets

E60: C14.1

1995

#	Gadget name	English 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

1996

1997

1998

1999

2000

2001

2002

2003

2004

Table 4.2: Commitment gadget (G1; envelope)

E61: C1.15

2005

2006

2008

2009

2010

2012

2013

2014

2016

2017

2018

2020

2021

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

2023

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

E62: C14.1

2024 2025	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2027 2028 2029	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	$\text{Verify}(VK, M, \sigma) = \text{True}$
2031 2032 2033	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 $\text{Verify}(VK, M, \sigma) = \text{True}$

2035

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

E63: C14.1

2036 2037	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2039 2040 2041	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

2043

Table 4.5: Distributed-decryption gadget (G4; envelope with a receiver stamp that requires multiple people to open)

E64: C14.1

2044

2045 2046	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2048 2049 2050	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

2052

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

2053 2054	Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
2056 2057	Verifiable random function (VRF)	VRF was computed from a secret seed and a public (or secret) input	Secret seed W	Input X , Output Y	$Y = \text{VRF}(W, X)$

2059

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

E65: C14.1

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

2070

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

Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
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'	$\forall j, Dec(SK, \pi(C_j)) = Dec(SK, C'_j)$
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	$\forall j, Dec(SK, \pi(C_j)) = P_j$

2078

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

Enhanced gadget (after adding ZKP)	ZKP statement (in a PoK notation)	Prover knows a witnessfor the public instances.t. the following predicate holds
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	$C(x, w) = y$
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 $z = C'(x, w)$	$C(x, w) = y$

2091 4.5 Identity framework

2092 4.5.1 Overview

2093 In this section we describe identity management solutions using zero knowledge proofs. The idea
2094 is that some user has a set of attributes that will be attested to by an issuer or multiple issuers,
2095 such that these attestations correspond to a validation of those attributes or a subset of them.

2096 After attestation it is possible to use this information, hereby called a credential, to generate a
2097 claim about those attributes. Namely, consider the case where Alice wants to show that she is
2098 over 18 and lives in a country that belongs to the European Union. If two issuers were responsible
2099 for the attestation of Alice's age and residence country, then we have that Alice could use zero
2100 knowledge proofs in order to show that she possesses those attributes, for instance she can use zero
2101 knowledge range proofs to show that her age is over 18, and zero knowledge set membership to
2102 prove that she lives in a country that belongs to the European Union. This proof can be presented
2103 to a Verifier that must validate such proof to authorize Alice to use some service. Hence there are
2104 three parties involved: (i) the credential holder; (ii) the credential issuer; (iii) and the verifier.

2105 4.5.2 Motivation for Identity and Zero Knowledge

2106 Digital identity has been a problem of interest to both academics and industry practitioners since
2107 the creation of the internet. Specifically, it is the problem of allowing an individual, a company,
2108 or an asset to be identified online without having to generate a physical identification for it, such
2109 as an ID card, a signed document, a license, etc. Digitizing Identity comes with some unique
2110 risks, loss of privacy and consequent exposure to Identity theft, surveillance, social engineering and
2111 other damaging efforts. Indeed, this is something that has been solved partially, with the help
2112 of cryptographic tools to achieve moderate privacy (password encryption, public key certificates,
2113 internet protocols like TLS and several others). Yet, these solutions are sometimes not enough
2114 to meet the privacy needs to the users / identities online. Cryptographic zero knowledge proofs
2115 can further enhance the ability to interact digitally and gain both privacy and the assurance of
2116 legitimacy required for the correctness of a process.

2117 The following is an overview of the generalized version of the identity scheme. We define the
2118 terminology used for the data structures and the actors, elaborate on what features we include and
2119 what are the privacy assurances that we look for.

2120 4.5.3 Terminology / Definitions

2121 In this protocol we use several different data structures to represent the information being trans-
2122 ferred or exchanged between the parties. We have tried to generalize the definitions as much as
2123 possible, while adapting to the existing Identity standards and previous ZKP works.

2124 **Attribute.** The most fundamental information about a holder in the system (e.g.: age, nation-
2125 ality, univ. Degree, pending debt, etc.). These are the properties that are factual and from which
2126 specific authorizations can be derived.

2127 **(Confidential and Anonymous) Credential.** The data structure that contains attribute(s)
 2128 about a holder in the system (e.g.: credit card statement, marital status, age, address, etc). Since
 2129 it contains private data, a credential is not shareable.

2130 **(Verifiable) Claim.** A zero-knowledge predicate about the attributes in a credential (or many of
 2131 them). A claim must be done about an identity and should contain some form of logical statement
 2132 that is included in the constraint system defined by the zk-predicate.

2133 **Proof of Credential.** The zero knowledge proof that is used to verify the claim attested by the
 2134 credential. Given that the credential is kept confidential, the proof derived from it is presented as
 2135 a way to prove the claim in question.

2136 The following are the different parties present in the protocol:

2137 **Holder.** The party whose attributes will be attested to. The holder holds the credentials that
 2138 contain his / her attributes and generates Zero Knowledge Proofs to prove some claim about these.
 2139 We say that the holder presents a proof of credential for some claim.

2140 **Issuer.** The party that attests attributes of holders. We say that the issuer issues a credential to
 2141 the holder.

2142 **Verifier.** The party that verifies some claim about a holder by verifying the zero knowledge proof
 2143 of credential to the claim.

2144 Remark: The main difference between this protocol and a non-ZK based Identity protocol is the
 2145 fact that in the latter, the holder presents the credentials themselves as the proof for the claim
 2146 / authorization, whereas in this protocol, the holder presents a zero knowledge proof that was
 2147 computed from the credentials.

2148 4.5.4 The Protocol Description

2149 **Functionality.** There are many interesting features that we considered as part of the identity
 2150 protocol. There are four basic functionalities that we decided to include from the get go:

- 2151 (1) third party anonymous and confidential attribute attestations through **credential issuance**
 2152 by the issuer;
- 2153 (2) confidentially proving claims using zero knowledge proofs through the **presentation of proof**
 2154 **of credential** by the holder;
- 2155 (3) **verification of claims** through zero knowledge proof verification by the verifier; and
- 2156 (4) unlinkable **credential revocation** by the issuer.

2157 There are further functionalities that we find interesting and worth exploring but that we did not
 2158 include in this version of the protocol. Some of these are credential transfer, authority delegation
 2159 and trace auditability. We explain more in detail what these are and explore ways they could be
 2160 instantiated.

2161 **Privacy requirements.** One should aim for a high level of privacy for each of the actors in
 2162 the system, but without compromising the correctness of the protocol. We look at anonymity
 2163 properties for each of the actors, confidentiality of their interactions and data exchanges, and
 2164 at the unlinkability of public data (in committed form). These usually can be instantiated as
 2165 cryptographic requirements such as commitment non-malleability, indistinguishability from random
 2166 data, unforgeability, accumulator soundness or as statements in zero-knowledge such as proving
 2167 knowledge of preimages, proving signature verification, etc.

- 2168 • Holder anonymity: the underlying physical identity of the holder must be hidden from the
 2169 general public, and if needed from the issuer and verifier too. For this we use pseudo-random
 2170 strings called identifiers, which are tied to a secret only known to the holder.
- 2171 • Issuer anonymity: only the holder should know what issuer issued a specific credential.
- 2172 • Anonymous credential: when a holder presents a credential, the verifier may not know who
 2173 issued the certificate. He / She may only know that the credential was issued by some
 2174 approved issuer.
- 2175 • Holder untraceability: the holder identifiers and credentials can't be used to track holders
 2176 through time.
- 2177 • Confidentiality: no one but the holder and the issuer should know what the credential at-
 2178 tributes are.
- 2179 • Identifier linkability: no one should be able to link two identifier unless there is a proof
 2180 presented by the holder.
- 2181 • Credential linkability: No one should be able to link two credentials from the publicly available
 2182 data. Mainly, no two issuers should be able to collude and link two credentials to one same
 2183 holder by using the holder's digital identity.

2184 **In depth view.** For the specific instantiation of the scheme, we examine in Tables 4.10–4.13
 2185 the different ways that these requirements can be achieved and what are the trade-offs to be done
 2186 (e.g.: using pairwise identifiers vs. one fixed public key; different revocation mechanisms; etc.) and
 2187 elaborate on the privacy and efficiency properties of each.

2188 **Functionalities vs. privacy and robustness requirements.** The following four tables de-
 2189 scribe, for four functionalities/problems, Several aspects of instantiation method, proof details and
 2190 privacy/robustness are described in the following four tables related to four functionalities/problems:

- 2191 • Table 4.10: Holder identification
- 2192 • Table 4.11: Issuer identification
- 2193 • Table 4.12: Credential issuance
- 2194 • Table 4.13: Credential revocation

E66: C1.16

2195

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

	Instantiation Method	Proof Details	Privacy / Robustness
2197 2198 2199 2200 2201 2202 2203 2204	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	<ul style="list-style-type: none"> - 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 	<ul style="list-style-type: none"> - 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
2206 2207 2208 2209 2210 2211 2212 2213	Single identifier in the self-sovereign realm: PRF based Public Key (idPK) self derived by the entity.	<ul style="list-style-type: none"> - 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 	<ul style="list-style-type: none"> - 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”
2215 2216 2217 2218 2219 2220 2221 2222	Multiple identifiers: Pairwise identification through identifiers. For each new interaction the holder generates a new identifier.	<ul style="list-style-type: none"> - 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. 	<ul style="list-style-type: none"> - Physical identity is hidden and does not necessarily have to be connected to the public key - Issuers cannot collude 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”

2224

Table 4.11: Issuer identification

	Instantiation Method	Proof Details	Privacy / Robustness
2226 2227 2228 2229 2230 2231 2232 2233 2234	Federated permissions: there is a list of approved issuers that can be updated by either a central authority or a set of nodes	<ul style="list-style-type: none"> - 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 	<ul style="list-style-type: none"> - The verifier / public would not know who the issuer of the credential is but would know it is approved.
2236 2237 2238 2239 2240 2241 2242 2243 2244	Free permissions: anyone can become an issuer, which use identifiers: <ul style="list-style-type: none"> - 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 	<ul style="list-style-type: none"> - 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. 	<ul style="list-style-type: none"> - 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 unlinkable to credentials

2246

Table 4.12: Credential Issuance

	Instantiation Method	Proof Details	Privacy / Robustness
2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260	Blind signatures: the issuer signs on a commitment of a self-attested credential after seeing a proof of correct attestation; a second kind of proof would be needed in the system	<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
2262 2263 2264 2265 2266 2267	In the clear signatures: the issuer generates the attestation, signing the commitment and sending the credential in the clear to the holder	<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

2269

Table 4.13: Credential Revocation

	Instantiation Method	Proof Details	Privacy / Robustness
2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281	Credential Revocation Positive accumulator revocation: the issuer revokes the credential by removing an element from an accumulator [BCDL+17]	<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
2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294	Negative accumulator revocation: the issuer revokes by adding an element to an accumulator	<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

2296 **Gadgets.** Each of the methods for instantiating the different functionalities use some of the
 2297 following gadgets that have been described in the Gadgets section. There are three main parts to
 2298 the predicate of any proof.

- 2299 1. The first is proving the veracity of the identity, in this case the holder, for which the following
 2300 gadgets can / should be used:
 - 2301 • **Commitment** for checking that the identity has been attested to correctly.
 - 2302 • **PRF** for proving the preimage of the identifier is known by the holder
 - 2303 • **Equality of strings** to prove that the new identifier has a connection to the previous
 2304 identifier used or to an approved identifier.
- 2305 2. Then there is the part of the constraint system that deals with the legitimacy of the creden-
 2306 tials, the fact that it was correctly issued and was not revoked.
 - 2307 • **Commitment** for checking that the credential was correctly committed to.
 - 2308 • **PRF** for proving that the holder knows the credential information, which is the preimage
 2309 of the commitment .
 - 2310 • **Equality of strings** to prove that the credential was issued to an identifier connected
 2311 to the current identifier.

- 2312 • **Accumulators (Set membership / non-membership)** to prove that the commit-
2313 ment to the credential exists in some set (usually an accumulator), implying that it was
2314 issued correctly and that it was not revoked.
- 2315 3. Finally there is the logic needed to verify the rules / constraints imposed on the attributes
2316 themselves. This part can be seen as a general gadget called “credentials”, which allows to
2317 verify the specific attributes embedded in a credential. Depending on the credential type, it
2318 uses the following low level gadgets:
 - 2319 • **Data Type** used to check that the data in the credential is of the correct type
 - 2320 • **Range Proofs** used to check that the data in the credential is within some range
 - 2321 • **Arithmetic Operations (field arithmetic, large integers, etc.)** used for verifying
2322 arithmetic operations were done correctly in the computation of the instance.
 - 2323 • **Logical Operators (bigger than, equality, etc.)** used for comparing some value in
2324 the instance to the data in the credentials or some computation derived from it.

2325 Security caveats

- 2326 1. If the Issuer colludes with the Verifier, they could use the revocation mechanism to reveal
2327 information about the Holder if there is real-time sharing of revocation information.
- 2328 2. Furthermore, if the commitments to credentials and the revocation information can be tracked
2329 publicly and the events are dependent of each other (e.g.: revocation by removing a commit-
2330 ment), then there can be linkability between issuance and revocation.
- 2331 3. In the case of self-attestation or collusion between the issuer and the holder, there is a much
2332 lower assurance of data integrity. The inputs to the ZKP could be spoofed and then the proof
2333 would not be sound.
- 2334 4. The use of Blockchains create a reliance on a trusted oracle for external state. On the other
2335 hand, the privacy guaranteed at blockchain-content level is orthogonal to network-level traffic
2336 analysis.

2337 4.5.5 A use-case example of credential aggregation

2338 We are going to focus our description on a specific use case: accredited investors. In this scenario
2339 the credential holder will be able to show that she is accredited without revealing more information
2340 than necessary to prove such a claim.

2341 **Use-case description.** As a way to illustrate the above protocol, we present a specific use-case
2342 and explicitly write the predicate of the proof. Mainly, there is an identity, Alice, who wants to
2343 prove to some company, Bob Inc. that she is an accredited investor, under the SEC rules, in order
2344 to acquire some company shares. Alice is the prover; the IRS, the AML entity and The Bank are
2345 all issuers; and Bob Inc. is the verifier.

2346 The different processes in the adaptation of the use-case are the following:

E67: C12.7

- 2347 1. Three confidential credentials are issued to Alice which represent the rules that we apply on
 2348 an entity to be an accredited investor¹:
- 2349 (a) The IRS issues a tax credential, C_0 , that testifies to the claim “from 1/1/2017 until
 2350 1/1/2018, Alice, with identifier X_0 , owes 0\$ to the IRS, with identifier Y ” and holds two
 2351 attributes: the net income of Alice, $\$income$, and a bit b such that $b = 1$ if Alice has
 2352 paid her taxes.
- 2353 (b) The AML entity issues a KYC credential, C_1 , that testifies to claim $T_1 :=$ “Alice, with
 2354 identifier X_1 , has NO relation to a (set of) blacklisted organization(s)”
- 2355 (c) The Bank issues a net-worth credential, C_2 , that testifies to claim $T_2 :=$ “Alice has a net
 2356 worth of V_{Alice} ”
- 2357 2. Alice then proves to Bob Inc. that:
- 2358 (a) “Alice’s identifier, X_{Bob} , is related to the identifiers, X_i for $i = 0, 1, 2$ that are connected
 2359 to the confidential credentials C_i ”
- 2360 (b) “I know the credentials, which are the preimage of some commitment, C_i , were issued
 2361 by the legitimate issuers”
- 2362 (c) “The credentials, which are the preimage of some commitment, C_i , that exist in an
 2363 accumulator, U , satisfy the three statements T_i ”

2364 **Instantiation details.** Based on the different options laid out in the table above, the following
 2365 have been used:

- 2366 • Holder identification: we instantiate the identifiers as a unique anonymous identifier, pub-
 2367 licKey
- 2368 • Issuance identification: the identity of the issuers is known to all the participants, who can
 2369 publicly verify the signature on the credentials they issue².
- 2370 • Credential issuance: credentials are issued by publishing a signed commitment to a positive
 2371 accumulator and sharing the credential in the clear to Alice.
- 2372 • Credential revocation: is done by removing the commitment of credential from a dynamic and
 2373 positive accumulator. Alice must prove membership of commitment to show her credential
 2374 was not revoked.
- 2375 • Credential verification: Bob Inc. then verifies the cryptographic proof with the instance.

2376 Note that the transfer of company shares as well as the issuance of company shares is outside of the
 2377 scope of this use-case, but one could use the “Asset Transfer” section of this document to provide
 2378 that functionality.
 2379

2380 On another note, the fact that the proving and verification keys were validated by the SEC is an
 2381 assurance to Bob Inc. that proof verification implies Alice is an accredited investor.

¹We assume that the SEC generates the constraint system for the accreditation rules as the circuit used to generate the proving and verification keys. In the real scenario, here are the [Federal Rules for accreditation](#).

²With public signature verification keys that are hard coded into the circuit

2382 **The Predicate**

- 2383 • Blue = publicly visible in protocol / statement
- 2384 • Red = secret witness, potentially shared between parties when proving

2385 **Definitions / Notation:**

2386 Public state: **Accumulator**, for issuance and revocation, which includes all the commitments to the
2387 credentials.

2388 **ConfCred** = Commitment to Cred = { **Revoke**, **certificateType**, **publicKey**, **Attribute(s)** }

2389 Where, again, the IRS, AML and Bank are authorities with well-known public keys. Alice's **pub-**
2390 **licKey** is her long term public key and one cannot create a new credential unless her long term ID
2391 has been endorsed. The goal of the scheme is for the holder to create a fresh **proof of confidential**
2392 **aggregated credentials to the claim of accredited investor**.

2393 IRS issues a **ConfCred_{IRS}** = Commitment(**openIRS**, **revokeIRS**, "IRS", **myID**, **\$Income**, **b**), **sigIRS**
2394 AML issues **ConfCred_{AML}** = Commitment(**openAML**, **revokeAML**, "AML", **myID**, "OK", **sigAML**

2395 Holder generates a fresh public key **freshCred** to serve as an ephemeral blinded aggregate credential,
2396 and a ZKP of the following:

2397 **ZkPoK**{ (witness: **myID**, **ConfCred_{IRS}**, **ConfCred_{AML}**, **sigIRS**, **sigAML**, **\$Income**, , **mySig**, **openIRS**,
2398 **openAML** statement: **freshCred**, **minIncomeAccredited**) : Predicate:

- 2399 - **ConfCred_{IRS}** is a commitment to the IRS credential (**open_{IRS}**, "IRS", **myID**, **\$Income**)
- 2400 - **ConfCred_{AML}** is the AML credential to (**open_{AML}**, "AML", **myID**, "OK")
- 2401 - **\$Income** >= **minIncomeAccredited**
- 2402 - **b** = 1 = "myID paid full taxes"
- 2403 - **mySig** is a signature on **freshCred** for **myID**
- 2404 - **ProveNonRevoke**()
- 2405 }

2406 Present the credential to relying party: **freshCred** and **zpk**.

2407 **ProveNonRevoke**(**rhIRS**, **w_hrIRS**, **rhAML**, **w_hrAML**, **a_IRS**

- 2408 • **revokeIRS**: revocation handler from IRS. Can be embedded as an attribute in **ConfCred_{IRS}**
2409 and is used to handle revocations.
- 2410 • **wit_{thIRS}**: accumulator witness of **revokeIRS**.
- 2411 • **revokeAML**: revocation handler from AML. Can be embedded as an attribute in **ConfCred_{AML}**
2412 and is used to handle revocations.
- 2413 • **wit_{thAML}**: accumulator witness of **revokeAML**.
- 2414 • **acc_{IRS}**: accumulator for IRS.

- 2415 • $\text{CommRevoke}_{\text{IRS}}$: commitment to $\text{revoke}_{\text{IRS}}$. The holder generates a new commitment for
2416 each revocation to avoid linkability of proofs.
 - 2417 • acc_{AML} : accumulator for AML.
 - 2418 • $\text{CommRevoke}_{\text{AML}}$: commitment to $\text{revoke}_{\text{AML}}$. The holder generates a new commitment for
2419 each revocation to avoid linkability of proofs.
- 2420 $\text{ZkPoK}\{$ (witness: $\text{rh}_{\text{IRS}}, \text{open}_{\text{rh}_{\text{IRS}}}, \text{w}_{\text{rh}_{\text{IRS}}}, \text{rh}_{\text{AML}}, \text{open}_{\text{rh}_{\text{AML}}}, \text{w}_{\text{rh}_{\text{AML}}}$ || statements: $\text{C}_{\text{IRS}}, \text{a}_{\text{IRS}},$
2421 $\text{C}_{\text{AML}}, \text{a}_{\text{AML}}$): Predicate:
- 2422 - C_{IRS} is valid commitment to ($\text{open}_{\text{rh}_{\text{IRS}}}, \text{rh}_{\text{IRS}}$)
 - 2423 - rh_{IRS} is part of accumulator a_{IRS} , under witness $\text{w}_{\text{rh}_{\text{IRS}}}$
 - 2424 - rh_{IRS} is an attribute in Cert_{IRS}
 - 2425 - C_{AML} is valid commitment to ($\text{open}_{\text{rh}_{\text{AML}}}, \text{rh}_{\text{AML}}$)
 - 2426 - rh_{AML} is part of accumulator a_{AML} , under witness $\text{w}_{\text{rh}_{\text{AML}}}$
 - 2427 - rh_{AML} is an attribute in Cert_{AML}
 - 2428 }
 - 2429 - myCred is unassociated with myID, with sigIRS, sigAML etc.
 - 2430 - Withstands partial compromise: even if IRS leaks myID and sigIRS, it cannot be used to
2431 reveal the sigAML or associated myID with myCred

2432 4.6 Asset Transfer

2433 4.6.1 Privacy-preserving asset transfers and balance updates

2434 In this section, we examine two use-cases involving using ZK Proofs (ZKPs) to facilitate private
2435 asset-transfer for transferring fungible or non-fungible digital assets. These use-cases are motivated
2436 by privacy-preserving cryptocurrencies, where users must prove that a transaction is valid, without
2437 revealing the underlying details of the transaction. We explore two different frameworks, and
2438 outline the technical details and proof systems necessary for each.

2439 There are two dominant paradigms for tracking fungible digital assets, tracking ownership of assets
2440 individually, and tracking account balances. The Bitcoin system introduced a form of asset-tracking
2441 known as the UTXO model, where Unspent Transaction Outputs correspond roughly to single-use
2442 “coins”. Ethereum, on the other hand, uses the balance model, and each account has an associated
2443 balance, and transferring funds corresponds to decrementing the sender’s balance, and incrementing
2444 the receiver’s balance accordingly.

2445 These two different models have different privacy implications for users, and have different rules
2446 for ensuring that a transaction is valid. Thus the requirements and architecture for building ZK
2447 proof systems to facilitate privacy-preserving transactions are slightly different for each model, and
2448 we explore each model separately below.

2449 In its simplest form, the asset-tracking model can be used to track non-fungible assets. In this
2450 scenario, a transaction is simply a transfer of ownership of the asset, and a transaction is valid if:
2451 the sender is the current owner of the asset. In the balance model (for fungible assets), each account
2452 has a balance, and a transaction decrements the sender's account balance while simultaneously
2453 incrementing the receivers. In a "balance" model, a transaction is valid if 1) The amount the
2454 sender's balance is decremented is equal to the amount the receiver's balance is incremented, 2)
2455 The sender's balance remains non-negative 3) The transaction is signed using the sender's key.

2456 4.6.2 Zero-Knowledge Proofs in the asset-tracking model

2457 In this section, we describe a simple ZK proof system for privacy-preserving transactions in the
2458 asset-tracking (UTXO) model. The architecture we outline is essentially a simplification of the
2459 ZCash system. The primary simplification is that we assume that each asset ("coin") is indivisible.
2460 In other words, each asset has an owner, but there is no associated value, and a transaction is
2461 simply a transfer of ownership of the asset.

2462 **Motivation:** Allow stakeholders to transfer non-fungible assets, without revealing the ownership
2463 of the assets publicly, while ensuring that assets are never created or destroyed.

2464 **Parties:** There are three types of parties in this system: a Sender, a Receiver and a distributed
2465 set of validators. The sender generates a transactions and a proof of validity. The (distributed)
2466 validators act as verifiers and check the validity of the transaction. The receiver has no direct role,
2467 although the sender must include the receiver's public-key in the transaction.

2468 **What is being proved:** At high level, the sender must prove three things to convince the
2469 validators that a transaction is valid.

- 2470 • The asset (or "note") being transferred is owned by the sender. (Each asset is represented by
2471 a unique string)
- 2472 • The sender proves that they have the private spending keys of the input notes, giving them
2473 the authority to send asset.
- 2474 • The private spending keys of the input assets are cryptographically linked to a signature over
2475 the whole transaction, in such a way that the transaction cannot be modified by a party who
2476 did not know these private keys.

2477 **What information is needed by the verifier:**

- 2478 • The verifiers need access to the CRS used by the proof system
- 2479 • The validators need access to the entire history of transactions (this includes all UTXOs,
2480 commitments and nullifiers as described later). This history can be stored on a distributed
2481 ledger (e.g. the Bitcoin blockchain)

2482 **Possible attacks:**

- 2483 • CRS compromise: If an attacker learns the private randomness used to generate the CRS,
2484 the attacker can forge proofs in the underlying system
- 2485 • Ledger attacks: validating a transaction requires reading the entire history of transactions,
2486 and thus a verifier with an incorrect view of the transaction history may be convinced to

2487 accept an incorrect transaction as valid.

- 2488 • Re-identification attacks: The purpose of incorporating ZKPs into this system is to facilitate
2489 transactions without revealing the identities of the sender and receiver. If anonymity is not
2490 required, ZKPs can be avoided altogether, as in Bitcoin. Although this system hides the
2491 sender and receiver of each transaction, the fact that a transaction occurred (and the time of
2492 its occurrence) is publicly recorded, and thus may be used to re-identify individual users.
- 2493 • IP-level attacks: by monitoring network traffic, an attacker could link transactions to spe-
2494 cific senders or receivers (each transaction requires communication between the sender and
2495 receiver) or link public-keys (pseudonyms) to real-world identities
- 2496 • Man-it-the-Middle attacks: An attacker could convince a sender to transfer an asset to an
2497 “incorrect” public-key

2498 **Setup scenario:** This system is essentially a simplified version of Zcash proof system, modified
2499 for indivisible assets. Each asset is represented by a unique AssetID, and for simplicity we assume
2500 that the entire set of assets has been distributed, and no assets are ever created or destroyed.

2501 At any given time, the public state of the system consists of a collection of “asset notes”. These notes
2502 are stored as leaves in a Merkle Tree, and each leaf represents a single indivisible asset represented
2503 by unique assetID. In more detail, a “note” is a commitment to {Nullifier, publicKey, assetID},
2504 indicating that publicKey “owns” assetID.

2505 **Main transaction type:** Sending an asset from Current Owner *A* to New Owner *B*

2506 **Security goals:**

- 2507 • Only the current owner can transfer the asset
- 2508 • Assets are never created or destroyed

2509 **Privacy goals:** Ideally, the system should hide all information about the ownership and trans-
2510 action patterns of the users. The system sketched below does not attain that such a high-level of
2511 privacy, but instead achieves the following privacy-preserving features

- 2512 • Transactions are publicly visible, i.e., anyone can see that a transaction occurred
- 2513 • Transactions do not reveal which asset is being transferred
- 2514 • Transactions do not reveal the identities (public-keys) of the sender or receiver.
 - 2515 – Limitation: Previous owner can tell when the asset is transferred. (Mitigation: after
2516 receiving asset, send it to yourself)

2517 **Details of a transfer:** Each transaction is intended to transfer ownership of an asset from a
2518 Current Owner to a New Owner. In this section, we outline the proofs used to ensure the validity
2519 of a transaction. Throughout this description, we use **Blue** to denote information that is globally
2520 and **publicly** visible in the protocol / statement. We use **Red** to denote **private** information, e.g.

Applications

2521 a secret witness held by the prover or information shared between the Current Owner and New
2522 Owner.

2523 The Current Owner, A , has the following information

- 2524 • A **publicKey** and corresponding **secretKey**
- 2525 • An **assetID** corresponding to the asset being transferred
- 2526 • A **note** in the **MerkleTree** corresponding to the asset
- 2527 • Knows how to open the **commitment** (**Nullifier**, **assetID**, **publicKey**) **publicKeyOut** of the new
2528 Owner B

2529 The Current Owner, A , generates

- 2530 • A new **NullifierOut**
- 2531 • A new commitment **commitment** (**NullifierOut**, **assetID**, **publicKey**)

2532 The Current owner, A , sends

- 2533 • Privately to B : **NullifierOut**, **publicKeyOut**, **assetID**
- 2534 • Publicly to the blockchain: **Nullifier**, **comOut**, **ZKProof** (the structure of **ZKProof** is outlined
2535 below)

2536 If **Nullifier** does not exist in **MerkleTree** and **ZKProof** validates, then **comOut** is added to the
2537 merkleTree.

2538 **The structure of the Zero-Knowledge Proof:** We use a modification of Camenisch-Stadler
2539 notation to describe the structure of the proof.

2540 Public state: **MerkleTree** of Notes: Note = **Commitment** to { **Nullifier**, **publicKey**, **assetID** }

2541 **ZKProof** = $ZkPoK_{pp}\{$

2542 (witness: **publicKey**, **publicKeyOut**, **merkleProof**, **NullifierOut**, **com**, **assetID**, **sig**

2543 statement: **MerkleTree**, **Nullifier**, **comOut**) :

2544 predicate:

- 2545 - **com** is included in **MerkleTree** (using **merkleProof**)
- 2546 - **com** is a commitment to (**Nullifier**, **publicKey**, **assetID**)
- 2547 - **comOut** is a commitment to (**NullifierOut**, **publicKeyOut**, **assetID**)
- 2548 - **sig** is a signature on **comOut** for **publicKey**

2549 }

2550 4.6.3 Zero-Knowledge proofs in the balance model

2551 In this section, we outline a simple system for privately transferring fungible assets, in the “balance
2552 model.” This system is essentially a simplified version of **zkLedger**. The state of the system is an

2553 (encrypted) account balance for each user. Each account balance is encrypted using an additively
 2554 homomorphic cryptosystem, under the account-holder’s key. A transaction decrements the sender’s
 2555 account balance, while incrementing the receiver’s account by a corresponding amount. If the
 2556 number of users is fixed, and known in advance, then a transaction can hide all information about
 2557 the sender and receiver by simultaneously updating all account balances. This provides a high-
 2558 degree of privacy, and is the approach taken by zkLedger. If the set of users is extremely large,
 2559 dynamically changing, or unknown to the sender, the sender must choose an “anonymity set” and
 2560 the transaction will reveal that it involved members of the anonymity set, but not the amount of the
 2561 transaction or which members of the set were involved. For simplicity of presentation, we assume
 2562 a model like zkLedger’s where the set of parties in the system is fixed, and known in advance, but
 2563 this assumption does not affect the details of the zero-knowledge proofs involved.

2564 **Motivation:** Each entity maintains a private account balance, and a transaction decrements the
 2565 sender’s balance and increments the receiver’s balance by a corresponding amount. We assume that
 2566 every transaction updates every account balance, thus all information the origin, destination and
 2567 value of a transaction will be completely hidden. The only information revealed by the protocol is
 2568 the fact that a transaction occurred.

2569 **Parties:**

- 2570 • A set of n stakeholders who wish to transfer fungible assets anonymously
- 2571 • The stakeholder who initiates the transaction is called the “prover” or the “sender”
- 2572 • The receiver, or receivers do not have a distinguished role in a transaction
- 2573 • A set of validators who maintain the (public) state of the system (e.g. using a blockchain or
 2574 other DLT).

2575 **What is being proved:** The sender must convince the validators that a proposed transaction is
 2576 “valid” and the state of the system should be updated to reflect the new transaction. A transaction
 2577 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:

- 2578 • The sum of all committed values is 0 (i.e., $x_1 + \dots + x_n = 0$)
- 2579 • The sender owns the private key corresponding to all negative x_i
- 2580 • After the update, all account balances remain positive

2581 What information is needed by the verifier:

- 2582 • The verifiers need access to the CRS used by the proof system
- 2583 • The verifiers need access to the current state of the system (i.e., the current vector of n
 2584 encrypted account balances). This state can be stored on a distributed ledger

2585 Possible attacks:

- 2586 • CRS compromise: If an attacker learns the private randomness used to generate the CRS,
 2587 the attacker can forge proofs in the underlying system
- 2588 • Ledger attacks: validating a transaction requires knowing the current state of the system
 2589 (encrypted account balances), thus a validator with an incorrect view of the current state
 2590 may be convinced to accept an incorrect transaction as valid.

Applications

- 2591 • Re-identification attacks: The purpose of incorporating ZKPs into this system is to facilitate
2592 transactions without revealing the identities of the sender and receiver. If anonymity is not
2593 required, ZKPs can be avoided altogether, as in Bitcoin. Although this system hides the
2594 sender and receiver of each transaction, the fact that a transaction occurred (and the time of
2595 its occurrence) is publicly recorded, and thus may be used to re-identify individual users.
- 2596 • IP-level attacks: by monitoring network traffic, an attacker could link transactions to specific
2597 senders or receivers (each transaction requires communication between the sender and the
2598 validators) or link public-keys (pseudonyms) to real-world identities
- 2599 • Man-it-the-Middle attacks: An attacker could convince a sender to transfer an asset to an
2600 “incorrect” public-key. This is perhaps less of a concern in the situation where the user-base
2601 is static, and all public-keys are known in advance.

2602 **Setup scenario:** There are fixed number of users, n . User i has a known public-key, pk_i . Each
2603 user has an account balance, maintained as an additively homomorphic encryption of their current
2604 balance under their pk . Each transaction is a list of n encryptions, corresponding to the amount
2605 each balance should be incremented or decremented by the transaction. To ensure money is never
2606 created or destroyed, the plaintexts in an encrypted transaction must sum to 0. We assume that
2607 all account balance are initialized to non-negative values.

2608 **Main transaction type:** Transferring funds from user i to user j

2609 Security goals:

- 2610 • An account balance can only be decremented by the owner of that account
- 2611 • Account balances always remain non-negative
- 2612 • The total amount of money in the system remains constant

2613 **Privacy goals:** Ideally, the system should hide all information about the ownership and trans-
2614 action patterns of the users. The system sketched below does not attain that such a high-level of
2615 privacy, but instead achieves the following privacy-preserving features:

- 2616 • Transactions are publicly visible, i.e., anyone can see that a transaction occurred
- 2617 • Transactions do not reveal which asset is being transferred
- 2618 • Transactions do not reveal the identities (public-keys) of the sender or receiver.

2619 Limitation: transaction times are leaked

2620 **Details of a transfer:** Each transaction is intended to update the current account balances
2621 in the system. In this section, we outline the proofs used to ensure the validity of a transaction.
2622 Throughout this description, we use **Blue** to denote information that is globally and **publicly** visible
2623 in the protocol / statement. We use **Red** to denote **private** information, e.g. a secret witness held
2624 by the prover.

2625 The Sender, A , has the following information

- 2626 • Public keys pk_1, \dots, pk_n
- 2627 • $secretKey_i$ corresponding to $publicKey_i$, and a values x_j , to transfer to user j
- 2628 • The sender's own current account balance, y_i

2629 The Sender, A , generates

- 2630 • a vector of ciphertexts, C_1, \dots, C_n with $C_t = Enc_{pk_t}(x_t)$

2631 The Sender, A , sends

- 2632 • The vector of ciphertexts C_1, \dots, C_n and $ZKProof$ (described below) to the blockchain

2633 ZK Circuit:

2634 Public state: The current state of the system, i.e., a vector of (encrypted) account balances,
2635 B_1, \dots, B_n .

2636 $ZKProof = ZkPoK_{pp}\{$ (witness: i, x_1, \dots, x_n, sk statement: C_1, \dots, C_n) :

2637 predicate:

- 2638 - C_t is an encryption to x_t under public key pk_t for $t = 1, \dots, n$
- 2639 - $x_1 + \dots + x_n = 0$
- 2640 - $x_t \geq 0$ OR sk corresponds to pk_t for $t = 1, \dots, n$
- 2641 - $x_t \geq 0$ OR current balance B_t encrypts a value no smaller than $|x_t|$ for $t = 1, \dots, n$
- 2642 }

2643 4.7 Regulation Compliance

2644 4.7.1 Overview

2645 An important pattern of applications in which zero-knowledge protocols are useful is within settings
2646 in which a regulator wishes to monitor, or assess the risk related to some item managed by a
2647 regulated party. One such example can be whether or not taxes are being paid correctly by an
2648 account holder, or is a bank or some other financial entity solvent, or even stable.

2649 The regulator in such cases is interested in learning “the bottom line”, which is typically derived
2650 from some aggregate measure on more detailed underlying data, but does not necessarily need to
2651 know all the details. For example, the answer to the question of “did the bank take on too many
2652 loans?” Is eventually answered by a single bit (Yes/No) and can be answered without detailing
2653 every single loan provided by the bank and revealing recipients, their income, and other related
2654 data.

2655 Additional examples of such scenarios include:

- 2656 - Checking that taxes have been properly paid by some company or person.

Applications

- 2657 – Checking that a given loan is not too risky.
- 2658 – Checking that data is retained by some record keeper (without revealing or transmitting the
2659 data)
- 2660 – Checking that an airplane has been properly maintained and is fit to fly

2661 The use of Zero knowledge proofs can then allow the generation of a proof that demonstrate the
2662 correctness of the aggregate result. The idea is to show something like the following statement:
2663 There is a commitment (possibly on a blockchain) to records that show that the result is correct.

2664 **Trusting data fed into the computation:** In order for a computation on hidden data to prove
2665 valuable, the data that is fed in must be grounded as well. Otherwise, proving the correctness
2666 of the computation would be meaningless. To make this point concrete: A credit score that was
2667 computed from some hidden data can be correctly computed from some financial records, but when
2668 these records are not exposed to the recipient of the proof, how can the recipient trust that they
2669 are not fabricated?

2670 Data that is used for proofs should then generally be committed to by parties that are separate
2671 from the prover, and that are not likely to be colluding with the prover. To continue our example
2672 from before: an individual can prove that she has a high credit score based on data commitments
2673 that were produced by her previous lenders (one might wonder if we can indeed trust previous
2674 lenders to accurately report in this manner, but this is in fact an assumption implicitly made in
2675 traditional credit scoring as well).

2676 The need to accumulate commitments regarding the operation and management of the processes
2677 that are later audited using zero-knowledge often fits well together with blockchain systems, in
2678 which commitments can be placed in an irreversible manner. Since commitments are hiding, such
2679 publicly shared data does not breach privacy, but can be used to anchor trust in the veracity of
2680 the data.

2681 **4.7.2 An example in depth: Proof of compliance for aircraft**

2682 An operator is flying an aircraft, and holds a log of maintenance operations on the aircraft. These
2683 records are on different parts that might be produced by different companies. Maintenance and
2684 flight records are attested to by engineers at various locations around the world (who we assume
2685 do not collude with the operator).

2686 The regulator wants to know that the aircraft is allowed to fly according to a certain set of rules.
2687 (Think of the Volkswagen emissions cheating story.)

2688 The problem: Today, the regulator looks at the records (or has an auditor do so) only once in a
2689 while. We would like to move to a system where compliance is enforced in “real time”, however,
2690 this reveals the real-time operation of the aircraft if done naively.

2691 Why is zero-knowledge needed? We would like to prove that regulation is upheld, without revealing
2692 the underlying operational data of the aircraft which is sensitive business operations. Regulators
2693 themselves prefer not to hold the data (liability and risk from loss of records), prefer to have
2694 companies self-regulate to the extent possible.

2695 What is the threat model beyond the engineers/operator not colluding? What about the parts

2696 manufacturers? Regulators? Is there an antagonistic relationship between the parts manufacturers?

2697 This scheme will work on regulation that isn't vague, such as aviation regulation. In some cases,
2698 the rules are vague on purpose and leave room for interpretation.

2699 4.7.3 Protocol high level

2700 Parties:

- 2701 • Operator / Party under regulation: performs operations that need to comply to a regulation.
2702 For example an airline operator that operates aircrafts
- 2703 • Risk bearer / Regulator : verifies that all regulated parties conform to the rules; updates the
2704 rules when risks evolve. For example, the FAA regulates and enforces that all aircrafts to
2705 be airworthy at all times. For an aircraft owner leasing their assets, they want to know that
2706 operation and maintenance does not degrade their asset. Same for a bank that financed an
2707 aircraft, where the aircraft is the collateral for the financing.
- 2708 • Issuer / 3rd party attesting to data: Technicians having examined parts, flight controllers
2709 attesting to plane arriving at various locations, embarked equipment providing signed readings
2710 of sensors.

2711 What is being proved:

- 2712 • The operator proves to the regulator that the latest maintenance data indicates the aircraft
2713 is airworthy
- 2714 • The operator proves to the bank that the aircraft maintenance status means it is worth a
2715 given value, according to a formula provided by that bank

2716 What are the privacy requirements?

- 2717 • An operator does not want to reveal the details of his operations and assets maintenance
2718 status to competition
- 2719 • The aircraft identity must be kept anonymous from all parties except the regulators and the
2720 technicians.
- 2721 • The technician's identity must be kept anonymous from the regulator but if needed the
2722 operator can be asked to open the commitments for the regulator to validate the reports

2723 **The proof predicate:** "The operator is the owner of the aircraft, and knows some signed data
2724 attesting to the compliance with regulation rules: all the components are safe to fly".

- 2725 • The plane is made up of the components x_1, \dots, x_n and for each of the components:
 - 2726 – There is an legitimate attestation by an engineer who checked the component, and signed
2727 it's OK
 - 2728 – The latest attestation by a technician is recent: the timestamp of the check was done
2729 before date D

2730 What is the public / private data:

- 2731 • Private:

Applications

- 2732 – Identity of the operator
- 2733 – Airplane record
- 2734 – Examination report of the technicians
- 2735 – Identity of the technician who signed the report

- 2736 • **Public:**

- 2737 – Commitment to airplane record

2738 There is a record for the airplane that is committed to a public ledger, which includes miles flown.
2739 There are records that attest to repairs / inspections by mechanics that are also committed to the
2740 ledger. The decommitment is communicated to the operator. These records reference the identifier
2741 of the plane.

2742 Whenever the plane flies, the old plane record needs to be invalidated, and a new one committed
2743 with extra mileage.

2744 When a proof of “airworthiness” is required, the operator proves that for each part, the mileage
2745 is below what requires replacement, or that an engineer replaced the part (pointing to a record
2746 committed by a technician).

2747 **At the gadget level:**

- 2748 • The prover proves knowledge of a de-commitment of an airplane record (decommitment)
- 2749 • The record is in the set of records on the blockchain (set membership)
- 2750 • and knowledge of de-commitments for records for the parts (decommitment) that are also in
2751 the set of commitments on the ledger (set membership)
- 2752 • The airplane record is not revoked (i.e., it is the most recent one), (requires set non-membership
2753 for the set of published nullifiers)
- 2754 • The id of the plane noted in the parts is the same as the id of the plane in the plane record.
2755 (equality)
- 2756 • The mileage of the plane is lower than the mileage needed to replace each part (range proofs)
2757 OTHERWISE
- 2758 • There exists a record (set membership) that says that the part was replaced by a technician
2759 (validate signature of the technician (maybe use ring signature outside of ZK?))

2760 4.8 Conclusions

- 2761 – The asset transfer and regulation can be used in the identity framework in a way that the
2762 additions complete the framework.
- 2763 – External oracles such as blockchain used for storing reference to data commitments

Page intentionally blank

Acknowledgments

2764

2765 The development of this community reference counts with the support of numerous individuals.

E69: C1.20

2766 **Version 0.** The “proceedings” of the 1st ZKProof workshop (Boston, May 2018) formed the initial
2767 basis for this document. The contributions were organized in three tracks:

2768 • **Implementation track.** Chairs: Sean Bowe, Kobi Gurkan, Eran Tromer. Participants:
2769 Benedikt Bünz, Konstantinos Chalkias, Daniel Genkin, Jack Grigg, Daira Hopwood, Jason
2770 Law, Andrew Poelstra, abhi shelat, Muthu Venkitasubramaniam, Madars Virza, Riad S.
2771 Wahby, Pieter Wuille.

2772 • **Applications Track.** Chairs: Daniel Benarroch, Ran Canetti, Andrew Miller. Participants:
2773 Shashank Agrawal, Tony Arcieri, Vipin Bharathan, Josh Cincinnati, Joshua Daniel, Anuj
2774 Das Gupta, Angelo De Caro, Michael Dixon, Maria Dubovitskaya, Nathan George, Brett
2775 Hemenway Falk, Hugo Krawczyk, Jason Law, Anna Lysyanskaya, Zaki Manian, Eduardo
2776 Morais, Neha Narula, Gavin Pacini, Jonathan Rouach, Kartheek Solipuram, Mayank Varia,
2777 Douglas Wikstrom, Aviv Zohar.

2778 • **Security track.** Chairs: Jens Groth, Yael Kalai, Muthu Venkitasubramaniam. Partici-
2779 pants: Nir Bitansky, Ran Canetti, Henry Corrigan-Gibbs, Shafi Goldwasser, Charanjit Jutla,
2780 Yuval Ishai, Rafail Ostrovsky, Omer Paneth, Tal Rabin, Maryana Raykova, Ron Rothblum,
2781 Alessandra Scafuro, Eran Tromer, Douglas Wikström.

2782 **Version 0.1.** Prior to the 2nd ZKProof workshop, the ZKProof organization team requested feed-
2783 back from NIST about the developed documentation. The NIST PEC team (Luís Brandão, René
2784 Peralta, Angela Robinson) then elaborated the “NIST comments on the initial ZKProof documen-
2785 tation” with 28 comments/suggestions for subsequent development of a “Community Reference
2786 Document”. Luís Brandão ported to LaTeX the proceedings into a LaTeX version, along with
2787 inline comments, which became named as version 0.1.

2788 **Version 0.2.** The contributions from version 0.1 to version 0.2 followed the editorial process
2789 initiated at the 2nd ZKProof Workshop (Berkeley, April 2019). Several suggested contributions
2790 stemmed from the breakout discussions in the workshop, which were possible by the collaboration
2791 of *scribes*, *moderators* and *participants*, as documented in the Workshop Notes [ZKP19]. The ac-
2792 tual content contributions were developed thereafter by several *contributors*, including Yu Hang,
2793 Eduardo Morais, Justin Thaler, Ivan Visconti, Riad Wahby and Yupeng Zhang, besides the NIST
2794 PEC team (Luís Brandão, René Peralta, Angela Robinson) and the Editors team (Daniel Benar-
2795 roch, Luís Brandão, Eran Tromer). The detailed description of the changes, contributions and
2796 contributors appears in the “diff” version of the community reference.

2797 **Miscellaneous.** A general “thank you” goes to all who have so far collaborated with the ZKProof
2798 initiative. This includes the workshop speakers, participants, organizers and sponsors, as well as the
2799 ZKProof steering committee and program committee members, and the participants in the online
2800 ZKProof forum. Detailed information about ZKProof is available on the zkproof.org website.

Page intentionally blank

References

- 2801
- 2802 [AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkatasubramanian. “Ligero: Lightweight
2803 Sublinear Arguments Without a Trusted Setup”. In: *Proceedings of the 2017 ACM*
2804 *SIGSAC Conference on Computer and Communications Security*. CCS ’17. Pub.
2805 by ACM, 2017, pp. 2087–2104. DOI: [10.1145/3133956.3134104](https://doi.org/10.1145/3133956.3134104). 22
- 2806 [BCDL+17] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin,
2807 and S. Yakoubov. “Accumulators with Applications to Anonymity-Preserving Re-
2808 vocation”. In: *2017 IEEE European Symposium on Security and Privacy (EuroSP)*.
2809 Apr. 2017, pp. 301–315. DOI: [10.1109/EuroSP.2017.13](https://doi.org/10.1109/EuroSP.2017.13). IACR Cryptology Eprint
2810 Archive: ia.cr/2017/043. 49, 58
- 2811 [BCGG+14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M.
2812 Virza. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *2014 IEEE*
2813 *Symposium on Security and Privacy*. May 2014, pp. 459–474. DOI: [10.1109/SP.2014.](https://doi.org/10.1109/SP.2014.36)
2814 [36](https://doi.org/10.1109/SP.2014.36). <http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf>. 38, 49
- 2815 [BCGT13] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. “On the Concrete Efficiency of
2816 Probabilistically-checkable Proofs”. In: *Proceedings of the Forty-fifth Annual ACM*
2817 *Symposium on Theory of Computing*. STOC ’13. Pub. by ACM, 2013, pp. 585–594.
2818 DOI: [10.1145/2488608.2488681](https://doi.org/10.1145/2488608.2488681). 23
- 2819 [BCGTV13] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. “SNARKs for C:
2820 Verifying Program Executions Succinctly and in Zero Knowledge”. In: *Advances in*
2821 *Cryptology – CRYPTO 2013*. Ed. by R. Canetti and J. A. Garay. Pub. by Springer
2822 Berlin Heidelberg, 2013, pp. 90–108. DOI: [10.1007/978-3-642-40084-1_6](https://doi.org/10.1007/978-3-642-40084-1_6). IACR
2823 Cryptology Eprint Archive: ia.cr/2013/507. 32, 33
- 2824 [BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. “Interactive Oracle Proofs”. In: *Theory*
2825 *of Cryptography*. Ed. by M. Hirt and A. Smith. Pub. by Springer Berlin Heidelberg,
2826 2016, pp. 31–60. DOI: [10.1007/978-3-662-53644-5_2](https://doi.org/10.1007/978-3-662-53644-5_2). IACR Cryptology Eprint
2827 Archive: ia.cr/2016/116. 22, 23
- 2828 [BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. “Scalable Zero Knowledge via
2829 Cycles of Elliptic Curves”. In: *Advances in Cryptology – CRYPTO 2014*. Ed. by J. A.
2830 Garay and R. Gennaro. Pub. by Springer Berlin Heidelberg, 2014, pp. 276–294. DOI:
2831 [10.1007/978-3-662-44381-1_16](https://doi.org/10.1007/978-3-662-44381-1_16). IACR Cryptology Eprint Archive: ia.cr/2014/595. 46
- 2832 [BCTV17] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. “Scalable Zero Knowledge Via
2833 Cycles of Elliptic Curves”. In: *Algorithmica* 79.4 (Dec. 2017), pp. 1102–1160. DOI:
2834 [10.1007/s00453-016-0221-0](https://doi.org/10.1007/s00453-016-0221-0). 47
- 2835 [BCDE+14] P. Bichsel, J. Camenisch, M. Dubovitskaya, R. R. Enderlein, S. Krenn, I. Krontiris,
2836 A. Lehmann, G. Neven, J. D. Nielsen, C. Paquin, F.-S. Preiss, K. Rannenberg,
2837 A. Sabouri, and M. Stausholm. *D2.2 - Architecture for Attribute-based Credential*
2838 *Technologies - Final Version*. Ed. by A. Sabour. Aug. 2014. [https://abc4trust.eu/](https://abc4trust.eu/download/Deliverable_D2.2.pdf)
2839 [download/Deliverable_D2.2.pdf](https://abc4trust.eu/download/Deliverable_D2.2.pdf). 49

- 2840 [BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. “Recursive Composition and
2841 Bootstrapping for SNARKS and Proof-carrying Data”. In: *Proceedings of the Forty-*
2842 *fifth Annual ACM Symposium on Theory of Computing*. STOC ’13. Pub. by ACM,
2843 2013, pp. 111–120. DOI: [10.1145/2488608.2488623](https://doi.org/10.1145/2488608.2488623). 47
- 2844 [BCIOP13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. “Succinct Non-
2845 interactive Arguments via Linear Interactive Proofs”. In: *Theory of Cryptography*.
2846 Ed. by A. Sahai. 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). IACR Cryptology Eprint Archive: ia.cr/2012/718. 21, 22
- 2848 [BISW17] D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu. “Lattice-Based SNARGs and Their
2849 Application to More Efficient Obfuscation”. In: *Advances in Cryptology – EURO-*
2850 *CRYPT 2017*. Ed. by J.-S. Coron and J. B. Nielsen. Pub. by Springer International
2851 Publishing, 2017, pp. 247–277. DOI: [10.1007/978-3-319-56617-7_9](https://doi.org/10.1007/978-3-319-56617-7_9). IACR Cryptol-
2852 ogy Eprint Archive: ia.cr/2017/240. 22
- 2853 [BCCGP16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. “Efficient Zero-Knowledge
2854 Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: *Advances in*
2855 *Cryptography – EUROCRYPT 2016*. Ed. by M. Fischlin and J.-S. Coron. Pub. by
2856 Springer Berlin Heidelberg, 2016, pp. 327–357. DOI: [10.1007/978-3-662-49896-](https://doi.org/10.1007/978-3-662-49896-5_12)
2857 [5_12](https://doi.org/10.1007/978-3-662-49896-5_12). IACR Cryptology Eprint Archive: ia.cr/2016/263. 22
- 2858 [BCGGHJ17] J. Bootle, A. Cerulli, E. Ghadafi, J. Groth, M. Hajiabadi, and S. K. Jakobsen.
2859 “Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability”. In: *Ad-*
2860 *vances in Cryptology – ASIACRYPT 2017*. Ed. by T. Takagi and T. Peyrin. Pub.
2861 by Springer International Publishing, 2017, pp. 336–365. DOI: [10.1007/978-3-319-](https://doi.org/10.1007/978-3-319-70700-6_12)
2862 [70700-6_12](https://doi.org/10.1007/978-3-319-70700-6_12). IACR Cryptology Eprint Archive: ia.cr/2017/872. 22
- 2863 [BCGJM18] J. Bootle, A. Cerulli, J. Groth, S. Jakobsen, and M. Maller. “Arya: Nearly Linear-
2864 Time Zero-Knowledge Proofs for Correct Program Execution”. In: *Advances in*
2865 *Cryptography – ASIACRYPT 2018*. Ed. by T. Peyrin and S. Galbraith. Pub. by
2866 Springer International Publishing, 2018, pp. 595–626. DOI: [10.1007/978-3-030-](https://doi.org/10.1007/978-3-030-03326-2_20)
2867 [03326-2_20](https://doi.org/10.1007/978-3-030-03326-2_20). 20
- 2868 [CDD17] J. Camenisch, M. Drijvers, and M. Dubovitskaya. “Practical UC-Secure Delegatable
2869 Credentials with Attributes and Their Application to Blockchain”. In: *Proceedings*
2870 *of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.
2871 CCS ’17. Pub. by ACM, 2017, pp. 683–699. DOI: [10.1145/3133956.3134025](https://doi.org/10.1145/3133956.3134025). 49
- 2872 [CKS10] J. Camenisch, M. Kohlweiss, and C. Soriente. “Solving Revocation with Efficient
2873 Update of Anonymous Credentials”. In: *Security and Cryptography for Networks*.
2874 Ed. by J. A. Garay and R. De Prisco. Pub. by Springer Berlin Heidelberg, 2010,
2875 pp. 454–471. DOI: [10.1007/978-3-642-15317-4_28](https://doi.org/10.1007/978-3-642-15317-4_28). 49
- 2876 [CCHL+19] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum,
2877 and D. Wichs. “Fiat-Shamir: From Practice to Theory”. In: *Proceedings of the 51st*
2878 *Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. Pub. by
2879 ACM, 2019, pp. 1082–1090. DOI: [10.1145/3313276.3316380](https://doi.org/10.1145/3313276.3316380). 23
- 2880 [CT10] A. Chiesa and E. Tromer. “Proof-Carrying Data and Hearsay Arguments from
2881 Signature Cards”. In: *Innovations in Computer Science — ICS 2010*. Vol. 10. 2010,
2882 pp. 310–331. 46, 47

References

- 2883 [CD98] R. Cramer and I. Damgård. “Zero-knowledge proofs for finite field arithmetic, or:
2884 Can zero-knowledge be for free?” In: *Advances in Cryptology — CRYPTO ’98*.
2885 Ed. by H. Krawczyk. Pub. by *Springer Berlin Heidelberg*, 1998, pp. 424–441. DOI:
2886 [10.1007/BFb0055745](https://doi.org/10.1007/BFb0055745). 21
- 2887 [DFKP16] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. “Cinderella: Turning
2888 Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of
2889 Verifiable Computation”. In: *2016 IEEE Symposium on Security and Privacy (SP)*.
2890 May 2016, pp. 235–254. DOI: [10.1109/SP.2016.22](https://doi.org/10.1109/SP.2016.22). 38
- 2891 [GGPR13a] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. “Quadratic Span Programs
2892 and Succinct NIZKs without PCPs”. In: *Advances in Cryptology – EUROCRYPT
2893 2013*. Ed. by T. Johansson and P. Q. Nguyen. Pub. by *Springer Berlin Heidelberg*,
2894 2013, pp. 626–645. DOI: [10.1007/978-3-642-38348-9_37](https://doi.org/10.1007/978-3-642-38348-9_37). IACR Cryptology Eprint
2895 Archive: ia.cr/2012/215. 22, 23
- 2896 [GGPR13b] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. “Quadratic Span Programs
2897 and Succinct NIZKs without PCPs”. In: *Advances in Cryptology – EUROCRYPT
2898 2013*. Ed. by T. Johansson and P. Q. Nguyen. Pub. by *Springer Berlin Heidelberg*,
2899 2013, pp. 626–645. DOI: [10.1007/978-3-642-38348-9_37](https://doi.org/10.1007/978-3-642-38348-9_37). IACR Cryptology Eprint
2900 Archive: ia.cr/2012/215. 32
- 2901 [GMO16] I. Giacomelli, J. Madsen, and C. Orlandi. “ZKBoo: Faster Zero-Knowledge for
2902 Boolean Circuits”. In: *25th USENIX Security Symposium (USENIX Security 16)*.
2903 Pub. by *USENIX Association*, 2016, pp. 1069–1083. 22
- 2904 [Gol13] O. Goldreich. “A Short Tutorial of Zero-Knowledge”. In: *Secure Multi-Party Com-
2905 putation*. Ed. by M. M. Prabhakaran and A. Sahai. Vol. 10. Cryptology and In-
2906 formation Security Series. 2013, pp. 28–60. DOI: [10.3233/978-1-61499-169-4-28](https://doi.org/10.3233/978-1-61499-169-4-28).
2907 24
- 2908 [GMW91] O. Goldreich, S. Micali, and A. Wigderson. “Proofs That Yield Nothing but Their
2909 Validity or All Languages in NP Have Zero-knowledge Proof Systems”. In: *J. ACM*
2910 38.3 (July 1991), pp. 690–728. DOI: [10.1145/116825.116852](https://doi.org/10.1145/116825.116852). 7
- 2911 [GMR89] S. Goldwasser, S. Micali, and C. Rackoff. “The Knowledge Complexity of Interactive
2912 Proof Systems”. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI:
2913 [10.1137/0218012](https://doi.org/10.1137/0218012). 1
- 2914 [GKR15] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. “Delegating Computation: Inter-
2915 active Proofs for Muggles”. In: *J. ACM* 62.4 (Sept. 2015), 27:1–27:64. DOI: [10.1145/
2916 2699436](https://doi.org/10.1145/2699436). 22
- 2917 [Gro10] J. Groth. “Short Non-interactive Zero-Knowledge Proofs”. In: *Advances in Cryptol-
2918 ogy - ASIACRYPT 2010*. Ed. by M. Abe. Pub. by *Springer Berlin Heidelberg*, 2010,
2919 pp. 341–358. DOI: [10.1007/978-3-642-17373-8_20](https://doi.org/10.1007/978-3-642-17373-8_20). 20
- 2920 [Gro16] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: *Advances
2921 in Cryptology – EUROCRYPT 2016*. Ed. by M. Fischlin and J.-S. Coron. Pub. by
2922 *Springer Berlin 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).
2923 IACR Cryptology Eprint Archive: ia.cr/2016/260. 23, 41
- 2924 [GOS06] J. Groth, R. Ostrovsky, and A. Sahai. “Perfect Non-interactive Zero Knowledge for
2925 NP”. In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by S. Vaudenay. LNC.
2926 Pub. by *Springer Berlin Heidelberg*, 2006, pp. 339–358. DOI: [10.1007/11761679_21](https://doi.org/10.1007/11761679_21). 24

- 2927 [IKOS07] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. “Zero-knowledge from Secure
2928 Multiparty Computation”. In: *Proceedings of the Thirty-ninth Annual ACM Sym-*
2929 *posium on Theory of Computing*. STOC '07. Pub. by ACM, 2007, pp. 21–30. DOI:
2930 [10.1145/1250790.1250794](https://doi.org/10.1145/1250790.1250794). 22
- 2931 [IMS12] Y. Ishai, M. Mahmoody, and A. Sahai. “On Efficient Zero-Knowledge PCPs”. In:
2932 *Theory of Cryptography*. Ed. by R. Cramer. Pub. by Springer Berlin Heidelberg,
2933 2012, pp. 151–168. DOI: [10.1007/978-3-642-28914-9_9](https://doi.org/10.1007/978-3-642-28914-9_9). 21
- 2934 [JSI96] M. Jakobsson, K. Sako, and R. Impagliazzo. “Designated Verifier Proofs and Their
2935 Applications”. In: *Advances in Cryptology — EUROCRYPT '96*. Ed. by U. Maurer.
2936 Pub. by Springer Berlin Heidelberg, 1996, pp. 143–154. DOI: [10.1007/3-540-68339-](https://doi.org/10.1007/3-540-68339-9_13)
2937 [9_13](https://doi.org/10.1007/3-540-68339-9_13). 48, 97
- 2938 [KR08] Y. T. Kalai and R. Raz. “Interactive PCP”. In: *Proceedings of the 35th International*
2939 *Colloquium on Automata, Languages and Programming, Part II*. ICALP '08. Pub.
2940 by Springer-Verlag, 2008, pp. 536–547. DOI: [10.1007/978-3-540-70583-3_44](https://doi.org/10.1007/978-3-540-70583-3_44). 21
- 2941 [Kil95] J. Kilian. “Improved Efficient Arguments”. In: *Advances in Cryptology — CRYPTO'95*.
2942 Ed. by D. Coppersmith. Vol. 1070. LNCS. Pub. by Springer Berlin Heidelberg,
2943 1995, pp. 311–324. DOI: [10.1007/3-540-44750-4_25](https://doi.org/10.1007/3-540-44750-4_25). 21, 23
- 2944 [KMSWP16] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. “Hawk: The Blockchain
2945 Model of Cryptography and Privacy-Preserving Smart Contracts”. In: *2016 IEEE*
2946 *Symposium on Security and Privacy (SP)*. May 2016, pp. 839–858. DOI: [10.1109/](https://doi.org/10.1109/SP.2016.55)
2947 [SP.2016.55](https://doi.org/10.1109/SP.2016.55). 49
- 2948 [Mic00] S. Micali. “Computationally Sound Proofs”. In: *SIAM J. Comput.* 30.4 (Oct. 2000),
2949 pp. 1253–1298. DOI: [10.1137/S0097539795284959](https://doi.org/10.1137/S0097539795284959). 21
- 2950 [Mik19] Mikelodder7/Ursa. *Z-mix*. 2019. <https://github.com/mikelodder7/ursa/tree/master/libzmix>. 49
- 2951 [NVV18] N. Narula, W. Vasquez, and M. Virza. “zkLedger: Privacy-Preserving Auditing for
2952 Distributed Ledgers”. In: *15th USENIX Symposium on Networked Systems Design*
2953 *and Implementation (NSDI 18)*. Pub. by USENIX Association, 2018, pp. 65–80.
2954 IACR Cryptology Eprint Archive: ia.cr/2018/241. 49
- 2955 [PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova. “Pinocchio: Nearly Practical
2956 Verifiable Computation”. In: *2013 IEEE Symposium on Security and Privacy*. May
2957 2013, pp. 238–252. DOI: [10.1109/SP.2013.47](https://doi.org/10.1109/SP.2013.47). IACR Cryptology Eprint Archive:
2958 ia.cr/2013/279. 32, 33
- 2959 [RRR16] O. Reingold, G. N. Rothblum, and R. D. Rothblum. “Constant-round Interactive
2960 Proofs for Delegating Computation”. In: *Proceedings of the Forty-eighth Annual*
2961 *ACM Symposium on Theory of Computing*. STOC '16. Pub. by ACM, 2016, pp. 49–
2962 62. DOI: [10.1145/2897518.2897652](https://doi.org/10.1145/2897518.2897652). 21, 23
- 2963 [Sch90] C. P. Schnorr. “Efficient Identification and Signatures for Smart Cards”. In: *Ad-*
2964 *vances in Cryptology — EUROCRYPT '89*. Ed. by J.-J. Quisquater and J. Vande-
2965 walle. Vol. 434. LNCS. Pub. by Springer Berlin Heidelberg, 1990, pp. 688–689. DOI:
2966 [10.1007/3-540-46885-4_68](https://doi.org/10.1007/3-540-46885-4_68). 6
- 2967 [Sov18] F. Sovrin. *SovrinTM: A Protocol and Token for Self-Sovereign Identity and Decen-*
2968 *tralized Trust*. Jan. 2018. [https://sovrin.org/wp-content/uploads/2018/03/Sovrin-](https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf)
2969 [Protocol-and-Token-White-Paper.pdf](https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf). 49

References

- 2970 [WTSTW18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. “Doubly-efficient
2971 zkSNARKs without trusted setup”. In: *2018 IEEE Symposium on Security and Pri-
2972 vacy (SP)*. IEEE. 2018, pp. 926–943. IACR Cryptology Eprint Archive: ia.cr/2017/1132.
2973 [22, 23](#)
- 2974 [XZZPS19] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. “Libra: Succinct Zero-
2975 Knowledge Proofs with Optimal Prover Computation”. In: *Advances in Cryptology -
2976 CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara,
2977 CA, USA, August 18-22, 2019, Proceedings, Part III*. 2019, pp. 733–764. DOI: [10.
2978 1007/978-3-030-26954-8_24](https://doi.org/10.1007/978-3-030-26954-8_24). IACR Cryptology Eprint Archive: ia.cr/2019/317. [23](#)
- 2979 [zca18] zcash-hackworks/babyzoe. *Baby ZoE - first step towards Zerocash over Ethereum*.
2980 2018. <https://github.com/zcash-hackworks/babyzoe>. [49](#)
- 2981 [ZGKPP17] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. “vSQL:
2982 Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases”. In: *2017
2983 IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 863–880. DOI:
2984 [10.1109/SP.2017.43](https://doi.org/10.1109/SP.2017.43). [22](#)
- 2985 [ZGKPP18] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. “vRAM:
2986 Faster Verifiable RAM with Program-Independent Preprocessing”. In: *2018 IEEE
2987 Symposium on Security and Privacy (SP)*. May 2018, pp. 908–925. DOI: [10.1109/
2988 SP.2018.00013](https://doi.org/10.1109/SP.2018.00013). [20](#)
- 2989 [ZKP19] ZKProof. *Notes of the 2nd ZKProof Workshop*. Ed. by D. Benarroch, L. T. A. N.
2990 Brandão, and E. Tromer. Pub. by zkproof.org, Dec. 2019. (The workshop was held
2991 at Berkeley, USA, in April 2019). [73](#)

Page intentionally blank

2992 Appendix A. Acronyms and glossary

2993 A.1 Acronyms

- | | | | |
|------|--|------|---|
| 2994 | • 3SAT: 3-satisfiability | 3010 | • PKI: public-key infrastructure |
| 2995 | • AND: AND gate (Boolean gate) | 3011 | • QAP: quadratic arithmetic program |
| 2996 | • API: application program interface | 3012 | • R1CS: rank-1 constraint system |
| 2997 | • CRH: collision-resistant hash (function) | 3013 | • RAM: random access memory |
| 2998 | • CRS: common-reference string | 3014 | • RSA: Rivest–Shamir–Adleman |
| 2999 | • DAG: directed acyclic graph | 3015 | • SHA: secure hash algorithm |
| 3000 | • DSL: domain specific languages | 3016 | • SMPC: secure multiparty computation |
| 3001 | • FFT: fast-Fourier transform | 3017 | • SNARG: succinct non-interactive argument |
| 3002 | • ILC: ideal linear commitment | 3018 | • SNARK: SNARG of knowledge |
| 3003 | • IOP: interactive oracle proofs | 3019 | • SRS: structured reference string |
| 3004 | • LIP: linear interactive proofs | 3020 | • UC: universal composability or universally composable |
| 3005 | • MA: Merlin–Arthur | 3021 | • URS: uniform random string |
| 3006 | • NIZK: non-interactive zero-knowledge | 3022 | • XOR: eXclusive OR (Boolean gate) |
| 3007 | • NP: non-deterministic polynomial | 3023 | • ZK: zero knowledge |
| 3008 | • PCD: proof-carrying data | 3024 | • ZKP: zero-knowledge proof |
| 3009 | • PCP: probabilistic checkable proof | 3025 | |

3026 A.2 Glossary

- 3027 • **NIZK:** Non-Interactive Zero-Knowledge. Proof system, where the prover sends a single message
3028 to the verifier, who then decides to accept or reject. Usually set in the common reference string
3029 model, although it is also possible to have designated verifier NIZK proofs.
- 3030 • **SNARK:** Succinct Non-interactive ARgument of Knowledge. A special type of non-interactive
3031 proof system where the proof size is small and verification is fast.
- 3032 • **Instance:** Public input that is known to both prover and verifier. Notation: x . (Some scientific
3033 articles use “instance” and “statement” interchangeably, but we distinguish between the two.)
- 3034 • **Witness:** Private input to the prover. Others may or may not know something about the
3035 witness. Notation: w .
- 3036 • **Application Inputs:** Parts of the witness interpreted as inputs to an application, coming from
3037 an external data source. The complete witness and the instance can be computed by the prover
3038 from application inputs.
- 3039 • **Relation:** Specification of relationship between instances and witness. A relation can be viewed
3040 as a set of permissible pairs (instance, witness). Notation: R .
- 3041 • **Language:** Set of instances that have a witness in R . Notation: L .

- 3042 • **Statement:** Defined by instance and relation. Claims the instance has a witness in the relation,
3043 which is either true or false. Notation: $x \in L$.
- 3044 • **Constraint System:** a language for specifying relations.
- 3045 • **Proof System:** A zero-knowledge proof system is a specification of how a prover and verifier
3046 can interact for the prover to convince the verifier that the statement is true. The proof system
3047 must be complete, sound and zero-knowledge.
 - 3048 – *Complete:* If the statement is true and both prover and verifier follow the protocol; the verifier
3049 will accept.
 - 3050 – *Sound:* If the statement is false, and the verifier follows the protocol; he will not be convinced.
 - 3051 – *Zero-knowledge:* If the statement is true and the prover follows the protocol; the verifier will
3052 not learn any confidential information from the interaction with the prover but the fact the
3053 statement is true.
- 3054 • **Backend:** an implementation of ZK proof system’s low-level cryptographic protocol.
- 3055 • **Frontend:** means to express ZK statements in a convenient language and to prove such state-
3056 ments in zero knowledge by compiling them into a low-level representation and invoking a suitable
3057 ZK backend.
- 3058 • **Instance reduction:** conversion of the instance in a high-level statement to an instance for a
3059 low-level statement (suitable for consumption by the backend), by a frontend.
- 3060 • **Witness reduction:** conversion of the witness to a high-level statement to witness for a low-level
3061 statement (suitable for consumption by the backend), by a frontend.
- 3062 • **R1CS (Rank 1 Constraint Systems):** an NP-complete language for specifying relations,
3063 as system of bilinear constraints (i.e., a rank 1 quadratic constraint system), as defined in
3064 [BCGTV13, Appendix E in extended version]. This is a more intuitive reformulation of QAP.
- 3065 • **QAP (Quadratic Arithmetic Program):** An NP-complete language for specifying relations
3066 via a quadratic system in polynomials, defined in [PHGR13]. See R1CS for an equivalent formu-
3067 lation.
- 3068 **Reference strings:**
 - 3069 • **CRS (Common Reference String):** A string output by the NIZK’s Generator algorithm,
3070 and available to both the prover and verifier. Consists of proving parameters and verification
3071 parameters. May be a URS or an SRS.
 - 3072 • **URS (Uniform Random String):** A common reference string created by uniformly sampling
3073 from some space, and in particular involving no secrets in its creation. (Also called Common
3074 Random String in prior literature; we avoid this term due to the acronym clash with Common
3075 Reference String).
 - 3076 • **SRS (Structured Reference String):** A common reference string created by sampling from
3077 some complex distribution, often involving a sampling algorithm with internal randomness that
3078 must not be revealed, since it would create a trapdoor that enables creation of convincing proofs
3079 for false statements. The SRS may be non-universal (depend on the specific relation) or universal
3080 (independent of the relation, i.e., serve for proving all of NP).
 - 3081 • **PP (Prover Parameters) or Proving Key:** The portion of the Common Reference String
3082 that is used by the prover.
 - 3083 • **VP (Verifier Parameters) or Verification Key:** The portion of the Common Reference
3084 String that is used by the verifier.

Appendix B. Version history

3085

3086 The development of the ZKProof Community reference can be tracked across a sequence of main
3087 versions. Here is a summarized description of their sequence:

E70: C1.21

3088 • **Version 0 [2018-08-01]: Baseline documents.** The proceedings of the 1st ZKProof
3089 Workshop (May 2018), with contributions settled by 2018-08-01 and available at [ZKProof.org](https://zkproof.org),
3090 along with the [ZKProof Charter](#), constitute the starting point of the ZKProof Community
3091 reference. Each of the three Workshop tracks — security, applications, implementation —
3092 lead to a corresponding proceedings document, named “ZKProof Standards (*track name*)
3093 Track Proceedings”. The ZKProof charter is also part of the baseline documents.

3094 • **Version 0.1 [2019-04-11]: LaTeX/PDF compilation.** Upon the ZKProof organization
3095 team requested feedback from the NIST-PEC team, the content in the several proceedings was
3096 ported to LaTeX code and compiled into a single PDF document entitled “ZKProof Commu-
3097 nity Reference” (version 0.1) for presentation and discussion at the 2nd ZKProof workshop.
3098 The version includes editorial adjustments for consistent style and easier indexation.

3099 • **Version 0.2 [2019-12-31]: Consolidated draft.** The process of consolidating the draft
3100 community reference document started at the 2nd ZKProof workshop (April 2019), where an
3101 editorial process was introduced and several “breakout sessions” were held for discussion on
3102 focused topics, including the “NIST comments on the initial ZKProof documentation”. The
3103 discussions yielded suggestions of topics to develop and incorporate in a new version of the
3104 document. Several concrete items of “proposed contributions” were then defined as GitHub
3105 issues, and the subsequently submitted contributions provided several content improvements,
3106 such as: distinguish ZKPs of knowledge vs. of membership; recommend security parameters
3107 for benchmarks; clarify some terminology related to ZKP systems (e.g., statements, CRS,
3108 R1CS); discuss interactivity vs. non-interactivity, and transferability vs. deniability; clarify
3109 the scope of use-cases and applications; update the “gadgets” table; add new references. The
3110 new version also includes numerous editorial improvements towards a consolidated document,
3111 namely a substantially reformulated frontmatter with several new sections (abstract, open to
3112 contributions, change log, acknowledgments, intellectual property, executive summary), a
3113 reorganized structure with a new chapter (still to be completed) on construction paradigms.
3114 The changes are tracked in a “diff” version of the document.

3115 **External resources.** Additional documentation covering the history of development of this com-
3116 munity reference can be found in the following online resources:

- 3117 • ZKProof GitHub repository: <https://github.com/zkstandard/>
- 3118 • ZKProof documentation: <https://zkproof.org/documents.html>
- 3119 • ZKProof Forum: <https://community.zkproof.org/>

Page intentionally blank

Tables of contribution descriptions v0.1 → v0.2

The following pages describe contributions integrated in the process of upgrading the draft reference document from version 0.1 (dated 2019-04-11, available during the 2nd ZKProof Workshop) to version 0.2.

Explanation of the tables of contributions

Each table describes proposed contributions and corresponding edits in comparison with the baseline version 0.1, in order to achieve version 0.2. Each table, indexed as Cx (where x is an integer), corresponds to a [GitHub issue](#) (GIy , where y is an integer) describing proposed contributions — see <https://github.com/zkpstandard/zkreference/issues>. However, compared with GitHub, the description here may have been adjusted for a better explanation and cross-referencing of the actual edits made in the document. Each table has a header as follows:

#	Item id	Location	Contribution topic Cx : <i>short description</i>	Related	Incorporated changes	Edit id
---	---------	----------	--	---------	----------------------	---------

From left to right, the columns represent:

- #: A consecutive positive integer, used to count all described items of contribution
- **Item id:** An index (e.g., [C1.5](#)) of the contribution item, with a numbering subordinate to index (e.g., [he](#) table where it belongs).
- **Location:** A hint about the location (e.g., section number) of the edits, either in the old or in the new document.
- **Contribution topic Cx : *short title*:** An identifier Cx (with integer x) of the contribution description, and a title of the issue / contributions.
- **Related:** Related references, such as references (GIx) to GitHub issues, and/or ids of other contribution items.
- **Changes made:** Contextual information about the proposed contribution, as well as a high level description of the changes in the document.
- **Edit id:** Index (or possibly several indices) of the edits (Ey , with integer y) made in the document. Across the document, changes will be marked in the right margin with this index, so that the reader can hyperlink it directly to the description of the contribution, i.e., to an explanation of why the change was made.

List of Contributions

C1: Implement editorial structural changes	86
C2: Set expectations on intellectual property disclosure	90
C3: Add an executive summary	91
C4: Clarify proofs of knowledge	91
C5: Explain the computational security parameter	93
C6: Clarify the public vs. non-public aspect of “common” in CRS enhancement	93
C7: Discuss transferability and deniability	94
C8: Explain the statistical security parameter	95
C9: Clarify the (implicit) scope of some use-cases	96
C10: Compare circuits vs. R1CS	97
C11: Add introduction to interactive zero-knowledge proofs	98
C12: Improve description of applications and predicates	98
C13: Improve motivation in the application chapter	99
C14: Improve the table of gadgets	100
C15: Include references in Application chapter	100

Structural changes by the editors

#	Item id	Location	Contribution topic C1: Implement editorial structural changes	Related	Incorporated changes	Edit id
1	C1.1	All document	<ul style="list-style-type: none"> – Context: Inherently related to the editorial development of the reference document. – Proposed contribution: Implement editorial structural changes to the document (e.g., new chapters, sections, subsections, etc.) , as useful based on the overall set of contributions. 	GI16 , C11.1 , C7.1 , C7.2	<ul style="list-style-type: none"> – Contributors: The editors (Daniel Benarroch, Luís Brandão, Eran Tromer) – Changed: See items below. 	

#	Item id	Location	Contribution topic C1: Implement editorial structural changes	Related	Incorporated changes	Edit id
2	C1.2	Cover		GI16	– Changed: Update the version number; update the version date; add a link to find the latest version; add a ZKProof logo.	E1, E2
3	C1.3	Front matter		GI16	– Changed: Add a note “ about this version ” clarifying the context of the current version; add a proposed citation format for this version.	E4
4	C1.4	Front matter		GI16	– Changed: Add a proposed citation format for this version.	E5
5	C1.5	Front matter		GI16	– Changed: Add an abstract and a list of keywords	E3
6	C1.6	Front matter	– Context: A significant portion of the incorporated text is based on the “Towards a reference document” section of the “NIST comments on the initial ZKProof documentation” (April 06, 2019).	GI16	– Changed: In the preamble of the document, add a section “ About this community reference ” providing context about the intended development process of the document.	E6
7	C1.7	Front matter		GI16	– Changed: Improve the placement and context of the ZKProof Charter within the document: <ul style="list-style-type: none"> • Move the original “ZKProof Charter” to before the Table of Contents, and frame it within a box (E7). • Correct typo: “standardardization” → “standardization” (E8). 	E7, E8
8	C1.8	Front matter		GI16, C2	– Changed: Add editorial footnote explaining that the scope of the creative commons license is widened to incorporate the community reference (E9).	E9

#	Item id	Location	Contribution topic C1: Implement editorial structural changes	Related	Incorporated changes	Edit id
9	C1.9	Front matter		GI16, C2	– Changed: Remove the ZKProof Code of Conducts (since it is tailored to events, rather than to documents).	
10	C1.10	Front matter		GI16, E13	– Changed: Increase the depth of the table of contents to also show subsections	
11	C1.11	New Chapter 2		GI16, GI17, C11	– Changed: Create structure to fit a new chapter “2. Construction paradigms” to contain explanations of different protocol paradigms for zero-knowledge proofs.	E37
12	C1.12	Old chapter 1; new Section 2.1		GI16	– Changed: Move the old section 1.8 (“taxonomy of constructions”) to be the first section in the new paradigms chapter.	E38
13	C1.13	Section 2.3		GI16, GI17	– Changed: List several possible ZKP protocol paradigms, each of which may later become its own section with a detailed explanation of the paradigm.	E43
14	C1.14	Old section 3.2 (“Notation and terminology”) in chapter “Applications”		C12.4, C9.2	– Note: The section “Notation and terminology” was only focused on distinguishing three types of verifiability requirements. – Changed: Change the section title to “Types of verifiability”, added a header label for each enumerated type, along with minor editorial adjustments. Move some newly proposed definitions of gadget and predicate (see C12.4) to the previous introductory section (Section 4.1). Edit proposed content about the scope of use-cases related to the designated verifier case (see C9.2).	E49, E50, E54

#	Item id	Location	Contribution topic C1: Implement editorial structural changes	Related	Incorporated changes	Edit id
15	C1.15	Section 4.4		C12.4	<ul style="list-style-type: none"> – Note: Tables of individual gadgets were in pages with landscape orientation. – Changed: Remove unused column “API” and adapt column lengths for better fit in pages with portrait orientation. A text text edits inside the cells. 	E61
16	C1.16	Section 4.5.4			<ul style="list-style-type: none"> – Note: In the old single table of functionalities, across three landscape pages, the first column “Functionality/problem” spanned a large vertical space, with a short label. – Changed: Converted each row defined by a “functionality/problem” into its own table, thus reducing the horizontal width and allowing a better fit in portrait mode. 	E66
17	C1.17	End of each old chapter	Consolidate the list of used references		<ul style="list-style-type: none"> – Changed: Remove the redundant lists of references that were remaining in the end of each chapter. A few of the listed references were not cited elsewhere and where thus placed were suitable. (The list of all references is now consolidated in a single “References” section.) 	E39, E47, E68
18	C1.18	Old chapter 4 ZCon0		GI16	<ul style="list-style-type: none"> – Changed: Remove the ZCon0 notes (old chapter 4). – Note: The current editorial process separates the workshop notes from the community reference. 	
19	C1.19	All document		GI16	<ul style="list-style-type: none"> – Changed: Remove all popup pdf-annotations (done by simply clearing the definition of the calling LaTeX command <code>\pdfcomment</code> — the comments remain in the LaTeX code for future address). 	

#	Item id	Location	Contribution topic C1: Implement editorial structural changes	Related	Incorporated changes	Edit id
20	C1.20	Before the references		GI16	– Changed: Add an acknowledgments section consistent with the contributions provided to the document.	E69
21	C1.21	Front matter		GI16	– Changed: Add a Version history section, with a summarized description of the sequence of main versions of the document.	E70

Proposed changes in content

#	Item id	Location	Contribution topic C2: Set expectations on intellectual property disclosure	Related	Incorporated changes	Edit id
22	C2.1	Preamble	<p>– Context: Proposed in item C22 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019).</p> <p>– Proposed contribution: Present (in one or two paragraphs), in a non-legalese way, several remarks about intellectual property (IP). A main goal is to raise awareness about the role that IP may take or might not take in the adoption of recommendations and requirements in the community reference document. We are aware this is a delicate topic, so a goal of the contribution is to also motivate future constructive discussion/consideration by the ZKProof community, e.g., about open-source, IP rights, reasonable and non-discriminatory IP terms, etc.</p>	GI5	<p>– Contributors: NIST-PEC team.</p> <p>– Changed: Added a new section entitled “Expectations on disclosure and licensing of intellectual property”</p>	E10

#	Item id	Location	Contribution topic C2: Set expectations on intellectual property disclosure	Related	Incorporated changes	Edit id
23	C2.2	Preamble	– Proposed contribution: After requesting feedback to the Steering committee, Hugo proposed that the disclosure of patent claims applies to both “your own or held by others.”	GI5	– Contributors: Suggested by Hugo Krawczyk. – Changed: (Editors:) Added the parenthetical note “(their own and those from others)”	E11
24	C2.3	Preamble	– Proposed contribution: As part of requesting feedback to the Steering committee, Hugo proposed clarifying that the disclosure of patent claims should include both “your own or held by others.”	GI5	– Contributors: Editors team. – Changed: Add to the proposed intellectual property text a note about the expected creative commons licensing for published documents.	E12

#	Item id	Location	Contribution topic C3: Add an executive summary	Related	Incorporated changes	Edit id
25	C3.1	Preamble of the document, before the table of contents	– Context: items C5, D1-D5 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019) – Proposed contribution: Include an “executive summary” describing at a high level the structure and content of the overall “ZKProof community reference” document; the new text may also allude to the purpose, aim, scope and format of the document.	GI1	– Contributors: NIST-PEC team – Changed: Added an executive summary	E14

#	Item id	Location	Contribution topic C4: Clarify proofs of knowledge	Related	Incorporated changes	Edit id
26	C4.1	Sections 1.1 and 1.5.3	<ul style="list-style-type: none"> – Context: item c7 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019) – Proposed contribution: Make a clearer distinction of ZK proofs of membership vs. ZK proofs of knowledge, including by means of examples and definitions; clarify how the formalism can adequately model proofs of knowledge; may also include a definition of “extractability” property/game. 	GI2	<ul style="list-style-type: none"> – Contributors: NIST-PEC team – Note: See several separate items below 	
27	C4.2	Sections 1.1			Introduce acronym ZKP	E15
28	C4.3	Sections 1.1			Clarify the meaning of “secrecy” of the “information” held by the prover.	E16
29	C4.4	Sections 1.1			Enumerate the basic examples, including two new ones (chess and theorem)	E17
30	C4.5	Sections 1.1			Allude to the need of an <i>instance</i>	E18
31	C4.6	Sections 1.1			Mention proof vs. argument	E19
32	C4.7	Sections 1.2			Enhance the table of basic examples	E22
33	C4.8	Sections 1.3			Distinguish types of statement: of knowledge vs. of membership	E23
34	C4.9	(New) Sections 1.4			Distinguish types of proof: of knowledge vs. of membership	E27
35	C4.10	(New) Section 1.4.1			Add example of ZKPoK of DL	E28
36	C4.11	(New) Section 1.4.2			Add example of ZKPoK of hash pre-image	E29

#	Item id	Location	Contribution topic C4: Clarify proofs of knowledge	Related	Incorporated changes	Edit id
37	C4.12	(New) Section 1.4.3			Add example of ZKP of graph non-isomorphism	E30
38	C4.13	Section 1.6.3			Add suggestion to define ZKPoK game	E31

#	Item id	Location	Contribution topic C5: Explain the computational security parameter	Related	Incorporated changes	Edit id
39	C5.1	Chapter 2 (“Implementation”), mostly in Section 2.5.	<ul style="list-style-type: none"> – Context: Proposed in the item 18 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019). – Proposed contribution: Add text about possible computational security parameters, and the different security properties they may apply to (e.g., soundness, ZK, short-term vs. long-term, ...). In section 2.5, replace occurrences of “120” by “128”. 	GI3	<ul style="list-style-type: none"> – Contributors: NIST-PEC team – Changed: See items below. 	E44
40	C5.2	Section 1.5			Wrt to required (approximate) level of security, change 120 to 128	E45 , E46
41	C5.3	Section 1.7.1			In benchmarks, characterize different security properties	E33
42	C5.4	Section 1.7.2			Computational security levels for benchmarks	E34 , E35

#	Item id	Location	Contribution topic C6: Clarify the public vs. non-public aspect of “common” in CRS enhancement	Related	Incorporated changes	Edit id
43	C6.1	Mostly in Chapter 1, starting in section 1.2; will also check for other applicable cases across the document.	<ul style="list-style-type: none"> – Context: proposed in the “NIST comments on the initial ZKProof documentation” (April 06, 2019) — item C11. – Proposed contribution: Clarify the distinction between common (as in shared between prover and verifier) and public knowledge (as in known externally). The lack of distinction was noticed in several parts of the document, when thinking of a comparison between transferable vs. non-transferable ZK proofs. CRS is sometimes being defined as public, although in practice it could be obtained as common to the intervening parties, yet private to a particular interaction. For example, line 177 says “common public input” when first talking of a “common reference string”, but the “public” aspect is arguable – being public vs. common-but-not-public may make the difference between transferability vs. non-transferability. 	GI4	<ul style="list-style-type: none"> – Contributors: NIST-PEC team – Changed: In Section 1.2, Syntax of setup — common and private components 	E21

#	Item id	Location	Contribution topic C7: Discuss transferability and deniability	Related	Incorporated changes	Edit id
44	C7.1	Section 1.6.6 Section 2.2.3	<p>– Context: Proposed in item C9 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019).</p> <p>– Proposed contribution: Elaborate more on the concept of transferability. For example, in an interactive protocol over the Internet, how do regular authenticated channels vs. “ideally” authenticated channels affect transferability? Would a non-transferable protocol become transferable when the prover signs all sent messages and the verifier uses the output of a cryptographic hash function to select random challenges?</p>	GI6, C7.2	<p>– Contributors: Luís Brandão</p> <p>– Changed: Add subsection 1.6.6 with introductory distinction between transferability and deniability. Add paragraphs in Section 2.2.3 with nuances on transferability vs. interactivity. Remove sentence (E55).</p>	E32, E41
45	C7.2	Section 2.2.3	<p>– Context: The “deniability” item was identified in the breakout session on “Interactive Zero Knowledge” in the 2nd ZKProof workshop.</p> <p>– Proposed contribution: Elaborate more on the concept of deniability.</p>	GI6, C7.1	<p>– Contributors: Ivan Visconti</p> <p>– Changed: Add several paragraphs about off-line / on-line non-transferability, designated verifier, and transferable proofs</p>	E42
46	C7.3	Old Section 3.2	<p>– Context: Proposed in the item C14 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019).</p> <p>– Proposed contribution: In Section 3.2, revise the incorrect assertion in item 1: “Only non-interactive ZK (NIZK) can actually hold this property” [being publicly verifiable / transferable?]. For example, if transferability is a design goal then there are settings where it is possible to design interactive protocols for which the view (transcript) of the original verifier (interacting with the original prover) can later serve as a transferable proof for other verifiers.</p>	GI6	<p>– Contributors: Luís Brandão,</p> <p>– Changed:</p>	E55

#	Item id	Location	Contribution topic C8: Explain the statistical security parameter	Related	Incorporated changes	Edit id
47	C8.1	Old sections 1.2, 1.4.3 and 2.5	<ul style="list-style-type: none"> – Context: proposed in item C19 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019). Also discussed in the breakout session on ”Interactive Zero Knowledge”. – Proposed contribution: Discuss various examples of acceptable values of statistical security parameter, e.g., 40 bits. Explore how interactive to non-interactive transformations may affect the requirements on the statistical security parameter, e.g., making it become a computational parameter when applying Fiat-Shamir. 	GI10	<ul style="list-style-type: none"> – Contributors: Luís Brandão. – Changed: Add paragraphs in new subsection 1.8.3, proposing statistical security parameters for benchmarking. 	E36

#	Item id	Location	Contribution topic C9: Clarify the (implicit) scope of some use-cases	Related	Incorporated changes	Edit id
48	C9.1	Section 4.2	<ul style="list-style-type: none"> – Context: Proposed in item C15 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019). – Proposed contribution: The last paragraph in Section 3.2 [old section number in version 0.1] says “digital money based applications belong to the first model” [public verifiable as a requirement]. This assertion appears implicitly scoped in a too narrow subset of conceivable applications about digital money. Conversely, one could consider a scenario where Alice wants to convince Bob, in a non-transferable way, that Alice bought something from Charlie. Consider clarifying better the scope of examples vs. the scope of areas of application. 	GI12, C7.3	<ul style="list-style-type: none"> – Contributors: Editors – Changed: Edit some text after the enumeration of verifiability types, setting some relation to application use-cases, including revising the submitted content of item C9.2. 	E57

#	Item id	Location	Contribution topic C9: Clarify the (implicit) scope of some use-cases	Related	Incorporated changes	Edit id
49	C9.2	Section 4.2		GI12	<ul style="list-style-type: none"> – Contributors: Yu Hang to editors – Submission mode: Email to editors – Changed: Provided some content, based on [JSI96], about use-cases of designated-verifier use-cases. Substantially edited by the editors, including to remove parts redundant with the new content in Section 2.2. 	E57

#	Item id	Location	Contribution topic C10: Compare circuits vs. R1CS	Related	Incorporated changes	Edit id
50	C10.1	Section 1.3.2	<ul style="list-style-type: none"> – Context: Proposed in item C10 of the “NIST comments on the initial ZKProof documentation” (April 06, 2019). – Proposed contribution: The “security/theory” track is mentioning Boolean circuits but not R1CS. The “implementation” track is focused on R1CS without explaining why/when it is preferable to a circuit representation. Consider explaining better (in the “security” track) what is R1CS. Consider introducing and exemplifying a circuit-to-R1CS translation and/or vice-versa. Consider clarifying better in the “implementation” track why the focus is on R1CS, for example compared with circuits. 	GI13	<ul style="list-style-type: none"> – Contributors: Yu Hang – Submission mode: Email – Changed: Add new introductory content about R1CS. (Modified with revisions by the editors.) 	E25

#	Item id	Location	Contribution topic C10: Compare circuits vs. R1CS	Related	Incorporated changes	Edit id
51	C10.2	Section 1.3		GI13, GI16	<ul style="list-style-type: none"> – Contributors: Editors – Changed: Split the content of Section 1.3 across subsections, for better indexing, as follows: <ul style="list-style-type: none"> • New subsection 1.3.1 for the existing content about circuits. • New subsection 1.3.2 for the new contributed introductory content on R1CS representation. • New subsection 1.3.3 for the existing content about types of statements. 	E24, E25, E26

#	Item id	Location	Contribution topic C11: Add introduction to interactive zero-knowledge proofs	Related	Incorporated changes	Edit id
52	C11.1	Security section	<ul style="list-style-type: none"> – Context: Discussed during the "Interactive Zero Knowledge" breakout session in the 2nd ZKProof Workshop – Proposed contribution: An introduction to advantages and disadvantages of interactive zero-knowledge proofs relative to non-interactive ones, and a discussion of scenarios and applications where interactive protocols may be particularly suitable or relevant. 	GI18, C1.11	<ul style="list-style-type: none"> – Contributors: Justin Thaler, Riad Wahby, Yupeng Zhang – Submission mode: Email to editors – Changed: New entire Section 2.2 on Interactivity. 	E40

#	Item id	Location	Contribution topic C12: Improve description of applications and predicates	Related	Incorporated changes	Edit id
53	C12.1	Chapter (applications)	<ul style="list-style-type: none"> – Context: Discussed during the breakout session about the ZKProof Community Reference document – Proposed contribution: Improve the accessibility of the Applications section to meet or exceed that of Security and Implementation. This includes the following: formally expand on the existing applications for correctness and ensure that the notion of “predicates” is well understood. 	GI20	<ul style="list-style-type: none"> – Contributors: Angela Robinson and Daniel Benarroch – Submission mode: Email to editors – Changed: See items below 	
54	C12.2	Section 4.1			– Changed: Review introductory paragraphs of the applications chapter	E48
55	C12.3	Section 4.1			– Changed: Remove the “What this document is NOT about” items	E53
56	C12.4	Section 4.1		C1.14	– Changed: Define terms “predicate” and “gadgets”	E49
57	C12.5	Section 4.3			– Changed: Add references on anonymous credentials and zerocash	E58
58	C12.6	Section 4.4			– Changed: Add text as preamble to the section on “Gadgets within predicates”	E59
59	C12.7	Section 4.4			– Changed: Move a paragraph that sets the focus on “accredited investors” from Section 4.5.1 to Section 4.5.5	E67

#	Item id	Location	Contribution topic C13: Improve motivation in the application chapter	Related	Incorporated changes	Edit id
60	C13.1	Old section 3.1	<ul style="list-style-type: none"> – Context: Breakout session: ZKProof Community Reference – Proposed contribution: Motivation for ZKPs must be improved in order to allow users to understand how ZKPs can be used to solve practical problems. In particular: Include some missing items as for example recursive composition and proof-carrying-data. 	GI22	<ul style="list-style-type: none"> – Contributors: Eduardo Morais – Submission mode: GitHub pull request – Changed: Included a paragraph to explain motivation for Proof Carrying Data (PCD). 	E51

#	Item id	Location	Contribution topic C14: Improve the table of gadgets	Related	Incorporated changes	Edit id
61	C14.1	Old section 3.4	<ul style="list-style-type: none"> – Context: Breakout session: ZKProof Community Reference – Proposed contribution: Different gadgets were mentioned during the workshops. Some are already described in the document, but it is necessary to review and complete this tables. 	GI23	<ul style="list-style-type: none"> – Contributors: Eduardo Morais – Submission mode: GitHub pull request – Changed: Updated the gadgets table by filling in missing elements and making a few corrections. Also updated the specific tables for the following gadgets: signature, encryption, Distributed-decryption and set membership. 	E60, E62, E63, E64, E65

#	Item id	Location	Contribution topic C15: Include references in Application chapter	Related	Incorporated changes	Edit id
62	C15.1	References	<ul style="list-style-type: none"> – Context: Breakout session: ZKProof Community Reference – Proposed contribution: Some important references are missing. It is necessary to reference papers whenever relevant. See comments in version 0.1. 	GI24, C12.5	<ul style="list-style-type: none"> – Contributors: Eduardo Morais – Submission mode: GitHub pull request – Changed: Added 3 references to the new paragraph (E51) in the introduction of the “Applications” chapter. 	E52