NIST-PEC contributions to advance the draft ZKProof Community Reference from version 0.1 to 0.2

Luís T. A. N. Brandão, René Peralta, Angela Robinson

National Institute of Standards and Technology

September 10, 2019, Gaithersburg USA

The Privacy-Enhancing Cryptography (PEC) team at the National Institute of Standards and Technology (NIST) is interested in the development of reference material to aid in the advancement of cryptographic technology that can be used to enhance privacy in a secure, practical, and interoperable way. In this scope, we are collaborating with the development of the ZKProof Community Reference, which intends to promote zero-knowledge proofs technology in an open and inclusive manner. Our collaboration follows the editorial process initiated in the 2nd ZKProof Workshop (April 10–12, 2019).

In the present document, we propose contributions to the process of advancing the draft version 0.1 of the ZKProof Community Reference to a new draft version 0.2. Our contributions are organized into seven topics, as indexed in the table below. Each topic relates to a comment included in the "NIST comments on the initial ZKProof documentation" (April 06, 2019) and further detailed as an "issue" in the Github repository of ZKProof.

Topic	NIST early comment #	Github issue #	In this document	
			Section	Comments
Intellectual property	C22	Issue #5	1	D1.1
Executive summary	C5	Issue #1	2	D2.1
Proofs of Knowledge	C7	Issue #2	3	D3.1–D3.12
CRS public or not	C11	Issue #4	4	D4.1–D4.2
Computational security	C18	Issue #3	5	D5.1–D5.7
Statistical security	C19	Issue #10	6	D6.1
Transferability	C9	Issue #6	7	D7.1–D7.3

We offer these contributions as part of an ongoing editorial process. The proposed items cover a limited number of components in a draft that is expected to be subjected to more revision stages. Our participation should not be construed as endorsement of standardization of any content.

1. Proposal about Intellectual Property

1.1. Context

In community efforts that promote the development of new technologies, the subject of intellectual property involves a diverse set of perspectives from different stakeholders. This topic deserves explicit consideration and setting of community expectations. For that reason we have proposed that some related guidance be included in the ZKProof Reference document.

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented the following: *"C22. [...] Consider discussing possible guidance regarding intellectual property."*

After the 2nd ZKProof workshop, we detailed further the proposed contribution in the GitHub ZKProof Reference repository, as Issue #5 ("Mention intellectual property"): *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.*

Section 1.2 shows the actual text proposed for inclusion in the Community Reference. The text tries, without an overly legalese format, to convey clear expectations related to licensing terms and disclosure of patent claims. We shared in advance with the ZKProof Editors and Steering Committee an earlier version of the proposal, for feedback. We propose that the current text, to be integrated in the new draft Community Reference, also be presented to the community during the 3rd ZKProof Workshop.

1.2. Proposed changes

D1.1. Add IP guidance

In the preamble, before the executive summary, add the following unnumbered section:

Expectations on disclosure and licensing of intellectual property

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 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. 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.

The ZKProof documentation will be updated based on received disclosures about pertinent patent claims. Please email information to editors@zkproof.org.

2. Proposal about Executive Summary

2.1. Context

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented the following: *"C5. [...] Consider adding an executive summary, describing at a high level the structure and content of the overall reference documentation. [...] "*

After the 2nd ZKProof workshop, we further detailed the proposed contribution in the GitHub ZKProof Reference repository, as Issue #1 ("Add an executive summary"): *"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."*

Some of the mentioned elements — purpose, aim, scope and format — can be left to an editorial note different from the executive summary. Section 2.2 shows the actual text proposed as an initial executive summary for inclusion in the Community Reference.

2.2. Proposed changes

D2.1. Add an executive summary

In the preamble of the document, add the following as a new unnumbered section:

Executive Summary

Zero-knowledge proofs (ZKPs) are an important privacy-enhancing tool from cryptography. They preserve confidentiality of data, while proving the veracity of statements about the data. They can also be used to prove knowledge of such data without having to disclose it. ZKPs can have a positive impact in industries, agencies, and for personal use, by allowing privacy-preserving applications where designated private data can be made useful to third parties, despite not being disclosed to them. The development of this "ZKProof Community Reference" is a step towards enabling wider adoption of ZKP technology, possibly preceding the establishment of future standards.

ZKPs were developed by the academic community in the 1980s, and have seen tremendous improvements since. They are now of practical feasibility in multiple domains of interest to the industry, and to a large community of developers and researchers. This document aims to be a community-built reference for understanding and aiding the development of ZKP systems in a secure, practical and interoperable manner. It is not a substitution for research papers, technical books, or standards. It is intended to serve as a reference handbook of introductory concepts, basic techniques, implementation suggestions and application use-cases. This aims to serve the broader community, particularly those interested in understanding ZKP systems, making an impact in their advancement, and using related products.

ZKP systems involve at least two parties: a prover and a verifier. The goal of the prover is to convince the verifier that a statement is true, without revealing any additional information. For example, suppose the prover (let us call her Peggy) holds a birth certificate digitally signed by an authority. She wishes to convince a verifier that she is at least 18, in order to access some service. ZKP techniques allow her to do this without revealing her birthdate.

This document describes important aspects of the current state of the art in ZKP security, implementation, and applications. There are several use-cases and applications where ZKPs can add value. However, this requires benchmarking implementations under several metrics, evaluating tradeoffs between security and efficiency, and developing a basis for interoperability. The security of a proof system is paramount for the system users, but system efficiency is also essential for user experience.

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

Completeness requires that if the statement is true, then at the end of the interaction the prover is convinced of this fact. Soundness requires that not even a malicious prover can convince the verifier of a false statement. Zero knowledge requires that even a malicious verifier cannot extract any information beyond the truthfulness of the given statement.

The "Implementation" chapter focuses on devising a framework for the implementation of ZKPs, which is important for interoperability. One important aspect to consider upfront is the representation of statements. In a ZKP protocol, the statement needs to be converted into a mathematical object. For example, in the case of proving one's age is at least 18, the statement is equivalent to proving that one's private birthdate Y_1-M_1-D_1 (year-month-day) satisfies a relation with today's date Y_2-M_2-D_2, namely that their distance is greater than or equal to 18 years. This simple example 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 $Y_2=Y_1+18 \wedge M_2=M_1 \wedge D_2 \geq D_1$. An actual conversion suitable for ZKPs, namely for more complex statements, can pose an implementation challenge. There are nonetheless various techniques that enable converting a statement into a mathematical object, such as a circuit. This document gives special attention to the R1CS/QAP-style constraint system representation, which is adopted by several ZKP solutions in use today. Also, the document gives special emphasis to implementation of non-interactive proof systems.

The privacy enhancement offered by ZKPs can be applied to a wide range of applications. The "Applications" chapter presents two use-cases that can benefit from ZKP systems: identity framework and asset transfer. In a privacy-preserving identity framework, one can for example prove useful personal attributes, such as age and state of residency, without revealing more detailed personal data such as birthdate and address. In an asset-transfer setting, financial institutions that facilitate transactions usually require knowing the identities of the sender and receiver, and the asset type and amount. ZKP systems enable a privacy-preserving variant where the transaction is performed between anonymous parties, while at the same time ensuring they and their assets satisfy regulatory requirements. These use cases, as well as a wide range of many other conceivable privacy-preserving applications, can be enabled by a common set of tools, or gadgets, for example including commitments, signatures, encryption and circuits.

The interplay between security concepts and implementation guidelines must be carefully balanced to ensure the development of secure, practical, and interoperable ZKP applications. Solutions provided by ZKP technology must be ensured by careful security practices and realistic privacy guarantees. This document aims to summarize security properties and implementation techniques which help achieve these goals.

3. Proposal about Proofs of Knowledge

3.1. Context

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented: *"C7. [...] Consider making a clearer distinction of ZK proofs of membership vs. ZK proofs of knowledge, including by means of examples and definitions. Consider clarifying how the formalism can adequately model proofs of knowledge. A definition of an "extractability" property/game may be useful."*

After the 2nd ZKProof workshop, we detailed further the proposed contribution in the GitHub

ZKProof Reference repository, Issue #2 ("Explain the computational security parameter"): *"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."*

Section 3.2 shows the proposed changes to the Community Reference.

3.2. Proposed changes

The following proposed changes relate to old sections 1.1, 1.3, and 1.5.3, and propose a new subsection 1.4.

D3.1. Introduce acronym ZKP

In Section 1.1.1, line 131, introduce the ZKP acronym, when first writing the extended form. Where it says "A zero-knowledge proof makes it possible [...]", write:

> A zero-knowledge proof (ZKP) makes it possible [...]

D3.2. Clarify secrecy of witness

In Section 1.1.1, line 132, introduce sentences clarifying the meaning of secret information. Within the text "[...] secret information. There are numerous uses of [...]", write:

> [...] secret information. 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 [...]

D3.3. List examples at a high level

In section 1.1.1, lines 132–133 (before Table 1.1), the draft 0.1 contained Table 1.1 with examples of ZKP scenarios. Further below, comment D3.6 proposes enhancing the table, to convey intuition about additional elements in a ZKP. However, to retain a simple beginning of Section 1, before the table gets more complex, we can mention first in running text (in an enumerated environment) a simple description of the scenarios (including two new proposed ones).

Proposed change — where it says "There are numerous uses of zero-knowledge proofs. Table 1.1 gives three examples where proving claims about confidential data can be useful.", change the text to:

> 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 moves;
5. correctness (demonstrability) of a theorem, without revealing its mathematical proof.

D3.4. Allude to the need of an instance

In section 1.1.1, after the enumeration of scenarios proposed in the previous item, add a paragraph alluding to the need of a supporting instance (a substrate) and to the qualification of *statement of knowledge*. Proposed text:

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 handling the corresponding decryption key and plaintext (the witness) as a secret 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.

D3.5. Mention proof vs. argument

In the end of section 1.1.1, after old line 147, add a paragraph conveying at a high level the distinction between "proof" and "argument" and stating that in this document the terminology is simplified to simply use "proof":

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

D3.6. Enhance table of examples

From section 1.1.1, move and change Table 1.1 (old lines 134–137) to the end of section 1.2. Transpose the table to enable space for more examples, and add a column to describe the "instance" (commonly known by prover and verifier) for each example scenario, in order to enable intuition about the substrate (the instance) that supports the statement with respect to the confidential info. In the table, add an example with the chessboard configuration problem (mentioned in old section 3.4 "Gadgets within predicates"), as well as a similar but more formal example on theorem validity. — these will later be useful to compare the perspectives of ZKP of knowledge vs. ZKP of membership. As a way to let section 1.1 remain simple, we propose that the table, which became more complex, be moved to the end of old section 1.2 ("Terminology"), along with a prefix sentence mentioning it. Here is the proposed result for the end of section 1.2:

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

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	Identification smartcard chip	Birthdate and personal data (signed by a CA)
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	(The logical rules of inference)	A sequence of logical implications

Legend: CA = certification authority

D3.7. Distinguish types of statements: knowledge vs. membership

In section 1.3, old line 183, change "A statement is a claim $x \in L$, which can be true or false" to:

A *statement* is either a *membership* claim of the form "$x \in L$", or a *knowledge* claim of the form "In the scope of R, I know a witness for x." In an informal setting, the *knowledge* and *membership* types of statement can in some cases be considered interchangeable, but formally there are technical reasons to distinguish between the two notions. In particular, there are scenarios where statements of knowledge cannot be converted into statements of membership, and vice-versa (as exemplified in Section 1.4). The notion of statement of knowledge is the preponderant one in this document.

See related contribution items D3.8–D3.11.

D3.8. Distinguish types of ZKP: knowledge vs. membership

After Section 1.3, old line 221, add a new Section (1.4) conveying the distinction between ZKP of knowledge and ZKP of membership, as follows:

1.4 ZKPs of knowledge vs. ZKPs of membership. The theory of ZKPs distinguishes between two types of proofs, based on the type of statement (and also on the type of security properties — see Sections 1.6.2 and 1.6.3):

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

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

The “chessboard configuration” and the “theorem validity” scenarios in Table 1.1 are interesting in the sense of their instances not containing any cryptographic support (e.g., encryption, signature or commitment). Each of those statements can be seen as a claim of membership, in the sense of claiming that the expression belongs respectively to the language of theorems (i.e., of provable expressions) or the language of valid chessboard configurations (i.e., reachable by a sequence of moves). At the same time, a further specification of the statement can be expressed as a claim of knowledge of a sequence of legal moves or a sequence of logical implications.

D3.9. Example: ZKPoK of DL

After the text from contribution D3.8 (creating a new Section 1.4), add as a new subsection (1.4.1) a concrete example of a ZKP of knowledge (of discrete log) that does not have a dual ZKP of membership:

1.4.1 Example: ZKP of knowledge of discrete-log. Let p be a large prime (e.g., with 4096 bits) of the form $p = 2q + 1$, where q is also a prime. Let g be a generator of the group $\mathbb{Z}_p^* = \{1, ..., p-1\} = \left\{g^i : i = 1, ..., p-1\right\}$ under multiplication modulo p. Assume that it is computationally infeasible to compute discrete-logarithms in this group. Let w be a secret element (the witness) known by the prover, and let $x = g^w$ be the instance known by both the prover and verifier, corresponding to the following statement by the prover: “I know the discrete-log (base g) of the instance (x)” (in other words: “I know a secret exponent that raises the generator (g) into the instance (x)”). Consider now the relation $R = \{(x, w) : g^w = x \pmod p\}$. In this case, any empty string is a proof of membership for the corresponding language $L = \{x : \exists w : (x, w) \in R\}$, since this language is trivially equal to $\mathbb{Z}_p^*$, for which membership is self-evident (without any knowledge of w). However, whether or not the prover knows a witness is a non-trivial matter since (the current publicly-known state of the art does not provide a way to compute discrete logarithms in time polynomial in the size of the prime modulus, except if with a quantum computer). In summary, this is a case where a ZKPoK makes sense but a ZKP of membership does not.

D3.10. Example: ZKPoK of hash pre-image

After the example of ZKPoK from contribution D3.9, add as a new subsection (1.4.2) an example of a ZKP of knowledge (of a hash pre-image) with a different subtlety about the duality between knowledge vs. of membership:

1.4.2 Example: ZKP of knowledge of a hash pre-image. Consider a cryptographic hash function $H : \{0,1\}^{512} \rightarrow \{0,1\}^{256}$, restricted to binary inputs of length 512. For many (possibly all) 256-bit instances in the co-domain $\{0,1\}^{256}$ of H there are many pre-images in $\{0,1\}^{256}$. Let w be a witness (hash pre-image) know by the prover (and unpredictable to the verifier), for some instance $x = H(w)$ known by the verifier. Since a cryptographic hash function is one-way, it makes sense to give a ZKPoK of a pre-image, corresponding to proving knowledge of a witness in the relation $R = \{(x, w) : H(w) = x\}$. Such proof also constitutes directly a proof of membership, i.e., that the instance x does in fact belong to the co-domain of H. However, interestingly, membership in the co-domain of H is a problem that might or might not make sense depending on the known properties of the hash function H. If H is known to have as a co-domain the set of all bit-strings of length 256, then membership is self-evident. Otherwise it may be that an element x uniformly selected from the range $\{0,1\}^{256}$ is in fact not in the co-domain of H, case in which a proof of membership makes sense.

D3.11. Example: ZKPoK of graph non-isomorphism

After the previous proposed item, provide as a new subsection (1.4.3) an example of ZKP of membership without a ZKPoK counterpart:

1.4.3 Example: ZKP of membership for graph non-isomorphism. In the theoretical context of provers with super-computation ability, one can conceive a proof of membership without the notion of witness, therefore voiding the notion of a dual ZKP of knowledge. A classical example uses the language of pairs of non-isomorphic graphs, i.e., where the proof is about convincing a verifier that two graphs are not isomorphic. The classical example uses an interactive proof that does not follow from a witness but rather from a super-ability, by the prover, in deciding isomorphism between graphs. (The verifier challenges the prover to detect which of the two graphs is isomorphic to a random permutation of one of the two original graphs.) (This proof makes sense for families of pairs of graphs where non-isomorphism is not findable via an easy to compute characteristic of each graph.) This document is not focused on these settings that require infeasible provers (namely not polynomially bound). However, some conceivable applications may make use of similar notions, e.g., possibly in a context of proofs of work, or when actually dealing with "super-computers" or quantum computers as provers interacting with regular classical computers as verifiers.

D3.12. Suggestion to define ZKPoK game

In the end of old subsection 1.5.3 (new subsection 1.6.3), after old line 356, add the following editorial suggestion, to be carefully addressed in a future revision:

[[**To improve.** Consider adding, in a future version of this document draft, a game definition for the extractor required by the formal notion of proof of knowledge. This security property also arises naturally in the ideal/real simulation paradigm, in the context of an *ideal ZKP functionality* that, in the ideal world, receives the witness directly from the prover.]]

4. Proposal about CRS as public or not

4.1. Context

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented the following: *"C11. [...] Consider clarifying the distinction between common knowledge (between prover and verifier) and public knowledge. The lack of distinction is noticed in several parts when trying to think of a comparison between transferable vs. non-transferable cases. CRS is being defined as public, although in practice it could be obtained as common to the intervening parties, private to a particular interaction."*.

After the 2nd ZKProof workshop, we detailed further the proposed contribution in the GitHub ZKProof Reference repository, as Issue #4 ("Explain the computational security parameter"): *"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."

Section 4.2 shows the proposed changes to the Community Reference.

4.2. Proposed changes

The following proposed changes relate to locations mostly in Chapter 1, starting in section 1.2; will also check for other applicable cases across the document.

D4.1. Clarify public vs. common input

In section 1.2 "Terminology", old lines 187–188, change "Instance: Public input that is known to both prover and verifier. Sometimes scientific articles use 'instance' and 'statement' interchangeably, but we distinguish between the two." to:

> Instance: Input commonly known to both prover and verifier, and used to support the statement of what needs to be proven. This common input may either be local to the prover–verifier interaction, or public in the sense of being known by the external parties.

D4.2. Syntax of setup — common and private components

In section 1.2 "Terminology", old line 200–202, where it says "**Setup:** Input to e.g. prover and verifier. **Common reference string:** Some zero-knowledge systems require common public input, e.g., CRS = $\text{setup}_P = \text{setup}_V$." adjust as follows:

> "**Setup:** The inputs given to the prover and to the verifier, independent from the instance x and the witness w. The setup of each party can be decomposed into a private component ("PrivateSetup_P" or "PrivateSetup_V", respectively not known to the other party) and a common component "CommonSetup = CRS" (known by both parties), where CRS denotes a "common reference string" (required by some zero-knowledge proof systems). Notation: $\text{setup}_P = (\text{PrivateSetup}_P, \text{CRS})$ and $\text{setup}_V = (\text{PrivateSetup}_V, \text{CRS})$."
>
> For simplicity, some parameters of the setup are left implicit (possibly inside the CRS), such as the security parameters, and auxiliary elements defining the language and relation. See more details in section 1.5.3.
>
> Note: while the witness (w) and the instance (x) can be assumed as part of the setup of a concrete ZKP protocol execution, they are often distinguished in their own category. In other words, the term "Setup" is often used with respect to the setup of a proof system, which can then be instantiated for multiple executions with different instances (x) and witnesses (w)."

5. Proposal about Computational Security parameter

5.1. Context

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented the following: *"C18. [...] Consider providing rationale for the recommendation of 120 bits of computational security. [...] "* Note: we have meanwhile confirmed that 120 in the original documentation was a typo — the intended value was 128.

After the 2nd ZKProof workshop, we detailed further the proposed contribution in the GitHub ZKProof Reference repository, as Issue #3 ("Explain the computational security parameter"): *"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'."*

Section 5.2 shows the proposed changes to the Community Reference.

5.2. Proposed changes

The following proposed changes relate to the old section 2.5 = new section 3.5.

D5.1. Benchmark security levels

In old section 2.5.2 "How to run the benchmarks" (new section 3.5.2), old line 916: "The benchmarks should be run at approximately 120-bit security or larger." → "The benchmarks should be obtained preferably for more than one security level, following the recommendations stated in Section 3.5.4."

D5.2. SHA-256 at 128-bit level

In old section 2.5.3 "What benchmarks to run" (new section 3.5.3), old line 935: where it says "(e.g. SHA-256) [...] at 120-bit classical security." change to

> (e.g., SHA-256) [...] at 128-bit classical security.

D5.3. Primitives at 128-bit level

In old section 2.5.3 "What benchmarks to run" (new section 3.5.3), old line 959: where it says "the primitive should be given at a level of 120 bits or higher" change to

> the primitive underlying the ZKP statement should be given at a level of 128 bits or higher.

D5.4. Characterize the security properties

In old section "2.5.4 Security" (new section 3.5.4), old line 978–981, change "To aid this benchmarks should make it clear which security level (Definition see theory track document) is being used. In particular the benchmark should clearly state under which assumptions the claimed security is achieved. If the security is conjectured then benchmarks should display both the conjectured as well as the proven performance." to:

> **Characterize the security properties.** The benchmarking of a technique should clarify the distinct security levels achieved/conjectured for different security properties, e.g., soundness vs. zero-knowledge. In each case, the security type should also be clarified with respect to being unconditional, statistical or computational. When considering computational security, it should

clarified to what extent pre-computations may affect the security level, and whether/how known attacks may be parallelizable. All security claims/assertions should be qualified clearly with respect to being based on proven security reductions (and which computational assumptions and implementation requirements they depend on) vs. heuristic conjectures.

D5.5. Computational security levels

In old section "2.5.4 Security" (new section 3.5.4), lines 981-982, change "Benchmarks should be run with at least 120-bit security." to [Begin new paragraph]:

Computational security. The benchmarks for each technique SHALL include at least one parametrization for (conjectured) at least 128 bits of (approximate) computational security. Preferably, each technique SHOULD be benchmarked for computational security levels corresponding to 128, 192 and 256 bits. The evaluation at computational security below 128 bits may be justified for the purpose of clarifying how the execution complexity or time varies with the security parameter, but should not be construed as a recommendation for practical security.

D5.6. Exception for lower levels

In old section "2.5.4 Security" (new section 3.5.4), add paragraph

An exception allowing lower computational security parameter. With utmost care, a computational security level may be justified below 128 bits, including for benchmarking. Here is an exception:

In some interactive zero-knowledge protocols (see Section 2.2), there may be cryptographic properties that only need to be held during a portion of a protocol execution, which in turn may be required to take less than a fixed amount of time, say, one minute. For example, a commitment scheme used to enable temporary hiding during a coin-flipping protocol may only need to hold until the other party reveals a secret value. In such case the property may be implemented with less than 128 bits of security, under special care (namely with respect to composition in a concurrent setting) and if the difference in efficiency is substantial. Such decreased security level of a component of a protocol may actually be useful for example to enable properties of deniability (non-transferability)."

D5.7. Suggestion — reconsider text location about security levels

Editorial suggestion: Some of the text in the old section "2.5.4 Security" (new section 3.5.4) may in a future version of the draft document be better suited in "Chapter 1: Security".

6. Proposal about Statistical security

6.1. Context

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented: *"C19. [...] Consider discussing various examples of acceptable values of statistical security parameter. It can be useful to 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."*

After the 2nd ZKProof workshop, we transcribed the proposed contribution in the GitHub ZKProof Reference repository, Issue #10 ("Explain the statistical security parameter"), with a slight variation that added the example of a possibility of 40 bits of statistical security.

Section 6.2 shows the proposed changes to the Community Reference.

6.2. Proposed changes

D6.1. Statistical security levels

In old section "2.5.4 Security" (new section 3.5.4), add a paragraph with considerations about the statistical security parameter, including guidance for benchmarking:

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

While an appropriate minimal parameter may depend on the application scenario, this document reference requires here that a benchmarking be done with statistical security of at least 64 bits. Complementary, it is recommended that, whenever the efficiency variation is substantial across variations of statistical security parameter, the cases of 40, 64, 80 and 128 bits be benchmarked, along with 256 bits if considering security against quantum adversaries. For techniques where the efficiency upon using 64 bits of statistical security is similar to that of using a parameter similar to computation security (at least 128 bits), then the benchmark should simply equalize both security parameters.

The value of benchmarking interactive protocols using statistical security of the order of 128 or 256 bits is in allowing applicability of the benchmark for non-interactive transformed versions of the protocol, e.g., as usually achieved by a Fiat-Shamir transformation or similar. In the resulting non-interactive protocols, the prover is the sole generator of the proof. Therefore, a malicious prover can rewind and restart an attempt to generate a forged proof whenever a non-interactively produced challenge is unsuitable to complete the forgery. However, if the statistical security is equal to the computational parameter, the prover is not able to make enough computation attempts.

7. Proposal about transferability and deniability

7.1. Context

In the "NIST comments on the initial ZKProof documentation" (April 6, 2019) we commented: *"C9. [...] The concept of transferability could benefit from more attention. 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?"*

After the 2nd ZKProof workshop, we detailed further the proposed contribution in the GitHub

ZKProof Reference repository, as Issue #10 ("Discuss transferability and deniability"): *Elaborate more on the concept of transferability. [...content mentioned above...]*

Also, 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.

This topic was also discussed in the breakout session on "Interactive Zero Knowledge" during the 2nd ZKProof Workshop (April 10–12, 2019), and is expected to receive complementary contributions.

Section 7.2 shows the proposed changes to the Community Reference.

7.2. Proposed changes

[Note: some of the contributions described herein may be adapted to related contributions that may appear concurrently in connection with the discussion of interactivity in ZKP systems.]

D7.1. Mention deniability vs. transferability

After old section 1.5.5 (new section 1.6.5), after old line 417, add a new subsection (1.6.6) introducing the dual features of transferability and deniability:

1.6.6 Transferability vs. deniability

In the traditional notion of zero-knowledge, a ZKP system prevents the verifier from even being able to convincingly advertise having interacted with a legitimate prover. In other words, the verifier cannot transfer onto others the confidence gained about the proven statement. This property is sometimes called *deniability* or *non-transferability*, since a prover that has interacted as legitimate prover in a proof is later able to *plausibly deny* having done so, in case the original verifier tries to convince external parties that the protocol transcript was obtained upon interacting with a legitimate prover.

Despite *deniability* being often a desired property, the dual property of public verifiability can also be considered a feature, and such setting is also of interest in this document. This is often equated to *transferability*, in the sense that then the transference of a proof (transcript) to an external party lets said party be convinced that the corresponding statement is true. In the case of a statement of knowledge, this means being convinced that some prover did indeed have the claimed knowledge.

D7.2. Remove incorrect statement

In old section 3.2 (new section 4.2), lines 1325–1326, item 1 ("Publicly verifiable as a requirement), remove the statement "Only non-interactive ZK (NIZK) can actually hold this property."

D7.3. Nuances on transferability vs interactivity

Within a new section about interactivity (to be developed as section 2.2, in a new chapter 2, within the scope of another contribution — see GitHub Issue #18 "Introduction to interactive zero-knowledge proofs"), discuss the nuanced possibilities of transferability/deniability vs. interactivity:

Transferability/deniability vs. interactivity.

The relation between transferability and interactivity is not without nuances. Several possibilities:

- **Non-interactive and deniable.** A non-interactive ZKP may be non-transferable. This may be based for example on a setup assumption such as local CRS that is itself deniable. In that case, a malicious verifier cannot prove to an external party that the CRS was the one used in a real protocol execution, leading the external party to have reasonable suspicion that the verifier may have simulated the CRS in a way to become able to simulate a protocol execution transcript, without actual participation of a legitimate prover.
- **Non-interactive and transferable.** If transferability is intended as a feature, then a non-interactive protocol can be achieved for example with a public (undeniable) CRS. For example, if a CRS is generated by a trusted randomness beacon, and if soundness follows from the inability of the prover to control the CRS, then any external party (not involved with the prover at the time of proof generation) can at any later time verify that a proof transcript could have only been generated by a legitimate prover (one possessing the claimed knowledge).
- **Interactive and deniable.** A classical example for obtaining the deniability property comes from interactive ZKP protocols proven secure based on the use of rewinding, in a standalone setting (assuming no concurrent executions). Here, deniability follows from the simulatability of transcripts, for any malicious verifier. For each interactive step, the simulator learns the challenge issued by the possibly malicious verifier, and then rewinds to reselect the preceding message of the prover, in a way to be able to answering the subsequent challenge. Some techniques require the use of commitments or/and trapdoors, and may avoid enable this property even for straight-line simulation (provided there is an appropriate trusted setup).
- **Interactive and transferable.** In certain settings, it is possible from an interactive ZKP protocol execution to produce a transcript that constitutes a transferable proof. This may happen as a flaw to an intended deniable interactive ZKP, or as an intentional feature designed into the system. Usually the transferability property is achieved in a setting where the (possibly malicious) verifier can convincingly show to external parties that the challenges selected during a protocol execution were unpredictable at the time of the determination of the preceding messages of the prover in the protocol execution. The transferable proof transcript is then composed of the messages sent by the prover, and additional information from the internal state of a malicious verifier, including details about the generation of challenges. In a flawed protocol (that intended but failed to be deniable), a challenge produced (by the verifier) as a cryptographic hash output of the previous messages may later provide assurance that only a legitimate prover would have been able to generate a valid subsequent message (response); as another example, the protocol used for communication may be using a trusted service to cryptographically sign and timestamp the exchanged messages, such that the overall sequence of those certified messages then becomes, in the hands of the verifier, a transferable proof. Furthermore, for any of the above mentioned examples of a built transferable transcript, the actual transference may be recursively performed in an interactive way, by letting the verifier (in possession of the transcript), act as prover in a transferable ZKP of knowledge of a transferable transcript, thereby transferring to the external verifier a new transferable transcript.