A Subversion-Resistant SNARK

Research Seminar in Cryptography

Author: Hiie Vill
Supervisor: Karim Baghery
University of Tartu

1 Introduction

The purpose of this report is to give an overview of the paper "A Subversion-Resistant SNARK", written by Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa and Michał Zając, and published in ASIACRYPT in 2017, and to give an introduction to the topic of SNARKs. There have been several constructions for zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK). The most efficient zk-SNARK for Quadratic Arithmetic Programs was proposed by Jens Groth in 2016. However, in his construction, the CRS generator is expected to be a trusted third party. This paper explores the case when the CRS generator has been subverted and may cooperate with the verifier in an attempt to break zero-knowledge. A new construction for subversion-resistant zk-SNARKs is proposed, based on Groth's zk-SNARK [1].

2 Preliminaries and background

Before describing the protocol proposed in the paper, several terms must be defined in order to familiarize the reader with the background information and the topic of SNARKs. This section covers the necessary definitions used later on in the protocol description.

The phrase "this paper" used in the report always refers to the paper "A Subversion-Resistant SNARK".

2.1 SNARK

SNARK stands for Succinct Non-interactive Arguments of Knowledge.

2.1.1 Non-interactive

Assume that a prover P with a statement x and a witness w to verify the statement wishes to convince a verifier V that x has some property, but without revealing the witness to the verifier. An interactive protocol for this is as follows: the prover sends a statement x to V, V replies with a uniformly random challenge and the prover then responds with a proof based on the challenge to prove that x belongs to the language. A protocol that consists of only the prover sending a proof to the verifier, omitting the commitment and challenge stages, is called a non-interactive protocol. In a non-interactive protocol, the entire proof must be contained in one message and be sufficient for the verifier to accept it.

2.1.2 Succinct

Succinctness implies that the proof (in the form of a bitstring) is as short as possible. Here we assume that the proof size must be smaller than the size of witnesses.

2.1.3 Usage

One example of real-life applications for SNARKs is the crypto-currency Zcash. For several transactions, e.g. creating new coins or making purchases, the owner of the coins has to prove their ownership of the coins (statement), but without revealing their physical address (witness) and any secret information about the coins. This can efficiently be proven by using zk-SNARKs [5].

2.1.4 ZK-SNARK

ZK-SNARK, short for zero-knowledge SNARK, adds the following condition to the protocol: the verifier must learn nothing about the witness. Zero-knowledge is usually proven by showing the existence of a simulator – an algorithm that is capable of constructing proofs that are indistinguishable from the real proofs generated by the prover, but without using witnesses. Since the real and simulated proofs cannot be told apart from, it is guaranteed that no information about the witnesses is leaked and the protocol is zero-knowledge. This paper only looks at zero-knowledge SNARKs. Tha parties and flows of a zk-SNARK protocol are shown in Fig. 1.

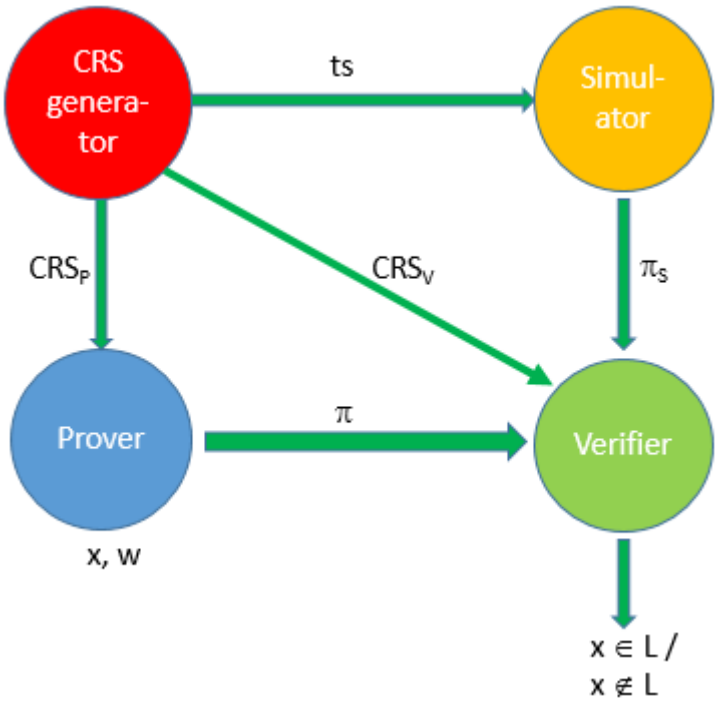

Fig. 1 ZK-SNARK

2.1.5 Arguments of knowledge

Arguments of knowledge means that the proof satisfies the knowledge soundness property, which means the prover must be able to convince the verifier that he really knows the witness. This assumes the extractability of the witness. In short, this requires the existence of an extraction function, which returns the value of the witness. This function is not run since it requires some type of secret knowledge (e.g. the source code of the CRS generator) and it would reveal the witness, which should be kept secret, but such a function must exist in order for the protocol to be knowledge-sound.

A knowledge-sound argument system is called “argument of knowledge“. (p 11)

The proof that is generated by the prover is referred to as an argument.

3 Groth's ZK-SNARK

3.1.1 CRS

Non-interactive proofs systems are usually constructed in either random oracle or common reference string models. In the case of both Groth's SNARK and the sub-ZK SNARK introduced in this paper, the CRS case was selected due to their efficiency in ZK-SNARKs over random oracle models.

A common reference string is a public string of polynomial length, generated by a CRS generator. In the case of Groth's ZK-SNARK and any other previously constructed zk-SNARKs, the CRS generator is a trusted third party. The CRS has a secret key known as a trapdoor, which is only shared with the simulator. The prover and verifier use relevant parts of the CRS to generate their proofs or to check the proofs. The simulator, knowing the trapdoor to the CRS, can generate a simulated proof using this information.

3.1.2 GBGM

GBGM, short for Generic Bilinear Group Model, is a model commonly used in pairing-based SNARKs. In the proofs, the prover and simulator compute a number of group elements and the verifier then checks them by using pairings [2].

The bilinear groups are defined as follows. Let gk = (p, G_1, G_2, G_T, e), where G_1, G_2 and G_T are three additive cyclic groups of order p (a prime number) with generators g_1, g_2 and g_T, and e is an efficient bilinear mapping between those groups: $G_1 \times G_2 \rightarrow G_T$. The operation e is called a pairing and such groups are called bilinear groups. Generic implies that only the generic group operations are used to create and manipulate the group elements [2,4].

The efficiency of e is paramount to the verifier's efficiency.

In this paper, the bracket notation is used to represent group elements: $[x]_i = g_i^x$., where $i \in \{1, 2, T\}$. The pairing function e is also denoted with a dot product operator ●. For example, the pairing between two elements from groups G_1 and G_2 can be written as $[x]_1 \bullet [y]_2 = e([x]_1, [y]_2) = e(g_1^x, g_2^y) = [xy]_T$.

3.1.3 Sub-GBGM

Sub-GBGM takes into account the fact that the CRS generator can be subverted. In order for the model to be more realistic, the model is extended with an operation enabling (the adversary) to create new group elements without knowing their discrete logarithms. The new model is called sub-GBGM.

3.1.4 Pairings

Pairings can be either symmetric or asymmetric. For some asymmetric pairings, there exists an efficient isomorphism between groups G_1 and G_2. This paper looks at asymmetric pairings of type III where there exists no efficient isomorphism between the groups. This lays the groundwork for using an assumption that an adversary cannot create $[x]_1$ without knowing $[x]_2$ or vice versa (the BDH-KE knowledge assumption is defined in 4.8) [4].

3.1.5 QAP

QAP, stading for Quadratic Arithmetic Program, is an NP relation where the prover with a statement x and a witness w builds a set of polynomial equations that are then verified by the verifier by using pairings. There exists a reduction for QAP from CIRCUIT-SAT [1].

3.1.6 Overview

In 2016, Jens Groth proposed a construction for a zk-SNARK that uses the GBGM setting. In Groth's zk-SNARKs, the CRS generator generates a CRS to be used by the prover to generate the proof and the verifier to verify. The prover creates three group elements in the GBGM setting and verifier efficiently checks them using pairings. The simulator receives a trapdoor from the CRS generator and uses this to generate a simulated proof. All of these algorithms are explained more thoroughly in section 4. At the date of this paper's publication, this ZK-SNARK construction was the most efficient one for QAP [1,2].

4 Contributions of the paper

This paper considers the case when the prover does not trust the CRS generator is not a trusted third party from the prover's perspective, but instead may have been subverted, and a new subversion-resistant version of the zk-SNARK is constructed that enables the prover to distrust the CRS generator. Since Groth's SNARK is the most efficient SNARK for QAP relation up to date, his SNARK construction was chosen as the basis.

4.1 Overview

A subverted CRS generator can either cooperate with the prover or the verifier. The CRS generator may cooperate with a dishonest prover to try to break the soundness of the protocol by sharing the CRS trapdoors with the prover, enabling the prover to create false proofs that the verifier would accept. Alternatively, the CRS generator can cooperate with the verifier to break the zero-knowledge property of the protocol by generating a CRS that would reveal any additional information from the prover other than the truth of the statements.

It has been shown that subversion-soundness and zero-knowledge in SNARKs are mutually exclusive – it is not possible to achieve both simultaneously [3]. However, since it is possible to achieve one of them, the zero-knowledge property was chosen as a basis for the subversion-resistance.

In order to protect the prover and ensure the protocol is still zero-knowledge even if the CRS generator has been subverted, the authors of this paper constructed a new subversion-resistant ZK-SNARK. The new sub-ZK SNARK was built on Groth's ZK-SNARK, extended with the following changes:

1. A new CRS verification algorithm (CV) introduced that is run by the prover before constructing his proof. The new CV algorithm verifies whether the CRS was generated correctly or not. Based on the result, the prover can either trust the CRS and generate his proof, or distrust the CRS and abort.
2. The CRS was made trapdoor extractable. Since the simulator cannot trust the CRS generator, he cannot trust that the trapdoor sent by the CRS generator is the correct one.

To counter this, the CRS was made trapdoor extractable, so that the simulator would be able to extract the trapdoor from the CRS himself.

3. These new functional changes were achieved by splitting the CRS generation algorithm into three separate stages and adding 2n + 3 additional elements to the CRS (n stands for the number of multiplication gates).

The Prover, Verifier and Simulator algorithms remain exactly the same as in Groth's ZK-SNARK and are thus not a contribution of this paper, but the algorithms are described in this section nevertheless for completeness.

4.2 Overview of the protocol

In a high-level view, the sub-ZNK SNARK consists of seven probabilistic polynomial-time algorithms: K_{tc} (CRS trapdoor generator), K_{crs} (CRS generator), K_{ts} (Simulation trapdoor generator), P (Prover), V (Verifier), S (Simulator) and CV (CRS verifier). The prover with a statement x and a witness w wishes to convince the verifier that $x \in L$. The CRS generator randomly picks a trapdoor (secret key), based on this generates a public CRS for all parties to use, and lastly generates a simulator trapdoor to be sent to the simulator. Since the prover cannot trust the CRS generator, he first runs the CRS verification algorithm that outputs 1 if the CRS was generated correctly, 0 otherwise. Using the witnesses, statements and the CRS, the prover constructs a proof and sends to the verifier. The simulator uses an extractor algorithm to extract the CRS trapdoors from the CRS. With the extracted CRS trapdoors, the simulator can generate a simulated proof and send to the verifier. The verifier, unable to distinguish between the proof and the simulated proof, verifies the proof (or the simulated proof) and outputs 1 if $x \in L$ and 0 if $x \notin L$. The parties and the flow of the protocol have been depicted on Fig. 2.

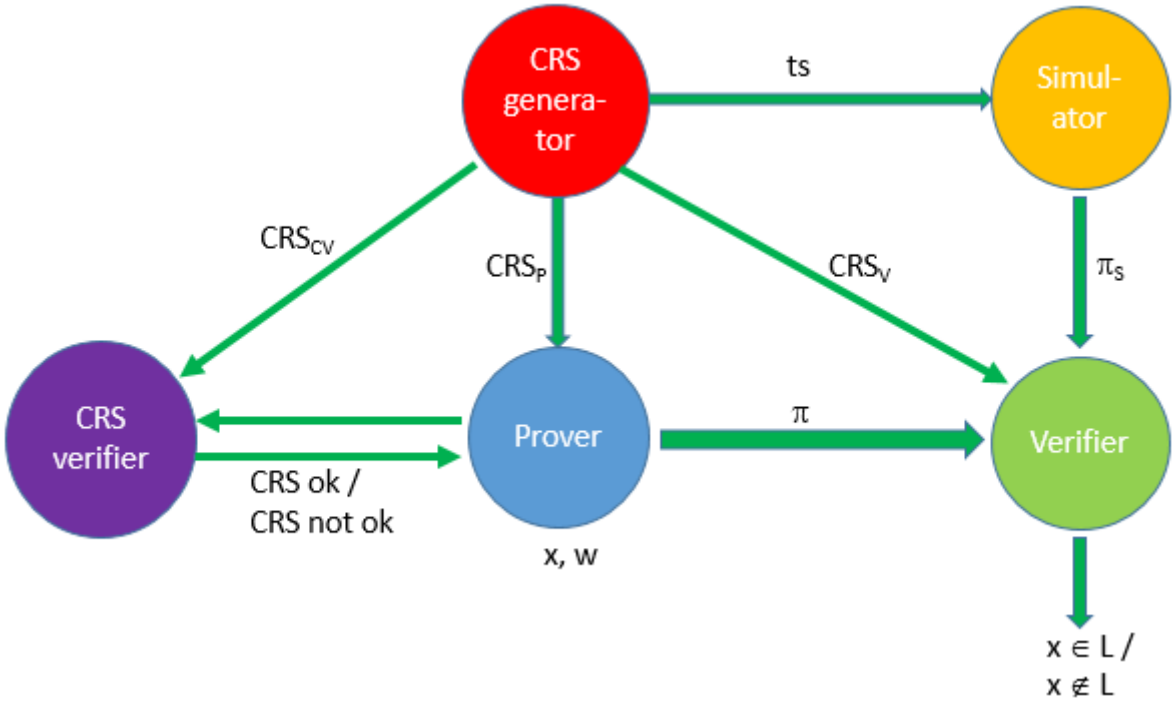

Fig. 2 Sub-ZK SNARK

The following subsections give a more detailed overview of all algorithms involved.

4.3 K_{tc} - CRS trapdoor generator

$K_{tc}(\mathbf{R}, z_R)$: Generate $tc = (\chi, \alpha, \beta, \gamma, \delta) \leftarrow_r Z_p^3 \times (Z_p^*)^2$.

K_{tc} is a probabilistic algorithm that is responsible for randomly generating a trapdoor tc. The trapdoor can be viewed as the secret key for the CRS and consists of five integers randomly chosen from Z_p and Z_p*. Here, p refers to the prime order of the groups G_1, G_2 and G_T. In this algorithm and the next, **R** is the binary relation {(x, w)} and z_R is the auxiliary information created by the relation generator. These will be given as inputs to all the algorithms.

4.4 K_{crs} - CRS generator

K_{crs} is a deterministic algorithm responsible for generating the CRS from the trapdoor tc = (α, β, γ, δ, χ). The CRS consists of three parts: CRS_P, CRS_V and CRS_{CV}, used respectively by the prover to create the proof, the verifier to check the proof and CV to check the CRS. Here, u, v, and w are circuit-specific constants and $1 < m_0 < m$. Alg. 1 refers to an algorithm for computing a unique polynomial of degree n [2].

$\mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc})$: Compute $(\ell_i(\chi))_{i=1}^n$ by using Alg. 1. Set $u_j(\chi) \leftarrow \sum_{i=1}^n U_{ij}\ell_i(\chi)$, $v_j(\chi) \leftarrow \sum_{i=1}^n V_{ij}\ell_i(\chi)$, $w_j(\chi) \leftarrow \sum_{i=1}^n W_{ij}\ell_i(\chi)$ for all $j \in \{0, \ldots, m\}$. Let

$$\mathsf{crs}_{\mathsf{P}} \leftarrow \left(\begin{array}{l} \left[\alpha, \beta, \delta, \left(\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\delta}\right)_{j=m_0+1}^{m}\right]_1, \\ \left[(\chi^i \ell(\chi)/\delta)_{i=0}^{n-2}, (u_j(\chi), v_j(\chi))_{j=0}^{m}\right]_1, \left[\beta, \delta, (v_j(\chi))_{j=0}^{m}\right]_2 \end{array} \right),$$

$$\mathsf{crs}_{\mathsf{V}} \leftarrow \left(\left[\left(\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\gamma}\right)_{j=0}^{m_0}\right]_1, [\gamma, \delta]_2, [\alpha\beta]_T \right),$$

$$\mathsf{crs}_{\mathsf{CV}} \leftarrow \left(\left[\gamma, (\chi^i)_{i=1}^{n-1}, (\ell_i(\chi))_{i=1}^{n}\right]_1, \left[\alpha, \chi, \chi^{n-1}\right]_2 \right).$$

Return $\mathsf{crs} \leftarrow (\mathsf{crs}_{\mathsf{CV}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{crs}_{\mathsf{V}})$.

Fig. 3 CRS generation algorithm

4.5 K_{ts} - Simulation trapdoor generator

$K_{ts}(\mathbf{R}, z_R, tc)$: Set $ts = (\chi, \alpha, \beta, \delta)$.

K_{ts} is a deterministic algorithm that creates a trapdoor for the simulator ts from the CRS trapdoor tc, with which the simulator can extract the CRS trapdoor from the CRS.

4.6 P – Prover

$\boldsymbol{\pi} \leftarrow \mathsf{Prove}(R, \boldsymbol{\sigma}, a_1, \ldots, a_m)$: Pick $r, s \leftarrow \mathbb{F}$ and compute a $3 \times (m + 2n + 4)$ matrix Π such that $\boldsymbol{\pi} = \Pi\boldsymbol{\sigma} = (A, B, C)$ where

$$A = \alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta \qquad B = \beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta$$

$$C = \frac{\sum_{i=\ell+1}^{m} a_i \left(\beta u_i(x) + \alpha v_i(x) + w_i(x)\right) + h(x)t(x)}{\delta} + As + rB - rs\delta.$$

Fig. 4 Prover's algorithm

The prover generates a proof from CRS_P, statements $a_1 \ldots m_0$ and witnesses $a_{m0+1} \ldots a_m$. The proof consists of three group elements A, B and C, so that $A \in G_1$, $B \in G_2$ and $C \in G_1$.

The prover algorithm is only run on the condition that the CV algorithm run before output 1.

4.7 V – Verifier

The verifier receives a proof (either the real proof from the prover or a simulated proof from the simulator) and checks whether the statements are true. Like in all pairing-based proof systems, the check is done by checking that a pairing equation holds. Both the proof and simulated proof consist of $A \in G_1$, $B \in G_2$ and $C \in G_1$. Using A, B, C (denoted by **a**, **b**, and **c** in Fig. 5), the statements and the CRS_V, the verifier verifies that the following equation holds:

$$\mathsf{V}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x} = (A_1, \ldots, A_{m_0}), \pi = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})): \text{ assuming } A_0 = 1, \text{ check that}$$

$$\mathfrak{a} \bullet \mathfrak{b} = \mathfrak{c} \bullet [\delta]_2 + \left(\sum_{j=0}^{m_0} A_j \left[\frac{u_j(x)\beta + v_j(x)\alpha + w_j(x)}{\gamma}\right]_1\right) \bullet [\gamma]_2 + [\alpha\beta]_T$$

Fig. 5 Verifier's algorithm

The verifier outputs 1 if the pairing equation holds, 0 otherwise. The verifier has been constructed so that it would accept proofs both from the prover and the simulator with an equal probability, while not being able to distinguish where the proof originated from.

4.8 S – Simulator

$$\boldsymbol{\pi} \leftarrow \mathsf{Sim}(R, \boldsymbol{\tau}, a_1, \ldots, a_\ell): \text{ Pick } A, B \leftarrow \mathbb{F} \text{ and compute } C = \frac{AB - \alpha\beta - \sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta}.$$
$$\text{Return } \boldsymbol{\pi} = (A, B, C).$$

Fig. 6 Simulator's algorithm

The simulator has access to the statements and CRS along with the simulation trapdoors, based on which he generates a simulated proof by computing three group elements A, B and C, just like the prover. The difference is that while the prover has access to the statements $a_1, \ldots, a_{m0}$ (m_0 is denoted by *l* in Fig. 6) and witnesses $a_{m0+1}, \ldots, a_m$, the simulator can access only the statements and must generate a simulated proof without using witnesses.

The simulator uses an extractor algorithm to extract the simulation trapdoors from the CRS in order to use them for generating the simulated proof. For extraction, we need either the extractor to be computationally unbounded, or use a knowledge assumption. This article considers both options. The knowledge assumption used is BDH-KE (Bilinear Diffie-Hellman Knowledge of Exponents). As per the BDH-KE assumption, given generators $g_1 = [1]_1 \in G_1$ and $g_2 = [1]_2 \in G_2$, and given $[\alpha_1]_1$ and $[\alpha_2]_2$, so that $[1]_1 \bullet [\alpha_2]_2 = [\alpha_1]_1 \bullet [1]_2$, we can assume that the adversary knows the value $\alpha_1 = \alpha_2$. This implies that if somebody is able to generate an element in two different groups, then they know the value of this element. It also follows that it is not possible to generate this element in two different groups without knowing its value.

4.9 CV – CRS verifier

$\mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs})$:

1. For $\iota \in \{\gamma, \delta\}$: check that $[\iota]_1 \neq [0]_1$
2. For $\iota \in \{\alpha, \beta, \gamma, \delta\}$: check that $[\iota]_1 \bullet [1]_2 = [1]_1 \bullet [\iota]_2$,
3. For $i = 1$ to $n-1$: check that $\left[\chi^i\right]_1 \bullet [1]_2 = \left[\chi^{i-1}\right]_1 \bullet [\chi]_2$,
4. Check that $([\ell_i(\chi)]_1)_{i=1}^n$ is correctly computed by using Alg. 2,
5. For $j = 0$ to m:
   (a) Check that $[u_j(\chi)]_1 = \sum_{i=1}^n U_{ij} [\ell_i(\chi)]_1$,
   (b) Check that $[v_j(\chi)]_1 = \sum_{i=1}^n V_{ij} [\ell_i(\chi)]_1$,
   (c) Set $[w_j(\chi)]_1 \leftarrow \sum_{i=1}^n W_{ij} [\ell_i(\chi)]_1$,
   (d) Check that $[v_j(\chi)]_1 \bullet [1]_2 = [1]_1 \bullet [v_j(\chi)]_2$,
6. For $j = 0$ to m_0: check that $[(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\gamma]_1 \bullet [\gamma]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$,
7. For $j = m_0 + 1$ to m: check that $[(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\delta]_1 \bullet [\delta]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$,
8. Check that $\left[\chi^{n-1}\right]_1 \bullet [1]_2 = [1]_1 \bullet \left[\chi^{n-1}\right]_2$,
9. For $i = 0$ to $n-2$: check that $\left[\chi^i \ell(\chi)/\delta\right]_1 \bullet [\delta]_2 = \left[\chi^{i+1}\right]_1 \bullet \left[\chi^{n-1}\right]_2 - \left[\chi^i\right]_1 \bullet [1]_2$,
10. Check that $[\alpha]_1 \bullet [\beta]_2 = [\alpha\beta]_T$.

Fig. 7 CRS verification algorithm

The CRS verification algorithm performs a number of checks to ensure that the CRS has been generated correctly and does not include anything that would enable the verifier to extract any additional information from the prover's argument.

Firstly, the algorithm checks that neither of the random variables γ and δ would be equal to 0 in G_1. This is necessary, because for instance if $[\delta]_1 = [0]_1$, the argument element A constructed by the prover would lose its randomness, which can compromises zero-knowledge and leak information.

Secondly, for all 5 components (α, β, γ, δ and χ) of the trapdoor tc, a check is performed that $[i]_1 \bullet [1]_2 = [i]_T = [1]_1 \bullet [i]_2$. This check is necessary to avoid the subverter picking unsuitable non-group elements for random variables.

Similarly, all other checks ensure that the proofs generated by the prover and the simulator will have the same distributions and thus the zero-knowledge property holds and no information is leaked along with the proof.

4.10 Security properties of the protocol

In order for the sub-ZK SNARK to be secure, it must satisfy all the properties guaranteeing security that the name sub-ZK SNARK entails. The security properties are as follows:

- Subversion-completeness. In cryptography, completeness is achieved when an honest prover always convinces an honest verifier. Subversion-completeness additionally requires that if the honest CRS generator generated the CRS correctly, then it passes the CRS verification.
- Computational knowledge-soundness. Soundness means that a dishonest prover should not be able to convince an honest verifier, except for with a negligible probability that comes from guessing correctly. Computational knowledge-soundness states that the

probability of a dishonest prover creating an acceptable proof for an incorrect statement (one that is not in the language) and the existence of an extraction function for the witness is almost non-existent. If the verifier accepts a proof from the prover and there exists an extraction function for extracting the witness, then from this it can be inferred that the prover is in fact honest and really knows the witness.

- Statistical sub-zero-knowledge. In addition to the ordinary zero-knowledge definition that no information is leaked with the proof, the condition that the prover does not reveal any information with creating the argument even in the case of a subverted CRS generator must also apply. Note that this property must be fulfilled even when the CRS generator is efficient and the adversary (in the role of the proof distinguisher) is computationally undbounded.

The paper includes proofs that all of the defined properties hold for the constructed sub-ZK SNARK, but for simplicity the proofs are omitted in this report. Full proofs can be found in sections 5 and 6 of the original paper [1].

4.11 Efficiency and implementations

To support this paper, two implementations of the new sub-ZK SNARK were carried out to be compared, in addition to an already existing implementation of Groth's ZK-SNARK. The first implementation included all algorithms as described in this report. Since the CV algorithm proposed was not efficient when implemented, a new batched CV algorithm was proposed, which uses a batching technique to speed it up.

Since the sub-ZK SNARK is based on Groth's ZK-SNARK, they are comparable in terms of efficiency, especially as several parts of the algorithm (the prover, verifier and simulator) were left unchanged. The most notable addition, the CRS verification algorithm CV, is only called once (by the prover), which means that CV and P can have the same computational complexity. A common component that differs for both algorithms is the CRS generation. Due to the addition of new elements in the CRS algorithm, the sub-ZK SNARK CRS generation is expected to take more time than for Groth's ZK-SNARK.

The sub-ZK SNARK was implemented in C++ using a "libsnark" library.

Comparison of the CRS generation algorithm for the implementations of Groth's ZK-SNARK and the sub-ZK SNARK on an input size of 1000 with a different number of multiplication gates, results shown in seconds:

Protocol \ Circuit size	**30 000**	**60 000**	**120 000**	**250 000**	**500 000**
Groth's SNARK	5.43	9.83	17.3	32.5	61.1
Sub-ZK SNARK	7.30	13.1	22.9	43.0	89.9

According to the results, the CRS generation for sub-ZK SNARK is on average 30% slower than for Groth's ZK-SNARK. Considering the additions to the algorithm and that the CRS generation algorithm is only run once, this difference is acceptable.

The efficiency of the CRS verification algorithm in comparison to the prover's algorithm on an input size of 1000 with a different number of multilication gates, results shown in seconds:

Algorithm \ Circuit size	**30 000**	**60 000**	**120 000**	**250 000**	**500 000**
CV	5.43	9.57	17.2	32.3	62.8
Prover	5.57	10.5	19.5	38.0	74.0

The results confirm that the CRS verification algorithm takes slightly less time than the prover. The difference becomes more and more distinguishable as the circuit size increases.

Another thing that was compared is the CRS length, since the CRS should not increase by too much to guarantee efficiency of the protocol. The CRS length for Groth's ZK-SNARKs is 4m + n + 8 element (m stands for the number of wires and n for the number of multiplication gates). For sub-ZK SNARKs, the CRS length is 4m + 3n + 16 elements. On average, the CRS length was found to have increased by approximately 30% compared to the CRS string used in Groth's zk-SNARK.

5 Conclusion

The paper, on which this report is based on, studied the case when the CRS generator for ZK-SNARKs has been subverted. A new ZK-SNARK construction was described, which is subversion-resistant, enabling the prover to run an algorithm for verifying that the CRS has been generated correctly before constructing his own proof. This guarantees that the protocol is zero-knowledge even if the CRS generator is malicious. The implementations of this protocol proved it to be of a comparable efficiency with the ZK-SNARK that it was based on.

6 References

[1] Abdolmaleki B., Baghery K., Lipmaa H., Zając M. (2017) A Subversion-Resistant SNARK. In: Takagi T., Peyrin T. Advances in Cryptology – ASIACRYPT 2017. ASIACRYPT 2017. Lecture Notes in Computer Science, vol 10626. Springer, Cham

[2] Groth J. (2016) On the Size of Pairing-Based Non-interactive Arguments. In: Fischlin M., Coron JS. Advances in Cryptology – EUROCRYPT 2016. EUROCRYPT 2016. Lecture Notes in Computer Science, vol 9666. Springer, Berlin, Heidelberg

[3] Bellare M., Fuchsbauer G., Scafuro A. (2016) NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion. In: Cheon J., Takagi T. Advances in Cryptology – ASIACRYPT 2016. ASIACRYPT 2016. Lecture Notes in Computer Science, vol 10032. Springer, Berlin, Heidelberg

[4] https://courses.cs.ut.ee/MTAT.07.014/2018_fall/uploads/Main/CP2016.L15.pdf (visited 25.11.2018)

[5] https://z.cash/technology/zksnarks/ (visited 25.11.2018)