Designated-Verifier zk-SNARKs Made Easy

Chen Li[1,2] and Fangguo Zhang[1,2] ✉

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China
[2] Guangdong Province Key Laboratory of Information Security Technology, Guangzhou 510006, China
lich368@mail2.sysu.edu.cn, isszhfg@mail.sysu.edu.cn

Abstract. Zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) is a kind of proof system that enables a prover to convince a verifier that an NP statement is true efficiently. In the last decade, various studies made a lot of progress in constructing more efficient and secure zk-SNARKs. Our research focuses on designated-verifier zk-SNARKs, where only the verifier knowing some secret verification state can be convinced by the proof. A natural idea of getting a designated-verifier zk-SNARK is encrypting a publicly-verifiable zk-SNARK's proof via public-key encryption. This is also the core idea behind the well-known transformation proposed by Bitansky *et al.* in TCC 2013 to obtain designated-verifier zk-SNARKs. However, the transformation only applies to zk-SNARKs which requires the complicated trusted setup phase and sticks on storage-expensive common reference strings. The loss of the secret verification state also makes the proof immediately lose the designated-verifier property.
To address these issues, we first define "strong designated-verifier" considering the case where the adversary has access to the secret verification state, then propose a construction of strong designated-verifier zk-SNARKs. The construction inspired by designated verifier signatures based on two-party ring signatures does not use encryption and can be applied on any public-verifiable zk-SNARKs to yield a designated-verifiable variant. We introduce our construction under the circuit satisfiability problem and implement it in Circom, then test it on different zk-SNARKs, showing the validity of our construction.

Keywords: zero-knowledge proof · SNARKs · designated verifier · circuit satisfiability

1 Introduction

A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) for an NP relation $\mathcal{R}$ enables a prover to produce a proof π, which convinces a verifier his knowledge of a secret witness w satisfying an instance u i.e. $(u, w) \in R$. Also, the proof π must not reveal anything about w (zero-knowledge) and its length and verification time must be sublinear in the size of u and w (succinctness). Building efficient and practical zk-SNARKs has become a hotspot

of cryptographic research in recent years and there has been a large number of constructions from different computational models. Gennaro *et al.* [24] use a new characterization of NP relations called Quadratic Span Programs (QSP) to reduce arithmetic circuits satisfiability problems and constructed a zk-SNARK where the proof only contains 9 group elements. The QSP characterization is generalized into Quadratic Arithmetic Programs (QAP) by Parno *et al.* [38], and they proposed Pinocchio which significantly reduces setup time, prover time and proof size and be used in practical applications including the cryptocurrency Zcash [8] to achieve anonymity and prevent double-spending. Groth16 [29] is a further optimized construction where the proof size is only 3 group elements and is easier to verify. These constructions are built upon the classic pre-quantum discrete logarithm type assumptions and the information-theoretic tool linear probabilistically-checkable proof (LPCP) where the prover is restricted to compute a linear function of verifier's queries [13, 31]. A trusted setup phase is also required in these constructions as the party which runs the setup algorithm has access to the secret randomness and can forge proofs using them. Recent zk-SNARKs use different building blocks, such as the ZKBoo series [19, 26] and Ligero series [2, 12] from MPC-in-the-Head [32], and Aurora [10], Fractal [21], Spartan [40] and Brakedown [27] from polynomial IOPs. These constructions target post-quantum security and transparent setup, which means the randomness used in the setup phase is public and the deployment in the real world can be simplified.

Typically, zk-SNARKs are designed in the publicly-verifiable model, which means the proof can be verified by everyone. Nevertheless, sometimes we only want the proof only convince a specified group of people. For example, in e-voting, the voting center needs to prove to the voter that he has indeed cast his vote, but the proof can also disclose the fact that he has participated in the vote to others, which impacts anonymity. In business trading, both parties involved create proof to prove the validity of the transaction, but they might not wish a third party to be informed of the transaction. Another scenario is that the proof might be some sort of paid content and the prover just wants to give paid members access to the proof. For such use cases, an alternative line of research focusing on designed-verifier zk-SNARKs, where the verifier is required to hold a secret verification state to verify the proof, has been proposed. Designated-verifier zk-SNARKs can be obtained by transforming existing publicly-verifiable zk-SNARKs. A natural idea is to enable "access control" to the proofs via public-key encryption. Campanelli *et al.* [18] pointed out that if there exists a publicly-verifiable SNARK (zero-knowledge property is not required) and a public-key encryption scheme, then a key-less designated-verifier zk-SNARK can be directly obtained by encrypting the proof with the public key and treat the secret key as the verification state. However, the key-less zero-knowledge property mentioned here is weaker than the standard one, as it requires that the proof reveals nothing about the witness only if the adversary does not hold the verification state. The adversary's ability is limited in this case. Another widely utilized transformation is the efficient compiler proposed

by Bitansky *et al.* [13] from LPCP-based zk-SNARKs by applying additively homomorphic encryption on the common reference string (CRS).

1.1 The "LIPs to Designated-Verifier zk-SNARKs" compiler

At a very high level, Bitansky's compiler is performed in the following way. A two-message linear interactive proof (LIP) is constructed from LPCP first. In this case, the prover's proof is a linear combination of elements in the CRS generated during the zk-SNARK's trusted setup phase. To make it designated-verifier, the compiler involves a cryptographic primitive called linear-only encryption which only supports linear homomorphism. Now the trusted setup phase additionally generates a keypair for the encryption, encrypts the CRS and sets the secret key as an extra verification state. The prover runs the LIP's prover algorithm and invokes the homomorphic add on the encrypted CRS to output the proof. Then, the verifier decrypts the proof and decides whether to accept or reject it by running the LIP's verifier algorithm. Candidate encryption schemes that can be used in this compiler include variants of Paillier [37], Elgamal [23] and Benaloh [11] encryption, which all satisfy the homomorphism property.

This provides a general template for constructing Designated-Verifier zk-SNARKs and is used as a general blueprint in many related studies. Boneh *et al.* [14] improved the compilation by constructing from LPCP directly to get rid of the communication complexity and soundness penalty introduced in the LIP construction, and using a linear-only encryption scheme based on LWE to obtain a Designated-Verifier zk-SNARK. Their subsequent work [15] gives a lattice-based Designated-Verifier zk-SNARK with quasi-optimal prover complexity. Gennaro *et al.* [25] and Ishal *et al.* [33]'s work make further improvements in efficiency. There are also relevant works for pre-quantum zk-SNARKs, recently Zhu *et al.* [45] substituted pairing checks with Σ-protocols in the CRS consistency verification of an improved variant Groth16 which satisfies subversion zero-knowledge, making it compatible with the compiler.

The compiler only applies to zk-SNARKs where the CRS is required for each statement to be proved. As a result, the resulting Designated-Verifier zk-SNARKs have to stick on the trusted setup from a trusted party for each statement to ensure the secret randomness, which could be used to forge valid proofs and often referred to as "toxic waste" for this reason, is erased after publishing the CRS. However, such a trustworthy third party barely exists in the real world. While the ideal trusted third party can be substituted with secure multi-party computation [9,16,17], this is still an expensive and verbose procedure and might be vulnerable to subversion. To resolve this issue, a large number of zk-SNARKs with transparent setup instead of the trusted setup phase has been proposed in recent years [2, 6, 7, 10, 21, 27, 40, 43]. Unfortunately, the previously mentioned compiler does not apply to any of them because the prover is not restricted to computing linear functions (of the CRS) in these zk-SNARKs. Therefore we cannot construct Designated-Verifier zk-SNARKs from them.

Another drawback to the encryption-based construction is that it is not secure against stronger adversaries. Consider an adversary that performs an at-

tack on the designated verifier and successfully steals the secret key (or is made public by the verifier himself). In these situations, previously created proofs immediately lost the designated-verifier property. Therefore, we need to consider stronger security notions of Designated-Verifier zk-SNARKs which can resist such attacks.

After discussing these prior works, we can form our research question: Is there such a method of constructing Designated-Verifier zk-SNARKs other than encryption, which can be applied to as many existing zk-SNARKs as possible, whether they are pre-quantum or post-quantum, require trusted setup or transparent, and which also makes the designated-verifier property more difficult to break?

1.2 Our Contributions

In this paper, we focus on constructing Designated-Verifier zk-SNARKs in an easier and more generic way which also satisfies stronger security notions. We believe that our work can indicate a new direction in the study of zk-SNARKs.

We present several contributions to address the above research question:

1. We give a more formal and stronger simulation-based definition of Designated-Verifier zk-SNARKs inspired by designated verifier signatures proposed by Chaum [20] and Jakobsson *et al.* [34] for the first time since the proof in zk-SNARKs can be considered as a "signature" for knowing the secret witness satisfying the given instance. We call this definition "stronger" because we give the adversary access to the verification state in the definition.
2. We propose a new construction of Designated-Verifier zk-SNARKs which satisfies the stronger definition above. The construction is inspired by two-party ring signatures, which is also a way to construct designated verifier signatures [4, 39]. It requires the verifier to hold a statement indicating his identity, and then the prover composes this circuit with what he wants to prove into a new instance and uses a zk-SNARK to create the proof of the new instance as usual. The construction does not use encryption and has no additional requirements for the underlying zk-SNARK, therefore we consider our construction to be easier and more generic.
3. We implement our construction in Circom [5], a programming language for building circuits, then tests with two state-of-the-art zk-SNARKs: Groth16 [29] and Aurora [10], which indicates that our construction can be applied to zk-SNARKs either requires a trusted setup or transparent, pre-quantum or post-quantum. We also evaluate the proof size, prover time and verifier time for the prover's circuit in different sizes and using our construction or not, then analyze the potential additional overhead of our construction.

2 Preliminaries

We recall the definition of zk-SNARKs here.

Definition 1 (zk-SNARKs). *A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) is a tuple of PPT algorithms* $\prod = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *such that:*

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to (\mathsf{crs}, \mathsf{st}, \mathsf{td})$ *On input an NP relation* $\mathcal{R}$ *over public parameters, outputs a common reference string (CRS)* crs*, the corresponding verification state* st *and a simulation trapdoor* td*.*
- $\mathsf{Prove}(\mathsf{crs}, u, w) \to \pi$ *On input an instance* u *and the prover's secret witness* w*, outputs a proof* π*.*
- $\mathsf{Verify}(\mathsf{crs}, \mathsf{st}, u, \pi) \to \{0, 1\}$ *On input an instance* u*, a proof* π *and the verification state* st*, returns* 1 *if the proof is accepted and* 0 *otherwise.*

And satisfies the following properties:

- ***Completeness:*** *An honest prover with the true witness of the instance should convince an honest verifier. Formally, for all* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{st}, u, \pi) = 1 \middle| \begin{matrix}(\mathsf{crs}, \mathsf{st}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ (u, w) \in \mathcal{R} \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, u, w)\end{matrix}\right] = 1 \tag{1}$$

- ***Knowledge soundness:*** *For any PPT adversary, it is difficult to create a valid proof* π *without holding a valid witness. Formally, for any adversary* $\mathcal{A}$ *with auxiliary inputs* z*, there exists a PPT extractor* $\mathcal{E}$ *such that:*

$$\Pr\left[\begin{matrix}\mathsf{Verify}(\mathsf{crs}, \mathsf{st}, u, \pi) = 1 \\ \wedge (u, w) \notin R\end{matrix} \middle| \begin{matrix}(\mathsf{crs}, \mathsf{st}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ (u, \pi) \leftarrow \mathcal{A}(\mathsf{crs}, z) \\ w \leftarrow \mathcal{E}(\mathsf{crs}, \mathsf{st}, u, z)\end{matrix}\right] = \mathsf{negl}(\lambda) \tag{2}$$

- ***Zero-knowledge:*** *There exists an efficient simulator* $\mathsf{Sim}_{\mathsf{ZK}}$ *that can output a simulated proof* π' *with the simulation trapdoor* td *instead of the witness. The simulated proof* π' *is also valid and indistinguishable from the real proof* π*, which means that nothing about the witness is leaked. Formally, for all PPT distinguisher* $\mathsf{Dist}_{\mathsf{ZK}}$*:*

$$\Pr\left[\begin{matrix}\mathsf{Verify}(\mathsf{crs}, \mathsf{st}, u, \pi) = 1 \\ \wedge \mathsf{Dist}_{\mathsf{ZK}}(\mathsf{crs}, \mathsf{st}, u, \pi) = r\end{matrix} \middle| \begin{matrix}(\mathsf{crs}, \mathsf{st}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ (u, w) \in \mathcal{R} \\ r \xleftarrow{R} \{0, 1\} \\ \pi \leftarrow \begin{cases}\mathsf{Prove}(\mathsf{crs}, u, w) & r = 0 \\ \mathsf{Sim}_{\mathsf{ZK}}(\mathsf{crs}, \mathsf{td}, u) & r = 1\end{cases}\end{matrix}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \tag{3}$$

- ***Succinctness:*** *The proof* π *must be small and easy to verify. The size of the proof* π *and the verifier's time to check it is at most polylogarithmic in the size of the instance* u *and the witness* w *[41].*

Remark 1 (Transparent zk-SNARKs [6]). A zk-SNARK is transparent if the randomness used by Setup *and* Verify *is public. Other zk-SNARKs that do not satisfy*

this property commonly notate Setup *as the "trusted setup phase", since the non-public randomness (also known as "toxic waste") must be kept secret from the prover and can be used to forge proofs if leaked or not properly destroyed afterward.*

Remark 2 (Publicly-Verifiable and Designated-Verifier zk-SNARKs [13, 14, 33, 36]). A zk-SNARK is publicly-verifiable if Verify *only depends on the public* crs*. Otherwise, if* Setup *also outputs a verification state* st *which is used for* Verify*, and the security holds only if* st *remains secret against adversaries (only the holder of* st *can verify proofs), then we call such type of zk-SNARKs designated-verifier.*

The above definition is for zk-SNARKs for arbitrary NP relations. In this paper, we discuss zk-SNARKs under boolean circuit satisfiability (C-SAT) problems at first, as our construction is more convenient to state in terms of C-SAT, and C-SAT is NP-Complete so it can be reduced from any other NP problems in polynomial time.

Definition 2 (Boolean Circuit Satisfiability Problem). *The C-SAT problem of a boolean circuit* $C : \{0,1\}^n \to \{0,1\}$ *is defined by the relation* $\mathcal{R} = \{(a_1, \cdots, a_n) \in \{0,1\}^n : C(a_1, \cdots, a_n) = 1\}$.

3 A stronger definition of Designated-Verifier zk-SNARKs

Recall the definition of designated-verifier zk-SNARKs from above, the designated-verifier property depends on the secrecy of st. If the secrecy is lost, the previously created proofs immediately lose the designated-verifier property. This can happen in reality, as the verifier may accidentally leak st to an adversary: losing the storage device, man-in-the-middle attack on the network, or an even more extreme situation such as being forced to hand over st by threats.

Another scenario is that the verifier can share his authority of verifying the proof with others by giving st, or even just making the proof public to make it public-verifiable. Of course, it does not require the prover's consent.

For these situations, we need a stronger designated-verifier property which makes it impossible for the verifier to transfer his authority of verifying the proof, either by force or out of choice.

In the study of cryptology, there exists a cryptographic primitive named designated verifier signatures, proposed by Chaum [20] and Jakobsson *et al.* [34] independently. Notice that we can treat zk-SNARK proofs as a "signature" for the message "the prover knows a secret witness satisfying the given instance", which can be verified with the proof as the prover's "public key". So we can borrow the relevant definitions of designated verifier signatures to designated-verifier zk-SNARKs. This is the starting point of our research.

Jakobsson *et al.* gives the threat and trust model of designated verifier proofs in [34]. They figuratively name the two previously mentioned scenarios as "the demon attack" (taking total command of the verifier) and "the suicide attack" (transferring the verifier's identity to the adversary and then self-destructing).

More importantly, they use indistinguishability to define the designated-verifier property. We modify the definition slightly to make it adapt to zk-SNARK's definition.

Definition 3 (Strong Designated-Verifier zk-SNARKs). *A zk-SNARK* $\prod = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is strong designated-verifier if there exists an efficient simulator* $\mathsf{Sim}_{\mathsf{DV}}$ *that can output a simulated proof* π' *with the verification state* st *instead of the witness. The simulated proof* π' *is also valid and indistinguishable from the real proof* π*. Formally, for all PPT distinguisher* $\mathsf{Dist}_{\mathsf{DV}}$*:*

$$\Pr\left[\begin{array}{c} \mathsf{Dist}_{\mathsf{DV}}(\mathsf{crs},\mathsf{st},u,\pi)=r \\ \wedge \\ \left(r=0 \wedge \mathsf{Verify}(\mathsf{crs},\mathsf{st},u,\pi)=1 \right) \\ \vee r=1 \end{array} \middle| \begin{array}{c} (\mathsf{crs},\mathsf{st},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda,\mathcal{R}) \\ (u,w)\in\mathcal{R} \\ r \xleftarrow{R} \{0,1\} \\ \pi \leftarrow \begin{cases} \mathsf{Prove}(\mathsf{crs},u,w) & r=0 \\ \mathsf{Sim}_{\mathsf{DV}}(\mathsf{crs},\mathsf{st},u) & r=1 \end{cases} \end{array} \right] \leq \frac{1}{2}+\mathsf{negl}(\lambda) \tag{4}$$

Notice that this definition is very similar to the zero-knowledge property. The difference is that $\mathsf{Sim}_{\mathsf{ZK}}$ uses the simulation trapdoor td, which is hold neither by the prover nor the verifier. Some literature may refer the process of using the trapdoor as "rewinding". This can only happen in the ideal world. However, the verifier can run $\mathsf{Sim}_{\mathsf{DV}}$ with the verification state st in his hand indeed in the real world.

In the other words, given a proof π outputs from a designated-verifier zk-SNARK, we want that the adversary learns nothing about whether π is produced by the prover or the verifier (however, he can still verify the validity of π). Thus, the verifier can never convince the adversary that π is produced by the prover instead of the verifier himself [39]. Moreover, we give the access to the adversary to the verification state st. This is a stronger attack model then the previous definition, therefore we name the new definition as "strong designated-verifier". We also notice that this definition guarantees a property similar with the forward secrecy in key agreement protocols, as the leak of st cannot damage the designated-verifier property of proofs created before the leak. Of course, the leaked st should not be used to create new proofs in the future.

However, Bitansky's compiler [13] does not satisfy this stronger definition, as the distinguisher can be easily built with access to $\mathsf{st} = (sk, s)$ where in the context of zk-SNARKs sk is the secret key used to encrypt the CRS and s is the zk-SNARK's secret verification state (see construction 6.1 in the paper for details). The distinguisher first decrypts the proof π using sk then verifies it using the zk-SNARK's Verify algorithm. If the proof is a valid ciphertext of the chosen linear-only encryption scheme and it can pass the verification after decryption, then the probability that the proof is created by the prover, not simulated by the simulator, is overwhelming since the linear-only encryption's linear-only homomorphism property and zk-SNARK's knowledge soundness property makes it almost impossible for anyone who does not hold the witness w to forge the (encrypted) proof and fool the distinguisher. In short, anyone with access to st

will be convinced that the proof is indeed created by the prover with a valid witness, which does not correspond to the designated verifier property.

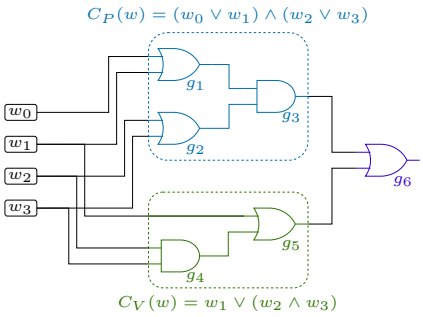
4 Generic Construction of (Strong) Designated-Verifier zk-SNARKs

In this section, we describe our new construction of designated-verifier zk-SNARKs. The new construction satisfies the previously defined stronger security notions. It is also a more generic construction, as it can be applied to any existing zk-SNARKs, whether it is pre-quantum or post-quantum, requires trusted setup or transparent, for free and without little extra cost for running time and proof size.

Recall that the definition of strong designated-verifier zk-SNARKs is derived from designated verifier signatures. Designated verifier signatures can be constructed with other basic cryptographic tools, such as undeniable signatures [34], ring signatures [4,39], key distribution mechanisms [44], key encapsulation mechanisms [28] and so on. We mainly pay attention to ring signature-based constructions. In ring signature schemes, several members form a ring and all ring members' public keys are used for signing and verifying. Of course, the signer's secret key is also required for signing. Due to the ring signature scheme's anonymity property, it is difficult to know who generated the signature among all possible ring members.

Now consider the special case where there are only two ring members named Alice and Bob. If Alice creates a ring signature, of course, it can be verified by Bob and any other verifier. The difference is that for Bob since the signature is not created by himself, he can definitely confirm that the signature is created by Alice. But for other verifiers, the signature will not be able to convince them since Bob could also be the signer (or to say the signatures created by Alice and Bob are indistinguishable) in their view. In this two-party case, Bob becomes a designated verifier. And since the ring cannot be changed after Alice creates the signature, Bob cannot transfer the identity of the designated verifier to someone else.

Similarly, we can also form a "ring" with the prover and the verifier for designated-verifier zk-SNARKs. Usually, the relation to be proved is public. Therefore we can use the relations and the circuits behind them to play the role of public keys. While the prover holding a circuit C_P that he wants to prove its satisfiability, the verifier is also required to hold a circuit C_V that only he knows a secret witness such that the circuit can be satisfied, which indicates his identity. To make the proof designated-verifier, what needs to be proved turns into "the statement the prover wants to prove to the verifier $\vee$ knowing some secret the verifier holds", or the satisfiability of the circuit $C_P \vee C_V$. This gives a feature similar to two-party ring signatures: both the prover and the designated verifier can create indistinguishable and valid proofs that can pass the verification. However, only the designated verifier can be convinced that the proof is created by the prover because it is not created by himself.



Input bits and logic gates' outputs						
w	g_1	g_2	g_3	g_4	g_5	g_6
$(1,0,1,0)$	1	1	1	0	0	1
$(0,0,1,1)$	0	1	0	1	1	1

Fig. 1. A tiny example of the composed circuit from C_P and C_V with different input sizes in our construction. The table on the right gives two inputs that satisfy C_P and C_V respectively, both of them also satisfy the composed circuit. Due to the zero-knowledge property of zk-SNARKs, the proof created from these two inputs is indistinguishable. Therefore, if you are a third party other than the prover and the designated verifier and received valid proof of this circuit, you cannot exclude the possibility that it was created by the verifier himself.

In the context of zk-SNARKs, the input is usually divided into two parts u (instance) and w (witness), where u denotes the public input and w denotes the private input that only the prover knows but does not want to reveal. Since we are describing the construction under boolean circuits, and C_P and C_V share the same input in the composed circuit $C_P \vee C_V$, we treat u as a part of the circuit and omit it for simplicity. Without loss of generality, we assume that the input sizes of C_P and C_V are the same. In cases where the input sizes are different, the circuit with a smaller input size can be padded by adding additional variables without any wire connections.

Now we can formally propose this designated-verifier zk-SNARK construction. Different from the previous definition of zk-SNARK (Definition 1), there is an extra procedure for assigning the designated verifier.

Construction 1 (Designated-Verifier zk-SNARKs from Arbitrary zk-SNARKs). *Let $\mathcal{R}_P$ and $\mathcal{R}_V$ be two C-SAT relations of boolean circuits $C_P : \{0,1\}^n \to \{0,1\}$ and $C_V : \{0,1\}^n \to \{0,1\}$, where the prover has $w_P \in \mathcal{R}_P$ which satisfies C_P. C_V can be used to check the verifier's identity and the verifier is assumed to hold $w_V \in \mathcal{R}_V$ which satisfies C_V. Let* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a zk-SNARK. A designated-verifier zk-SNARK* $(\mathsf{Assign}_{\mathsf{DV}}, \mathsf{Setup}_{\mathsf{DV}}, \mathsf{Prove}_{\mathsf{DV}}, \mathsf{Verify}_{\mathsf{DV}})$ *can be obtained as follows:*

- $\mathsf{Assign}_{\mathsf{DV}}(\mathcal{R}_P, \mathcal{R}_V) \to \mathcal{R}$ *Outputs a new relation* $\mathcal{R} = \{w \in \{0,1\}^n : C_P(w) \vee C_V(w) = 1\}$ *for the subsequent steps of the designated-verifier zk-SNARK.*
- $\mathsf{Setup}_{\mathsf{DV}}(1^\lambda, \mathcal{R}) \to (\mathsf{crs}, \mathsf{st}, \mathsf{td})$ *Works as* Setup *in the usual way.* w_V *is treated as a part of* st.
- $\mathsf{Prove}_{\mathsf{DV}}(\mathsf{crs}, u, w_P) \to \pi$ *Works as* Prove *in the usual way. The proof is different from the one created under* $\mathcal{R}_P$.
- $\mathsf{Verify}_{\mathsf{DV}}(\mathsf{crs}, \mathsf{st}, u, \pi) \to \{0,1\}$ *Works as* Verify *in the usual way.*

Theorem 1. $(\mathsf{Assign}_{\mathsf{DV}}, \mathsf{Setup}_{\mathsf{DV}}, \mathsf{Prove}_{\mathsf{DV}}, \mathsf{Verify}_{\mathsf{DV}})$ *from Construction 1 is a strong designated-verifier zk-SNARK.*

Proof. Completeness, knowledge soundness, zero-knowledge and succinctness directly follow from the corresponding properties of the underlying zk-SNARK.

For the strong designated-verifier property, since the verifier is assumed to hold $w_V \in \mathcal{R}_V$ *which satisfies* C_V*,* w_V *should also satisfy* $C_P \vee C_V$ *and he can do what the prover does in* Prove_{DV} *to simulate a valid proof. Due to the zero-knowledge property of the underlying zk-SNARK, both the proofs generated by* $\mathsf{Prove}(\mathsf{crs}, u, w_P)$ *and* $\mathsf{Prove}(\mathsf{crs}, u, w_V)$ *are indistinguishable from the simulated proofs generated by the simulator* $\mathsf{Sim}_{\mathsf{ZK}}$ *with the simulation trapdoor* td*, thus it is also difficult to distinguish between these two types of proofs.* □

Our construction is based on boolean circuits in the form $\{0,1\}^n \to \{0,1\}$. However, most currently existing zk-SNARKs and relevant toolchains are constructed targeting the satisfiability of arithmetic circuits like $\mathbb{F}^n \to \mathbb{F}^m$. As the arithmetic circuit satisfiability problem is also NP-complete, it is certainly feasible to reduce other NP problems to arithmetic circuits. This also includes boolean circuit satisfiability problems (adding additional constraints like $x(x-1)=0$ to ensure that variables must only be 0 or 1 and emulating logical gates with additions and multiplications). But this wastes $\log_2|\mathbb{F}|-1$ bits for each element in $\mathbb{F}$ and results in greater communication cost. The reduction of the whole problem can also be a bit expensive sometimes. For example, for problems like factorizing a large number, it would be simpler to instantiate it using an arithmetic circuit instead of a boolean circuit. Therefore, it is also necessary to consider how to implement the above construction under arithmetic circuits. For arithmetic circuits $\widetilde{C_P} : \mathbb{F}^n \to \mathbb{F}^{m_P}$ and $\widetilde{C_V} : \mathbb{F}^n \to \mathbb{F}^{m_V}$ (without loss of generality we can still assume that the input size is the same), the new relation to be proved now becomes something like $\{w \in \mathbb{F}^n : (\widetilde{C_P}(w) = p) \vee (\widetilde{C_V}(w) = v) = 1\}$ where $p \in \mathbb{F}^{m_P}, v \in \mathbb{F}^{m_V}$ are the expected outputs of $\widetilde{C_P}$ and $\widetilde{C_V}$. The construction consists of two parts: the equality testing and the OR relation, both can be implemented with a small number of additions and multiplications emulating the logical gates:

- Checking two variables' equality $a = b$ is equal to check $a - b = 0$. To check whether a variable x is zero or not, we need an auxiliary variable x_{inv} which is assumed to be the inverse of x (or 0 only if $x = 0$) and an additional constraint $x(1 - (x \cdot x_{inv})) = 0$ to ensure that. $z(x) = 1 - x \cdot x_{inv}$ gives the boolean result: if $x = 0$ then it outputs 1, otherwise 0. [42]
- Two arrays' equality $(a_1, \cdots, a_m) = (b_1, \cdots, b_m)$ is given by checking whether $\prod_{i=1}^{m} z(a_i - b_i) = 1$.
- For two boolean variables a, b in an arithmetic circuit, $a \vee b = 1$ is equal to the constraint $a + b - a \cdot b = 1$.

5 Concrete Implementation and Evaluation

We show a concrete implementation of our construction in Circom [5], an industrial and constraint-based language for building arithmetic circuits. Circom also comes with a compiler that compiles the code into corresponding rank-1 constraint system (R1CS) constraints and a program in C++ or WebAssembly for witness computation. An R1CS constraint is an equation of the form $A \cdot (1, w) \circ B \cdot (1, w) = C \cdot (1, w)$, which can represent several multiplication and addition gates in an arithmetic circuit.

Assume that $\widetilde{C_P}$ and $\widetilde{C_V}$ are declared using the following template:

```
template CircuitP(inLength, outLength) {
    signal input in[inLength];
    signal output out[outLength];

    // Constraints of the circuit
}

template CircuitV(inLength, outLength) {
    signal input in[inLength];
    signal output out[outLength];

    // Constraints of the circuit
}
```

Then we can construct the composed circuit in the following way:

```
// Use the IsEqual() template from the builtin Circomlib to
// test equality of two arrays
template IsEqualArray(length) {
    signal input in[2][length];
    signal output out;
    component eq[length];
    signal temp[length + 1];
    temp[0] <== 1;
    for (var i = 0; i < length; i++) {
        eq[i] = IsEqual();
        eq[i].in[0] <== in[0][i];
        eq[i].in[1] <== in[1][i];
        temp[i + 1] <== temp[i] * eq[i].out;
    }
    out <== temp[length];
}

template DVComposed(inLengthP, outLengthP, inLengthV, outLengthV) {
    // The larger of the input sizes of CircuitP and CircuitV
```

```
    signal input in[inLengthP > inLengthV ? inLengthP : inLengthV];
    // Public expected output of CircuitP and CircuitV
    signal input expectP[outLengthP];
    signal input expectV[outLengthV];

    // The two circuits share the same private input
    component circuitP = CircuitP(inLengthP, outLengthP);
    component circuitV = CircuitV(inLengthV, outLengthV);
    for (var i = 0; i < inLengthP; i++) {
        circuitP.in[i] <== in[i];
    }
    for (var i = 0; i < inLengthV; i++) {
        circuitV.in[i] <== in[i];
    }

    // Check if the output is the expected output
    component eqP = IsEqualArray(outLengthP);
    component eqV = IsEqualArray(outLengthV);
    for (var i = 0; i < outLengthP; i++) {
        eqP.in[0][i] <== circuitP.out[i];
        eqP.in[1][i] <== expectP[i];
    }
    for (var i = 0; i < outLengthV; i++) {
        eqV.in[0][i] <== circuitV.out[i];
        eqV.in[1][i] <== expectV[i];
    }
    // The final OR gate
    eqP.out + eqV.out - eqP.out * eqV.out === 1;
}

component main { public [expectP, expectV] } = DVComposed(...);
```

In practice, there is no unique way to construct $\widetilde{C_V}$. A feasible choice is the procedure of deriving the public key from a secret key of some public-key cryptosystem since a trusted public key of a particular person can be easily obtained from Public Key Infrastructures (PKI) in practice. We can demonstrate a simple example here, such as building a wrapper $\widetilde{C_V}$ of the `ECDSAPrivToPub` component from [1] to check ECDSA keypairs over secp256k1 curve[3]:

```
template CircuitV(inLength, outLength) {
    signal input in[4];
    signal output out[8];
```

[3] The secret key is a 256-bit integer, the (uncompressed) public key is a point on the curve and the x and y coordinates are also 256-bit integers. These integers are represented using four 64-bit integers in the circuit.

```
    component c = ECDSAPrivToPub(64, 4);

    for (var i = 0; i < 4; i++) {
        c.privkey[i] <== in[i];
    }
    for (var i = 0; i < 2; i++) {
        for (var j = 0; j < 4; j++) {
            c.pubkey[i][j] ==> out[i * 4 + j];
        }
    }
}
```

Assuming we have fetched the public key of the designated verifier from PKI[4], we can use it as a part of the public input of the composed circuit. The other part of the input is the expected output of $\widetilde{C_P}$.

```
[
    ...,
    "0xb9d3d296e43ff8e2", "0xce906d62615e2afc",
    "0xcf8561a3467ae190", "0xd5f103d0e369611b",
    "0xee9fb3b2b5d3bef4", "0xf8b75367a2bef8ee",
    "0x9a63e7e77f6bf6d4", "0xfb549ab9c5d25362"
]
```

For the designated verifier, he should hold the corresponding secret key, so it is possible for him to give the composed circuit the following private input with his secret key and create valid proofs without knowing any inputs satisfying $\widetilde{C_P}$:

```
{
    "in": [
        "0x71834475041066ec", "0x877e87fa54d39daa",
        "0x18ac73a985b5566d", "0x1b6b2d957e7b346b",
        ...
    ],
    "expectP": [...],
    "expectV": [
        "0xb9d3d296e43ff8e2", "0xce906d62615e2afc",
        "0xcf8561a3467ae190", "0xd5f103d0e369611b",
        "0xee9fb3b2b5d3bef4", "0xf8b75367a2bef8ee",
        "0x9a63e7e77f6bf6d4", "0xfb549ab9c5d25362"
    ]
}
```

[4] The keypair in this example is taken from the first set of test vectors from https://github.com/someone42/hardware-bitcoin-wallet/blob/master/test_vectors/keypairs.txt.

To verify the validity of our construction, we compiled the composed circuit with composite $\widetilde{C_P}$ of different number of constraints and the same ECDSA keypair checking $\widetilde{C_V}$ which contains about 95000 $\approx 2^{16.54}$ R1CS constraints[5], then prepared two sets of inputs that can satisfy $\widetilde{C_P}$ and $\widetilde{C_V}$ respectively and created proofs and witnesses of the composed circuit with these inputs, checking whether both of them are valid proofs. For the zk-SNARK's choice, We used the implementation of Groth16 [29] in snarkjs [30] for its popularity and first-class support for R1CS instances compiled from Circom. To check our construction is applied to transparent and post-quantum zk-SNARKs, we additionally chose a state-of-the-art protocol Aurora [10] in this field and ran the libiop [35] implementation with the same R1CS instances and inputs over the same BN128 prime field.

We keep records of the proof size, prover time and verifier time with and without using the designated-verifier construction we proposed to measure its extra overhead. The experiments are run on an Arch Linux virtual machine with 16 Intel Xeon w5-2465X CPU cores and 32 GB memory assigned.

Number of R1CS constraints in $\widetilde{C_P}$	Without DV	With DV
	P. time	P. time
2^{16}	2.45s	6.56s +167.6%
2^{17}	3.63s	7.16s +97.1%
2^{18}	5.90s	9.39s +59.2%
2^{19}	11.76s	15.24s +29.6%
2^{20}	22.85s	24.86s +8.8%

Table 1. Evaluation results of Groth16.

Number of R1CS constraints in $\widetilde{C_P}$	Without DV			With DV		
	P. size	P. time	V. time	P. size	P. time	V. time
2^{16}	132 KB	26.80s	0.56s	156 KB +17.9%	111.85s +317.4%	2.34s +319.4%
2^{17}	143 KB	55.50s	1.16s	156 KB +9.1%	112.10s +102.0%	2.36s +103.6%
2^{18}	156 KB	112.42s	2.27s	167 KB +7.4%	233.08s +107.3%	4.51s +98.4%
2^{19}	167 KB	236.52s	4.80s	181 KB +8.4%	482.78s +104.1%	8.95s +86.6%
2^{20}	181 KB	551.56s	9.30s	196 KB +7.8%	1003.36s +81.9%	17.85s +91.9%

Table 2. Evaluation results of Aurora.

We only compare the prover time for Groth16. This is because the proof size is constant, and though the verifier time is linear to the size of private and public inputs there is almost no difference since the verification is fast enough. The

[5] Constructions for deriving public keys in other widely used cryptosystems like RSA and Ed25519 also exist [3, 22], but not selected in evaluation because of the huge constraint number over 500000.

prover time is linear to the size of the R1CS instance, and since our construction composes $\widetilde{C_P}$ and $\widetilde{C_V}$ into a new circuit, the increase in prover time depends largely on the size of $\widetilde{C_V}$. In the experiment, the selected $\widetilde{C_V}$ with $2^{16.54}$ constraints will introduce a fixed 2-4s overhead to the prover time. For a small $\widetilde{C_P}$ with 2^{16} constraints, the increase is about $2^{16.54-16} \approx 145\%$ of the prover time without using the designated-verifier construction. But for an intermediate-sized $\widetilde{C_P}$ with over 2^{20} constraints, the increase is relatively negligible.

Aurora requires that the number of constraints must be a power of 2. For the $\widetilde{C_P}$ with 2^{16} constraints, the new composed circuit will contain $2^{16} + 2^{16.54} \approx 2^{17.30}$ constraints and then be padded to 2^{18}. Thus the proof size, prover time and verifier time are the same as creating a proof for a $\widetilde{C_P}$ with 2^{18} constraints without using the designated-verifier construction. An Aurora proof has size $O(\log^2 n)$, can be created in $O(n \log n)$ time and verified in $O(n)$ time where n is the number of constraints. Therefore, if composing $\widetilde{C_V}$ causes an increase in $\lceil \log_2 n \rceil$, the proof size will increase slightly and the prover time and verifier time will be doubled. Otherwise, there is no additional overhead.

6 Conclusion

We define Strong Designated-Verifier zk-SNARKs and then propose a new construction to fix the defect of existing designated-verifier's definition that the verifier may lose control of the secret verification state or make it public on his own, which breaks the designated-verifier property. The new construction, inspired by designated-verifier signatures based on two-party ring signatures, uses an additional circuit to validate the verifier's identity and composes it by the OR relation with the circuit that the prover wants to prove its satisfiability to ensure that anyone except the verifier cannot be convinced by the proof.

Our construction is more generic and easier than existing constructions since there is no need for special encryption to keep the proof designated verifier and our construction can be applied to any existing zk-SNARKs, especially for those more advanced zk-SNARKs that do not require the trusted setup phase and satisfy post-quantum security.

Due to the introduction of the additional circuit for the verifier's identity, the size of the statement to be proved becomes larger and the proof size, prover time and verifier time may increase. But this varies depending on the underlying zk-SNARK used. Regardless, choosing a smaller circuit can always reduce this extra overhead. This leaves room for improvement by relying on a simpler way to validate the verifier's identity.

References

1. 0xPARC: circom-ecdsa: Big integer arithmetic and secp256k1 ECC operations in circom (2024), https://github.com/0xPARC/circom-ecdsa

2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2087–2104. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134104
3. zkp application: circom-rsa-verify: Zero knowledge proof for RSA (2024), https://github.com/zkp-application/circom-rsa-verify
4. Au, M.H., Susilo, W.: Two-party (blind) ring signatures and their applications. In: Huang, X., Zhou, J. (eds.) Information Security Practice and Experience. pp. 403–417. Springer International Publishing, Cham (2014)
5. Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A., Baylina, J.: Circom: A circuit description language for building zero-knowledge applications. IEEE Transactions on Dependable and Secure Computing **20**(6), 4733–4751 (2023). https://doi.org/10.1109/TDSC.2022.3232813
6. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046 (2018), https://eprint.iacr.org/2018/046
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 701–732. Springer International Publishing, Cham (2019)
8. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
9. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure sampling of public parameters for succinct zero knowledge proofs. In: 2015 IEEE Symposium on Security and Privacy. pp. 287–304 (2015). https://doi.org/10.1109/SP.2015.25
10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 103–128. Springer International Publishing, Cham (2019)
11. Benaloh, J.: Dense probabilistic encryption. In: Proceedings of the workshop on selected areas of cryptography. pp. 120–128 (1994)
12. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: A new optimized sublinear IOP. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 2025–2038. CCS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3417893
13. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) Theory of Cryptography. pp. 315–333. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
14. Boneh, D., Ishai, Y., Sahai, A., Wu, D.J.: Lattice-based SNARGs and their application to more efficient obfuscation. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 247–277. Springer International Publishing, Cham (2017)
15. Boneh, D., Ishai, Y., Sahai, A., Wu, D.J.: Quasi-optimal SNARGs via linear multi-prover interactive proofs. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 222–255. Springer International Publishing, Cham (2018)

16. Bowe, S., Gabizon, A., Green, M.D.: A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In: Zohar, A., Eyal, I., Teague, V., Clark, J., Bracciali, A., Pintore, F., Sala, M. (eds.) Financial Cryptography and Data Security. pp. 64–77. Springer Berlin Heidelberg, Berlin, Heidelberg (2019)
17. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Paper 2017/1050 (2017), https://eprint.iacr.org/2017/1050
18. Campanelli, M., Khoshakhlagh, H.: Succinct publicly-certifiable proofs. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) Progress in Cryptology – INDOCRYPT 2021. pp. 607–631. Springer International Publishing, Cham (2021)
19. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1825–1842. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3133997
20. Chaum, D.: Private signature and proof systems, US Patent 5,493,614 (1996)
21. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 769–793. Springer International Publishing, Cham (2020)
22. Electron-Labs: ed25519-circom: Ed25519 implementation in circom (2024), https://github.com/Electron-Labs/ed25519-circom
23. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
24. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
25. Gennaro, R., Minelli, M., Nitulescu, A., Orrù, M.: Lattice-based zk-SNARKs from square span programs. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 556–573. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243845
26. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster Zero-Knowledge for boolean circuits. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 1069–1083. USENIX Association, Austin, TX (Aug 2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/giacomelli
27. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 193–226. Springer Nature Switzerland, Cham (2023)
28. Gong, B., Au, M.H., Xue, H.: Constructing strong designated verifier signatures from key encapsulation mechanisms. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). pp. 586–593 (2019). https://doi.org/10.1109/TrustCom/BigDataSE.2019.00084
29. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

30. iden3: snarkjs: zkSNARK implementation in JavaScript & WASM (2024), https://github.com/iden3/snarkjs
31. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short PCPs. In: Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07). pp. 278–291 (2007). https://doi.org/10.1109/CCC.2007.10
32. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing. pp. 21–30. STOC '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1250790.1250794
33. Ishai, Y., Su, H., Wu, D.J.: Shorter and faster post-quantum designated-verifier zkSNARKs from lattices. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 212–234. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484572
34. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Maurer, U. (ed.) Advances in Cryptology — EUROCRYPT '96. pp. 143–154. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
35. scipr lab: libiop: C++ library for IOP-based zkSNARKs (2024), https://github.com/scipr-lab/libiop
36. Nitulescu, A.: zk-SNARKs: a gentle introduction (2020), https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf
37. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
38. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. Commun. ACM **59**(2), 103–112 (jan 2016). https://doi.org/10.1145/2856449
39. Saeednia, S., Kremer, S., Markowitch, O.: An efficient strong designated verifier signature scheme. In: Lim, J.I., Lee, D.H. (eds.) Information Security and Cryptology - ICISC 2003. pp. 40–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
40. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 704–737. Springer International Publishing, Cham (2020)
41. Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552 (2023), https://eprint.iacr.org/2023/552
42. Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking Proof-Based verified computation a few steps closer to practicality. In: 21st USENIX Security Symposium (USENIX Security 12). pp. 253–268. USENIX Association, Bellevue, WA (2012), https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/setty
43. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943 (2018). https://doi.org/10.1109/SP.2018.00060
44. Yang, F.Y., Liao, C.M.: A provably secure and efficient strong designated verifier signature scheme. In: International Journal of Network Security. vol. 10, pp. 220–224 (2010)
45. Zhu, X., Song, X., Deng, Y.: Fast and designated-verifier friendly zkSNARKs in the BPK model. Cryptology ePrint Archive, Paper 2023/1806 (2023), https://eprint.iacr.org/2023/1806