Designated-Verifier zk-SNARKs Made Easy

Chen Li, Fangguo Zhang

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 indicated 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 designatedverifier" 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 publicverifiable 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.

Index Terms—zero-knowledge proof, SNARKs, designated verifier, circuit satisfiability

I. INTRODUCTION

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 ui.e. $(u, w) \in \mathbb{R}$. Also, the proof π must not reveal anything about w (zero-knowledge) and its length and verification time must be at most sublinear in the size of u and w (succinctness). There are many practical applications for zk-SNARK, such as privacy-preserving deep-learning [1], blockchains [2] and cryptocurrencies [3]. 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.* [4] 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.* [5], and they proposed Pinocchio which significantly reduces setup time, prover time and proof size and be used in practical applications including the cryptocurrency Zcash [3] to achieve anonymity and prevent double-spending. Groth16 [6] 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 prequantum discrete logarithm type assumptions, bilinear pairing and the information-theoretic tool linear probabilisticallycheckable proof (LPCP) where the prover is restricted to compute a linear function of verifier's queries [7], [8]. 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. Besides, These zk-SNARKs have to run the setup phase every time for each instance to be proved, which could be a barrier to real world deployments.

Later zk-SNARKs use different building blocks. The ZK-Boo series [9], [10] and Ligero series [11], [12] are based on the idea of MPC-in-the-Head [13]. Recently, it has become a trend to combine polynomial or vector commitments and interactive oracle proofs (IOPs) to design new zk-SNARKs. Such constructions include Marlin [14], Plonk [15], Aurora [16], Fractal [17], Spartan [18], Brakedown [19], Orion [20], Vortex [21] and HyperPlonk [22] from polynomial or vector commitments and interactive oracle proofs (IOPs). Marlin and Plonk (and its successors) are constructed in a "universal and updatable" fashion, which means that a single trusted setup can be applied to all instances of a certain bounded size and can be updated by any user if the current setup is considered corrupted. Others target transparent setup, which means the randomness used in the setup phase is public. These improvements in the setup phase simplify the deployment in the real world. Some of these zk-SNARKs also target postquantum security.

Typically, zk-SNARKs are designed in the publiclyverifiable model, which means the proof can be verified by everyone. There is also 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. Nevertheless, sometimes we only want the proof only convince a specified group of people. In this case, we can consider the designated-verifier property as an extra feature of zk-SNARKs. Here we give some potential use cases of designed-verifier zk-SNARKs.

• E-Voting. In e-voting, the voter needs to prove that the ballot indeed contains the option he selected, and the proof should only convince the voting center, just as in real voting which needs to go behind the cloth curtains of the voting station to cast his vote. Otherwise, we would know who he has voted for, which makes room for vote selling. On the other side, the voting center also needs to prove to the voter that they have indeed received his ballot, but the proof can also disclose the fact that he has participated in the vote to others, which impacts anonymity. By setting the voting center and the voter

as the designated verifier in the two cases above, we can ensure that nobody else will know whether the voter participated in the vote or not and who he has voted for.

- **Business trading.** In business trading, both parties involved create proof to prove the validity of the transaction, but this transaction may involve trade secrets and they might not wish a third party to be informed of the transaction, as well as not wanting to have traitors who could leak the details. By using designated-verifier zk-SNARKs in this case, even if the parties' competitors access the proof of the transaction, they can have a reason for suspecting that it is just a smoke screen since they are not the designated verifier and cannot be convinced by the proof.
- **Paid contents.** Another scenario is that the proof might be some sort of paid content. The prover just wants to give paid members (as the designated verifier) access to the proof. Moreover, the prover might not want them to share the proof content with others, just as some video-on-demand websites applying strict account sharing restrictions through checking geolocations and login device limits.

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 publickey encryption. Campanelli et al. [23] 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. [8] from LPCP-based zk-SNARKs by applying additively homomorphic encryption on the common reference string (CRS).

A. 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 designatedverifier, 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 [24], Elgamal [25] and Benaloh [26] 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. [27] 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 [28] gives a lattice-based Designated-Verifier zk-SNARK with quasi-optimal prover complexity. Gennaro et al. [29] and Ishal et al. [30]'s work makes further improvements in efficiency. There are also relevant works for pre-quantum zk-SNARKs, recently Zhu et al. [31] 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 [32]-[34], 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 universal setup [14], [15], [22] or transparent setup [11], [16]-[21], [35]-[37] instead of the trusted setup phase has been proposed in recent years. 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 attack 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?

B. 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:

- We give a more formal and stronger simulation-based definition of Designated-Verifier zk-SNARKs inspired by designated verifier signatures proposed by Chaum [38] and Jakobsson *et al.* [39] 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 [40], [41]. It requires the verifier to hold a statement indicating his identity (for example the secret key corresponding to his public key, or even the verifier's biometric authentication data including face, fingerprint and so on), 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 [42], a programming language for building circuits, then tests with three widely-used and state-of-the-art zk-SNARKs: Groth16 [6], Aurora [16] and Plonk [15], which indicates that our construction can be applied to zk-SNARKs either requires a trusted setup or transparent, pre-quantum or post-quantum and for different constraint systems. 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.

II. PRELIMINARIES

We recall the definition of zk-SNARKs here.

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

- Setup(1^λ, R) → (crs, st, td) On input an NP relation *R* over public parameters, outputs a common reference string (CRS) crs, the corresponding verification state st and a simulation trapdoor td.
- Prove(crs, u, w) $\rightarrow \pi$ On input an instance u and the prover's secret witness w, outputs a proof π .
- Verify(crs, st, u, π) → {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 $\lambda \in \mathbb{N}$:

$$\Pr \begin{bmatrix} \operatorname{Verify}(\operatorname{crs}, \operatorname{st}, u, \pi) & (\operatorname{crs}, \operatorname{st}, \operatorname{td}) \\ \leftarrow \operatorname{Setup}(1^{\lambda}, \mathcal{R}) \\ (u, w) \in \mathcal{R} \\ \pi \leftarrow \operatorname{Prove}(\operatorname{crs}, u, w) \end{bmatrix}$$
(1)
= 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:

$$\Pr \begin{bmatrix} \operatorname{Verify}(\operatorname{crs}, \operatorname{st}, u, \pi) & (\operatorname{crs}, \operatorname{st}, \operatorname{td}) \\ = 1 & \leftarrow \operatorname{Setup}(1^{\lambda}, \mathcal{R}) \\ (u, w) \notin R & (u, \pi) \leftarrow \mathcal{A}(\operatorname{crs}, z) \\ w \leftarrow \mathcal{E}(\operatorname{crs}, \operatorname{st}, u, z) \end{bmatrix}$$
(2)
= negl(λ)

• Zero-knowledge: There exists an efficient simulator Sim_{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 Dist_{ZK}:

$$\Pr \begin{bmatrix} \left| \begin{array}{c} (\mathsf{crs}, \mathsf{st}, \mathsf{td}) \\ & \leftarrow \mathsf{Setup}(1^{\lambda}, \mathcal{R}) \\ = 1 \\ & \wedge \\ \mathsf{Dist}_{\mathsf{ZK}}(\mathsf{crs}, \mathsf{st}, u, \pi) \\ = r \end{bmatrix} \\ \pi \leftarrow \begin{cases} \mathsf{Prove}(\mathsf{crs}, u, w) \\ & \mathsf{if} \ r = 0 \\ \mathsf{Sim}_{\mathsf{ZK}}(\mathsf{crs}, \mathsf{td}, u) \\ & \mathsf{if} \ r = 1 \end{cases} \end{bmatrix} \\ \leq \frac{1}{2} + \mathsf{negl}(\lambda) \end{cases}$$
(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 [43].

Remark 1 (Transparent zk-SNARKs [37]): 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 [8], [27], [30], [44]): A zk-SNARK is publiclyverifiable 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 be convinced by the proof), then we call such type of zk-SNARKs designatedverifier.

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 following relation:

$$\mathcal{R} = \{(a_1, \cdots, a_n) \in \{0, 1\}^n : C(a_1, \cdots, a_n) = 1\}$$
(4)

III. 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 [38] and Jakobsson *et al.* [39] 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 [39]. 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 = (Setup, Prove, Verify) is strong designated-verifier if there exists an efficient simulator Sim_{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 Dist_{DV}:

$$\Pr \begin{bmatrix} \operatorname{Dist}_{\mathsf{DV}}(\mathsf{crs},\mathsf{st},u,\pi) \\ = r \\ \land \\ \begin{pmatrix} \\ r = 0 \\ \\ \\ \\ \mathsf{Verify}(\mathsf{crs},\mathsf{st},u,\pi) \\ = 1 \end{bmatrix} \begin{pmatrix} (\mathsf{crs},\mathsf{st},\mathsf{td}) \\ \leftarrow \operatorname{Setup}(1^{\lambda},\mathcal{R}) \\ (u,w) \in \mathcal{R} \\ r \leftarrow \begin{cases} n \\ \\ r \leftarrow \\ \\ \mathsf{Prove}(\mathsf{crs},u,w) \\ \text{if } r = 0 \\ \\ \mathsf{Sim}_{\mathsf{DV}}(\mathsf{crs},\mathsf{st},u) \\ \text{if } r = 1 \end{bmatrix} \\ \leq \frac{1}{2} + \mathsf{negl}(\lambda) \end{aligned}$$
(5)

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

In other words, given a proof π outputs from a designatedverifier zk-SNARK, we want the adversary learn nothing about whether π is produced by the prover or the verifier. Thus, the verifier can never convince the adversary that π is produced by the prover instead of the verifier himself [40]. Moreover, we give access to the adversary to the verification state st. This is a stronger attack model than the previous definition, therefore we name the new definition as "strong designated-verifier". We also notice that this definition guarantees a property similar to 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 [8] does not satisfy this stronger definition, as the distinguisher can be easily built with access to 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 [8] 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 linearonly 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.

IV. 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



Fig. 1. A scenario example of constructing designated-verifier zk-SNARKs with encryption. Normally, only the designated verifier holding st can decrypt and verify the proof π , while others will not be convinced by the proof because they cannot verify it. However, in the latter two cases shown here, the designated verifier actively or passively gives st to someone else, and then the designated-verifier property is broken.

also a more generic construction, as it can be applied to any existing zk-SNARKs, whether it is pre-quantum or postquantum, 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 [39], ring signatures [40], [41], key distribution mechanisms [45], key encapsulation mechanisms [46] 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 OR knowing some secret the verifier holds", or the satisfiability of the circuit $C_P \lor C_V$. This gives a feature similar to two-party



Fig. 2. An illustration of our purposed construction, anyone outside of the "ring" cannot determine whether the proof is created by the prover or the verifier, even if they could verify the proof. But the designated verifier in the "ring" can definitely believe that the proof is created by the prover.

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.

Remark 3 (On the proper selection of C_V): A malicious verifier may submit a C_V to the prover, then claim that he actually doesn't know any witness satisfies C_V after receiving the proof of the satisfiability of $C_P \vee C_V$, thus turning the proof back into a plain proof of C_P 's satisfiability and destroying the designated-verifier property. For this reason, we need to emphasize that the C_V here should represent some relation that can be publicly verified and cannot be denied by the verifier. An example of such a relation is holding the secret key of a public 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. The public key can also attach signatures signed by other individuals or CAs the prover trusts. Another choice can be relations that the verifier made a public verifiable zero-knowledge proof previously. These relations use some evidence that cannot be erased to indicate the verifier's identity. On the contrary, if the C_V given by the verifier does not satisfy such a requirement, the prover should refuse to produce the proof.

Remark 4 (Does the designated-verifier property still hold?): One might argue that in our construction, the proof π can still be verified by anyone including the adversary, so this is actually a public-verifiable zk-SNARK and not a designated-verifier zk-SNARK. We need to point out that the essence of the designated-verifier property is not whether it can be verified, but that only the designated verifier can be convinced by the proof. The previously mentioned encryption-based Bitansky's compiler satisfies this (unless the secret key is leaked) because no one else can decrypt the proof, so they cannot verify and then be convinced by the proof. Our



Fig. 3. 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.

construction uses another approach which is somewhat similar to deniability to achieve the same result, since no one else can determine who created the proof.

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 : $\{0,1\}^n \rightarrow \{0,1\}$ and C_V : $\{0,1\}^n \rightarrow \{0,1\}$, 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 (Setup, Prove, Verify) be a zk-SNARK for any relation \mathcal{R} . A designated-verifier zk-SNARK (Assign_{DV}, Setup_{DV}, Prove_{DV}, Verify_{DV}) can be obtained as follows:

• Assign_{DV}($\mathcal{R}_P, \mathcal{R}_V$) $\rightarrow \mathcal{R}$ Outputs a new relation $\mathcal{R} =$

 $\{w \in \{0,1\}^n : C_P(w) \lor C_V(w) = 1\}$ for the subsequent steps of the designated-verifier zk-SNARK.

- Setup_{DV}(1^λ, R) → (crs, st, td) Works as Setup in the usual way. w_V is treated as a part of st.
- Prove_{DV}(crs, u, w_P) → π Works as Prove in the usual way. The proof is different from the one created under R_P.
- Verify_{DV}(crs, st, u, π) $\rightarrow \{0, 1\}$ Works as Verify in the usual way.

Theorem 1: (Assign_{DV}, Setup_{DV}, Prove_{DV}, Verify_{DV}) from Construction 1 is a strong designated-verifier zk-SNARK.

Proof: Completeness, knowledge soundness, zeroknowledge 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 $C_P \lor C_V$ and he can do what the prover does in Prove_{DV} to simulate a valid proof. Due to the zeroknowledge property of the underlying zk-SNARK, both the proofs generated by Prove(crs, u, w_P) and Prove(crs, u, w_V) are indistinguishable from the simulated proofs generated by the simulator Sim_{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 $\{0,1\}^n \rightarrow \{0,1\}$. 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) = 0to 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

$$\{w \in \mathbb{F}^n : (\widetilde{C}_P(w) = p) \lor (\widetilde{C}_V(w) = v) = 1\}$$
(6)

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 the variable x = a - b is zero or not, we need an auxiliary variable

$$x_{inv} = \begin{cases} x^{-1} & \text{if } x \neq 0\\ 0 & \text{if } x = 0 \end{cases}$$
(7)

and an additional constraint

$$x(1 - (x \cdot x_{inv})) = 0$$
 (8)

to ensure that

$$z(x) = 1 - x \cdot x_{inv} \tag{9}$$

gives the boolean result: if x = 0 then z(x) outputs 1, otherwise 0 [47].

• Two arrays' equality $(a_1, \cdots, a_m) = (b_1, \cdots, b_m)$ is given by checking whether

$$\prod_{i=1}^{m} z(a_i - b_i) = 1.$$
(10)

For two boolean variables a, b (output from the array equality test above in our construction) where the values are restricted to 0 or 1 even though in an arithmetic circuit, a∨b = 1 is equal to the constraint a+b-a·b = 1.

V. CONCRETE IMPLEMENTATION AND EVALUATION

A. Implementation

We show a concrete implementation of our construction in Circom [42], an industrial and constraint-based language for building arithmetic circuits. Circom also comes with a compiler that compiles the code into a rank-1 constraint system (R1CS) instance (often known as "arithmetization" as it uses polynomials to represent gates in a circuit) and a program in C++ or WebAssembly for witness computation. An R1CS instance is defined by three matrices $A, B, C \in \mathbb{F}^{m \times n}$ and a vector *io* containing public input and output. Denoting the vector consists of public input, public output and private input (collectively called "wires") as z = (1, w, io) (|z| = n), an NP relation of the instance and corresponding witnesses can be defined as

$$\mathcal{R}_{\text{R1CS}} = \{ \langle (\mathbb{F}, A, B, C, io), w \rangle : A \cdot z \circ B \cdot z = C \cdot z \}.$$
(11)

Considering each row of these matrices, then the above relation is equivalent to

$$a_i \cdot z \circ b_i \cdot z = c_i \cdot z \qquad \forall i \in \{1, \cdots, n\}.$$
(12)

Each equation of a single row is called an R1CS constraint, which can represent a multiplication gate and several addition gates in an arithmetic circuit.

Assume that C_P and 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];</pre>
    temp[i + 1] <== temp[i] * eq[i].out;</pre>
  }
  out <== temp[length];</pre>
}
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];</pre>
  }
  for (var i = 0; i < inLengthV; i++) {
    circuitV.in[i] <== in[i];</pre>
  }
  // Check if the output is the expected
  // output
  component eqP = IsEqualArray(outLengthP);
  component eqV = IsEqualArray(outLengthV);
  for (var i = 0; i < outLengthP; i++) {</pre>
```

```
eqP.in[0][i] <== circuitP.out[i];
eqP.in[1][i] <== expectP[i]; set
for (var i = 0; i < outLengthV; i++) { pro
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,
```

```
expectr,
expectV
] } = DVComposed(...);
```

As mentioned earlier, a feasible choice of constructing $\widetilde{C_V}$ is holding a secret key of a trusted public key. We can demonstrate a simple example here, such as building a wrapper $\widetilde{C_V}$ of the ECDSAPrivToPub component from [48] to check ECDSA keypairs over secp256k1 curve¹:

```
template CircuitV(inLength, outLength) {
   signal input in[4];
   signal output out[8];
   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 PKI², 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"
]
```

¹The 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.

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"
  ]
}
```

B. Comparison with zk-SNARKs without Designated-Verifier Settings

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 or 1600000 $\approx 2^{20.61}$ Plonkish constraints³, 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.

We evaluate our construction with the following zk-SNARKs, which cover as many types of zk-SNARKs as possible and have well-documented open-source implementations, allowing us to easily apply our custom (composed) circuits to them:

- Groth16 [6], the classic, simple and popular pairing-based *per-circuit setup* zk-SNARK. We use the implementation in snarkjs [51] for its popularity and first-class support for R1CS instances compiled from Circom.
- Aurora [16], a *transparent and post-quantum* zk-SNARK built upon polynomial commitments and IOPs. We use the implementation in libiop [52]. Since it uses C++ code to define the circuit and cannot directly read R1CS files compiled by Circom, we write a simple adapter for that.
- Plonk [15], another popular pairing-based but *universal setup* zk-SNARK. We also use the implementation in snarkjs. Plonk uses a constraint system other than R1CS

²The 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.

³Constructions for deriving public keys in other widely used cryptosystems like RSA and Ed25519 also exist [49], [50], but are not selected in evaluation because of the huge R1CS constraint number over 500000. In addition, none of them are optimized for the Plonkish constraint system. We will mention the difference between the two constraint systems later.

TABLE IEVALUATION RESULTS OF GROTH16.

Number of R1CS constraints in \widetilde{C}_{P}	Prover time
216 Without DV	2.470
2 ⁻⁵ without Dv	2.478
2^{16} With DV	6.538 +164.6%
2^{17} Without DV	3.45s
2^{17} With DV	7.71s +123.4%
2^{18} Without DV	5.83s
2^{18} With DV	11.20s +92.2%
2^{19} Without DV	12.90s
2^{19} With DV	17.60s +36.4%
2^{20} Without DV	25.96s
2^{20} With DV	30.088 +15.9%

called Plonkish for circuit arithmetization, but it is possible to convert R1CS instances into Plonkish instances and snarkjs has implemented this function. So we suppose our construction can also be applied to its successors like FFlonk [53] ⁴ and HyperPlonk [22] ⁵ as well.

The evaluations on the two R1CS-based zk-SNARKs are run with the same R1CS instances and inputs for \widetilde{C}_P over the BLS12-381 prime field. The evaluation on Plonk uses a set of similar \widetilde{C}_P with different sizes. 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 evaluations are run on an Arch Linux virtual machine with 16 Intel Xeon w5-2465X CPU cores and 64 GB memory assigned.

1) Evaluations on Groth16: 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 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 evaluation, the selected \widetilde{C}_V with $2^{16.54}$ R1CS constraints will introduce a fixed 4-5s overhead to the prover time. Theoretically, 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.

2) Evaluations on Aurora: 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

TABLE IIEvaluation results of Aurora.

Number of R1CS constraints in \widetilde{C}_P	Proof size	Prover time	Verifier time
2 ¹⁶ Without DV	132 KB	25.80s	0.57s
2^{16} With DV	156 KB +17.9%	107.01s +314.7%	2.41s +319.7%
2^{17} Without DV	143 KB	53.42s	1.17s
2^{17} With DV	156 KB +9.1%	114.00s +113.4%	2.478 +111.3%
2 ¹⁸ Without DV	156 KB	117.13s	2.54s
2^{18} With DV	167 KB +7.4%	227.88s +94.6%	4.61s +81.8%
2^{19} Without DV	167 KB	240.11s	5.21s
2^{19} With DV	181 KB +8.4%	461.37s +92.1%	9.47s +81.7%
2^{20} Without DV	181 KB	509.33s	10.31s
2^{20} With DV	205 KB +13.2%	1010.45s +98.4%	19.688 +91.0%

TABLE III Evaluation results of Plonk.

Number of Plonkish constraints in $\widetilde{C_P}$	Prover time
2 ¹⁷ Without DV	157.03s
2^{17} With DV	1302.388 +729.4%
2 ¹⁸ Without DV	160.58s
2^{18} With DV	1340.41s +734.7%
2 ¹⁹ Without DV	372.62s
2^{19} With DV	1567.70s +320.7%
2 ²⁰ Without DV	913.81s
2^{20} With DV	2120.598 +132.1%

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.

3) Evaluations on Plonk: Things become a bit different when it comes to Plonk and the Plonkish arithmetization. As mentioned earlier, Plonkish is a different constraint system than R1CS. A Plonkish constraint is represented by the equation

$$q_L \cdot w_a + q_R \cdot w_b + q_O \cdot w_c + q_M \cdot w_a w_b + q_C = 0, \quad (13)$$

where $q_L, q_R, q_O, q_M, q_C \in \mathbb{F}$ are called "selectors" and w_a, w_b, w_c denote two inputs and the output of a gate. By setting different values of these "selectors", such a constraint can represent a single addition or multiplication gate, or constrain the value of the circuit's final output (see section 6 in the Plonk paper for more details). The above equation can also be extended with additional selectors and witness inputs to represent any custom gate that can be expressed as a polynomial of witnesses, such as the lookup gate purposed in [54]. One of the main differences between the two constraint systems is the number of constraints when representing the same circuit. Addition gates and multiplication gates with a scalar input are free in R1CS, but in Plonkish each gate requires a separate constraint. As a result, the same circuit might require several times more constraints to be represented in Plonkish than in R1CS. The Plonk paper [15] points

⁴Snarkjs also contains an implementation of FFlonk. However, we did not evaluate it because it trades a $3 \times$ slower prover for a $3 \times$ faster verifier.

⁵The code of HyperPlonk is available for now. Currently, it can only be used with randomly generated mock circuits or hand-written circuits. However, the author indicates that integration with Circom and other frontends may be added in the future.

out that the number of addition gates is $2\times$ the number of multiplication gates in common circuits.

The evaluations on Plonk take a much longer time than the two R1CS-based zk-SNARKs. We believe this is because the evaluation toolchain is not well-optimized for Plonkish. The implementation utilizes only one CPU core, and a significant amount of time is taken by the prover to read and parse the huge outputs of the setup phase (up to 13 GB in our evaluation). We run evaluations on C_P with Plonkish constraint numbers from 2^{17} to 2^{20} and 2^{21} is not tested because of out of memory. The proof size and verifier time are constant in Plonk, so we only compare the prover time here. Similar to Groth16, the prover time is linear to the number of addition and multiplication gates (or the number of Plonkish constraints), and the selected $\widetilde{C_V}$ introduces $2^{20.61}$ Plonkish constraints and 1200s additional prover time, which is more inefficient compared with R1CS settings and might be an overkill for a small $\overline{C_P}$. In this case, it may be helpful to choose another C_V which is optimized for Plonkish, utilizes the custom gate feature and contains fewer constraints. Nevertheless, this evaluation shows that our construction can also be applied well to Plonkish-based zk-SNARKs.

C. Comparison with Designated-Verifier zk-SNARKs Constructed with Bitansky's Compiler

In this subsection, we compare our Designated-Verifier zk-SNARK construction with Bitansky's Compiler from the point of view of the amount of computation and size instead of running time.

We only take Groth16 and Elgamal encryption as an example here. This is because, as mentioned earlier, Bitansky's Compiler is not "compatible" with Plonk, Aurora and other non-LIP-based zk-SNARKs. For an arithmetic circuit \widetilde{C}_P that the prover wants to prove its satisfiability with m_P constraints (multiplication gates) and n wires (contains l public input and output), the amount of computation and size⁶ are as follows [6]:

- CRS size: $2m_P + n\mathbb{G}_1, m_P\mathbb{G}_2$
- Proof size: 2G₁, 1G₂
- Prover computation: $3m_P + n lE_1, m_PE_2$
- Verifier computation: $lE_1, 3P$

Groth16 maps the prover's messages *a* computed by some linear functions to $g^a \in \mathbb{G}_i$ where *g* is the primitive element of \mathbb{G}_i . Therefore, in the case of constructing a Designated-Verifier variant of Groth16 with encryption, it would be more convenient to encrypt the CRS with Elgamal encryption because of its multiplicative homomorphism property (or linear homomorphism property on exponents) since applying linear operations on exponents is equivalent to multiplying between different powers of the same base. Encrypting a group element plaintext requires two exponentiations, the ciphertext is two group elements and decryption requires one exponentiation. In Setup, each element of the CRS to be used in Prove needs to be encrypted, other elements that are only used in Verify can be treated as secret verification states. The verifier also needs to decrypt each element of the proof at the beginning of Verify, but in Groth16 the decryption only has to be done for a constant number of times. As a result, the amount of computation and size of this variant are as follows:

- CRS size: $4m_P + 2n l\mathbb{G}_1, 2m_P\mathbb{G}_2$
- Proof size: $4\mathbb{G}_1, 2\mathbb{G}_2$
- Prover computation: $3m_P + n lE_1, m_P E_2$ (Unchanged)
- Verifier computation: lE_1 , 3P (Unchanged)

Using our construction, assuming that the circuit $\widetilde{C_V}$ used to validate the identity of the verifier has m_V constraints (multiplication gates) and the same input and output size n(and l) as C_P , the amount of computation and size of the Designated-Verifier zk-SNARK for the composed circuit are as follows:

- CRS size: $2m_P + 2m_V + n\mathbb{G}_1, m_P + m_V\mathbb{G}_2$
- Proof size: $2\mathbb{G}_1, 1\mathbb{G}_2$ (Unchanged)
- Prover computation: $3m_P + 3m_V + n lE_1, m_P + m_V E_2$
- Verifier computation: lE_1 , 3P (Unchanged)

This also implies that our construction trades computational effort for stronger security notions of Designated-Verifier zk-SNARKs.

VI. CONCLUSION

In this paper, 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

- J. Weng, J. Weng, G. Tang, A. Yang, M. Li, and J.-N. Liu, "pvCNN: Privacy-preserving and verifiable convolutional neural network testing," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2218–2233, 2023.
- [2] T. Zhaolu, Z. Wan, and H. Wang, "Division of regulatory power: Collaborative regulation for privacy-preserving blockchains," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2533–2548, 2024.

⁶Notation: \mathbb{G}_i and E_i denotes group element and exponentiation in this group, P denotes Pairing. A bilinear pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is used in Groth16.

- [3] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in 2014 IEEE Symposium on Security and Privacy, 2014, pp. 459–474.
- [4] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in Advances in Cryptology – EUROCRYPT 2013, T. Johansson and P. Q. Nguyen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 626–645.
- [5] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," *Commun. ACM*, vol. 59, no. 2, pp. 103–112, jan 2016.
- [6] J. Groth, "On the size of pairing-based non-interactive arguments," in Advances in Cryptology – EUROCRYPT 2016, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326.
- [7] Y. Ishai, E. Kushilevitz, and R. Ostrovsky, "Efficient arguments without short PCPs," in *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, 2007, pp. 278–291.
- [8] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography*, A. Sahai, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 315–333.
- [9] I. Giacomelli, J. Madsen, and C. Orlandi, "ZKBoo: Faster Zero-Knowledge for boolean circuits," in 25th USENIX Security Symposium (USENIX Security 16). Austin, TX: USENIX Association, Aug. 2016, pp. 1069–1083.
- [10] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1825–1842.
- [11] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *Proceed*ings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 2087–2104.
- [12] R. Bhadauria, Z. Fang, C. Hazay, M. Venkitasubramaniam, T. Xie, and Y. Zhang, "Ligero++: A new optimized sublinear IOP," in *Proceedings* of the 2020 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 2025–2038.
- [13] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multiparty computation," in *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 21– 30.
- [14] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Advances* in Cryptology – EUROCRYPT 2020, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 738–768.
- [15] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953
- [16] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for R1CS," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, pp. 103–128.
- [17] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," in Advances in Cryptology – EUROCRYPT 2020, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 769–793.
- [18] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in Advances in Cryptology – CRYPTO 2020, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 704–737.
- [19] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-time and field-agnostic SNARKs for R1CS," in *Advances in Cryptology – CRYPTO 2023*, H. Handschuh and A. Lysyanskaya, Eds. Cham: Springer Nature Switzerland, 2023, pp. 193–226.
- [20] T. Xie, Y. Zhang, and D. Song, "Orion: Zero knowledge proof withǎlinear prover time," in *Advances in Cryptology – CRYPTO 2022*, Y. Dodis and T. Shrimpton, Eds. Cham: Springer Nature Switzerland, 2022, pp. 299–328.

- [21] A. Belling and A. Soleimanian, "Vortex : Building a lattice-based SNARK scheme with transparent setup," *IACR Cryptol. ePrint Arch.*, p. 1633, 2022. [Online]. Available: https://eprint.iacr.org/2022/1633
- [22] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "Hyperplonk: Plonk withălinear-time prover andăhigh-degree custom gates," in *Advances in Cryptology – EUROCRYPT 2023*, C. Hazay and M. Stam, Eds. Cham: Springer Nature Switzerland, 2023, pp. 499–530.
- [23] M. Campanelli and H. Khoshakhlagh, "Succinct publicly-certifiable proofs," in *Progress in Cryptology – INDOCRYPT 2021*, A. Adhikari, R. Küsters, and B. Preneel, Eds. Cham: Springer International Publishing, 2021, pp. 607–631.
- [24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in Advances in Cryptology — EUROCRYPT '99, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.
- [25] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.
- [26] J. Benaloh, "Dense probabilistic encryption," in Proceedings of the workshop on selected areas of cryptography, 1994, pp. 120–128.
- [27] D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu, "Lattice-based SNARGs and their application to more efficient obfuscation," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds. Cham: Springer International Publishing, 2017, pp. 247–277.
- [28] —, "Quasi-optimal SNARGs via linear multi-prover interactive proofs," in Advances in Cryptology – EUROCRYPT 2018, J. B. Nielsen and V. Rijmen, Eds. Cham: Springer International Publishing, 2018, pp. 222–255.
- [29] R. Gennaro, M. Minelli, A. Nitulescu, and M. Orrù, "Lattice-based zk-SNARKs from square span programs," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 556–573.
- [30] Y. Ishai, H. Su, and D. J. Wu, "Shorter and faster post-quantum designated-verifier zkSNARKs from lattices," in *Proceedings of the 2021* ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 212–234.
- [31] X. Zhu, X. Song, and Y. Deng, "Fast and designated-verifier friendly zkSNARKs in the BPK model," Cryptology ePrint Archive, Paper 2023/1806, 2023. [Online]. Available: https://eprint.iacr.org/2023/1806
- [32] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in 2015 IEEE Symposium on Security and Privacy, 2015, pp. 287–304.
- [33] S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-snark parameters in the random beacon model," Cryptology ePrint Archive, Paper 2017/1050, 2017. [Online]. Available: https: //eprint.iacr.org/2017/1050
- [34] S. Bowe, A. Gabizon, and M. D. Green, "A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK," in *Financial Cryptography and Data Security*, A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 64–77.
- [35] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Advances in Cryptology – CRYPTO* 2019, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 701–732.
- [36] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doublyefficient zkSNARKs without trusted setup," in 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 926–943.
- [37] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Paper 2018/046, 2018. [Online]. Available: https://eprint.iacr.org/2018/046
- [38] D. Chaum, "Private signature and proof systems, US Patent 5,493,614," Patent US5 493 614A, 1996.
- [39] M. Jakobsson, K. Sako, and R. Impagliazzo, "Designated verifier proofs and their applications," in *Advances in Cryptology – EUROCRYPT '96*, U. Maurer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 143–154.
- [40] S. Saeednia, S. Kremer, and O. Markowitch, "An efficient strong designated verifier signature scheme," in *Information Security and Cryptology* - *ICISC 2003*, J.-I. Lim and D.-H. Lee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 40–54.
- [41] M. H. Au and W. Susilo, "Two-party (blind) ring signatures and their applications," in *Information Security Practice and Experience*, X. Huang and J. Zhou, Eds. Cham: Springer International Publishing, 2014, pp. 403–417.

- [42] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zeroknowledge applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 4733–4751, 2023.
- [43] S. Setty, J. Thaler, and R. Wahby, "Customizable constraint systems for succinct arguments," Cryptology ePrint Archive, Paper 2023/552, 2023. [Online]. Available: https://eprint.iacr.org/2023/552
- [44] A. Nitulescu, "zk-SNARKs: a gentle introduction," 2020. [Online]. Available: https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf
- [45] F.-Y. Yang and C.-M. Liao, "A provably secure and efficient strong designated verifier signature scheme," in *International Journal of Network Security*, vol. 10, no. 3, 2010, pp. 220–224.
- [46] B. Gong, M. H. Au, and H. Xue, "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), 2019, pp. 586–593.
- [47] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking Proof-Based verified computation a few steps closer to practicality," in 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX Association, 2012, pp. 253–268.
- [48] 0xPARC, "circom-ecdsa: Big integer arithmetic and secp256k1 ECC operations in circom," 2024. [Online]. Available: https://github.com/ 0xPARC/circom-ecdsa
- [49] zkp application, "circom-rsa-verify: Zero knowledge proof for RSA," 2024. [Online]. Available: https://github.com/zkp-application/ circom-rsa-verify
- [50] Electron-Labs, "ed25519-circom: Ed25519 implementation in circom," 2024. [Online]. Available: https://github.com/Electron-Labs/ ed25519-circom
- [51] iden3, "snarkjs: zkSNARK implementation in JavaScript & WASM," 2024. [Online]. Available: https://github.com/iden3/snarkjs
- [52] scipr lab, "libiop: C++ library for IOP-based zkSNARKs," 2024. [Online]. Available: https://github.com/scipr-lab/libiop
- [53] A. Gabizon and Z. J. Williamson, "fflonk: a fast-fourier inspired verifier efficient version of PlonK," Cryptology ePrint Archive, Paper 2021/1167, 2021. [Online]. Available: https://eprint.iacr.org/2021/1167
- [54] L. Pearson, J. Fitzgerald, H. Masip, M. Bellés-Muñoz, and J. L. Muñoz-Tapia, "PlonKup: Reconciling PlonK with plookup," Cryptology ePrint Archive, Paper 2022/086, 2022. [Online]. Available: https: //eprint.iacr.org/2022/086



Chen Li received the bachelor's degree in cyberspace security from Jinan University, China in 2022. He is currently pursuing the Ph.D. degree with School of Computer Science and Engineering of Sun Yat-sen University, China. His research interests include zero-knowledge proofs.



Fangguo Zhang received his PhD from the School of Communication Engineering, Xidian University in 2001. He is currently a Professor at School of Computer Science and Engineering of Sun Yat-sen University, China and the director of Guangdong Key Laboratory of Information Security Technology. His research mainly focuses on cryptography and its applications. Specific interests are elliptic curve cryptography, post-quantum public key cryptosystem and blockchain.