Technical Note

A Privacy-preserving Outsourcing Scheme for Zero-knowledge Proof Generation

Makoto Nakamura[1,a)] Takeshi Miyamae[1,b)] Masanobu Morinaga[1,c)]

Received: August 17, 2021, Accepted: October 8, 2021

Abstract: We propose a privacy-preserving scheme to outsource zero-knowledge proof generation to a party that we call a worker. Our scheme can be applied to zk-SNARKs with a trusted setup, zero-knowledge proofs deployed in many applications. Compared to known privacy-preserving outsourcing schemes, our scheme is more practical in the sense that the computational and memory load on the worker is almost the same as that on the prover in cases where the provers generate proofs on their own.

Keywords: privacy-preserving outsourcing, zero-knowledge proofs, zk-SNARK

1. Introduction

1.1 Background

A zero-knowledge proof (often abbreviated to ZKP) is a cryptographic protocol, which enables a prover to convince a verifier that a proposition is true without disclosing the prover's secret data for the proposition. For instance, one can use a ZKP to convince another party that they *know* a correct input of a hash function without revealing the input itself. ZKPs have attracted much attention as privacy-enhancing technology since Goldwasser et al. first introduced the concept [1].

Today there are many practical ZKP protocols (e.g., Refs. [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], and [13]) and their applications in a variety of areas. The areas include verifiable computations [3], DNA profiling [14], anonymous credentials [15], image authentication [16], e-voting [17], supply chain tracing [18], EV charging scheduling [19] and COVID-19 contact tracing [20]. Furthermore, there are already real-world use cases, in particular, in the blockchain area. For example, the cryptocurrencies Zcash [21] and Monero [22] use ZKPs to hide transaction data.

Although ZKPs are useful privacy-enhancing tools and many applications are proposed, there are still bottlenecks to deploying ZKPs in more real-world use cases. In ZKPs, a prover generates a *proof*, a datum ensuring the correctness of a proposition. The verifier checks the proof. If the proof is valid, the verifier can be convinced that the proposition in question is true. An issue common to all ZKPs is that proof generation requires a high computational cost. It takes 6 minutes to generate a proof even when using one of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) that are the most efficient type of ZKP as seen in Ref. [23] [*1]. The proof generation procedure is also a memory hog [24]. The high computational and memory costs for proof generation are an obstacle to applying ZKPs in cases in which the user has only limited computing resources such as a smartphone.

One way to reduce a prover's task is to outsource the proof generation procedure entirely or partly to other parties, each of which we call a *worker*. For example, one can use a distributed zk-SNARK system DIZK [24] to allow proof generation by cluster computing, to outsource proof generation partly to workers. A prover has to trust workers in such a scheme since proof generation requires the prover's secret data. Hence the prover must reveal it to the workers if the prover executes no additional procedure on the secret data.

The secret data may include sensitive information. A (practical) privacy-preserving outsourcing scheme is therefore needed.

1.2 Our Contribution

In this paper, we propose a privacy-preserving outsourcing scheme for proof generation. The scheme can be applied to zk-SNARKs with a trusted setup, which are deployed in many applications.

In our scheme, a prover sends their secret data to a worker *after encryption* according to random parameters that the prover picks. The secret data are secure unless the random parameters are known to a worker. The scheme is practical in the following sense:

- The prover can outsource the proof generation procedure entirely to the worker.
- The worker's computational and memory cost is almost the same as the prover when the prover generates a proof on his

[1] Data and Security Research Laboratory, Fujitsu Limited, Kawasaki, Kanagawa 211–0053, Japan
[a)] nakamura5830@fujitsu.com
[b)] miyamae.takeshi@fujitsu.com
[c)] morinaga@fujitsu.com

[*1] The measurement in Ref. [23] uses a 70 GB RAM and a CPU with 3 GHz virtual core. We note that the proof generation time depends on a proposition. The measurement uses a proposition concerned with a Merkle hash tree with 256 leaves.

own.

2. Related Works

Trinocchio [25] is a scheme based on a zk-SNARK Pinocchio [3]. In the scheme, a prover outsources the proof generation procedure to several workers. The prover splits its secret data into multiple parts via Shamir's secret sharing, and sends each part to each worker. Every worker performs some computation on the part and sends the result back to the prover. Then the prover integrates the results to a proof. The computational cost of the integration procedure is less than that for proof generation.

Rahimi et al. [26] proposed a secure outsourcing scheme using a secret sharing procedure somewhat different from Shamir's. The scheme of Rahimi et al. is more efficient than Trinocchio in terms of the computational load of each worker.

In their schemes, the secret data are secure unless some of the workers colludes with each other. Trinocchio for example requires at least three workers assuming that one worker is not trustworthy.

3. Preliminaries

This section briefly reviews zk-SNARKs and how to use them. For theoretical descriptions of zk-SNARKs, we recommend the reader to refer to literature such as Refs. [3], [4], and [8].

3.1 zk-SNARKs

Let $\mathbb{F}$ be a finite field. In a typical zk-SNARK, a proposition to prove is as follows: a prover knows some vector $w \in \mathbb{F}^k$ so that $(w, p) \in \mathbb{F}^k \times \mathbb{F}^l$ becomes a solution of a system of equations of some type supported by the zk-SNARK. Such a system of equations is called a *constraint system*.

At present, zk-SNARKs are the most efficient type of ZKPs in terms of proof size and verification time. In most zk-SNARKs, the proof size is small and constant (i.e., independent of a proposition), and a verifier can check a proof in time linear to the size of the public data. Another advantage is non-interactiveness; namely, a prover needs to communicate with a verifier only once. Non-interactiveness is preferable for real-world use since many interactions between the prover and the verifier might cause network latency issues. Due to such efficiency, zk-SNARKs are adopted in the cryptocurrency Zcash [21] and the blockchain Layer2 technology zkSync [27].

To achieve efficiency, typical zk-SNARKs require an additional setup phase. In the setup, a third party calculates a *common reference string* (CRS) and provides it to both a prover and a verifier before proof generation. While a CRS has the effect of reducing computational loads on a prover and a verifier, it also causes the following problems:

- In CRS generation, the third party obtains a trapdoor, which enables one to create a counterfeit proof that passes the verifier's inspection.
- A CRS depends on a proposition.

Therefore (in typical zk-SNARKs), participants must trust the third party, and a CRS must be calculated per proposition. In Zcash, disclosing a trapdoor may cause currency counterfeiting. Hence Zcash uses a multi-party computation protocol in CRS generation so only one participant must be honest in the protocol.

A proof in a typical zk-SNARK with a trusted setup is generated from a solution (w, p) of the constraint system in question and a CRS for it. The proof is verified using the public vector p and the CRS. It is difficult to generate a valid proof unless the solution (w, p) is known. Hence the verifier can be convinced that the proposition is true if the proof is valid.

We remark that recently zk-SNARKs have been proposed to deal with CRS issues. For example, Groth et al. [8] constructed a zk-SNARK with universal and updatable CRS. Here the term "universal" means that a single CRS can be used for all constraint systems of some bounded size, and the term "updatable" means that anyone can update the CRS. Their zk-SNARK was improved in Refs. [10], [11], and [13].

3.2 How to Use zk-SNARKs

A zk-SNARK enables a prover to convince a verifier that the prover knows correct inputs and outputs of a vector-valued function $F : \mathbb{F}^{d_1} \to \mathbb{F}^{d_2}$ without revealing parts of inputs and outputs. The scheme based on a zk-SNARK with a trusted setup is as follows:

(1) First of all, to use a zk-SNARK, one must convert the function F into a constraint system C supported by the zk-SNARK. The constraint system C is designed so that a solution for it can be derived from correct inputs and outputs of F.
(2) A trusted third party (TTP) calculates a CRS for a proposition concerned with C and sends it to the prover and the verifier.
(3) The prover generates a proof for knowledge of a solution of C, and sends the proof to the verifier.
(4) Finally, the verifier checks the proof.

To be familiar with the scheme, we explain how to convert F into a rank-1 constraint system (R1CS), which is supported by many zk-SNARKs such as Refs. [2], [3] and [4]. An R1CS is a system of equations of the form,

$$\langle x, u\rangle = \langle x, v\rangle\langle x, w\rangle,$$

for some $u, v, w \in \mathbb{F}^{m+1}$, where $x = (x_0, x_1, \cdots, x_m)$ is a variable with the assignment $x_0 \equiv 1$, and $\langle , \rangle$ denotes the usual inner product:

$$\langle a, b\rangle := \sum_{i=0}^{m} a_i b_i, \quad a = (a_0, \ldots, a_m), b = (b_0, \ldots, b_m).$$

It is well-known that each component F_j of $F = (F_1, \ldots, F_{d_2})$ can be written as a polynomial function since $\mathbb{F}$ is a finite field. Therefore F can be represented as an arithmetic circuit consisting of addition, addition-by-scalar, multiplication and multiplication-by-scalar gates and wires connected to gates. In particular, we can assume that every multiplication gate has a fan-in of 2. Each gate of the arithmetic circuit constrains values carried by wires connected to the gate; for example, if the input wires connected to an addition gate carry $a_1, a_2, a_3 \in \mathbb{F}$, respectively, then the output wire has to carry $a_4 \in \mathbb{F}$ such that $a_4 = a_1 + a_2 + a_3$. From this point of view, we interpret the circuit as an R1CS since every multiplication gate has a fan-in of 2; see also **Fig. 1**. One can derive a

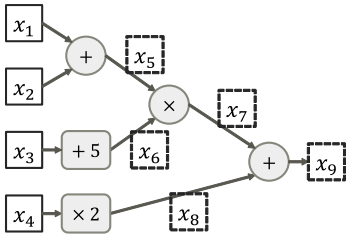

Fig. 1 The function $F(x_1, x_2, x_3, x_4) = x_1x_3 + x_2x_3 + 5x_1 + 5x_2 + 2x_4$ can be converted into an R1CS consisting of 5 equations $x_1 + x_2 = x_5$, $x_3 + 5 = x_6$, $x_5x_6 = x_7$, $2x_4 = x_8$, and $x_7 + x_8 = x_9$.

solution of the R1CS from correct inputs and outputs of F. There are infinitely many R1CSs that represent the same function.

4. Our Scheme

We propose a privacy-preserving scheme to outsource proof generation in a zk-SNARK with a trusted setup to a worker. As the basic scheme explained in Section 3.2, our scheme allows a prover to convince a verifier that the prover knows correct inputs and outputs of a vector-valued function $F : \mathbb{F}^{d_1} \to \mathbb{F}^{d_2}$ without revealing parts of inputs and outputs.

4.1 Overview of the Scheme

Without loss of generality, we can assume that the prover wants to hide the first k_1 inputs $s_1 \in \mathbb{F}^{k_1}$ and the first k_2 outputs $s_2 \in \mathbb{F}^{k_2}$. The rest of the inputs and outputs are disclosed to the worker, the verifier, and a TTP. We denote by $p_1 \in \mathbb{F}^{l_1}$, $p_2 \in \mathbb{F}^{l_2}$ the public inputs and outputs, respectively, where $l_i := d_i - k_i$ $(i = 1, 2)$.

The prover's secret data s_1, s_2 are sent to the worker after being encrypted by functions $E_1 : \mathbb{F}^{k_1} \to \mathbb{F}^{k_1}$, $E_2 : \mathbb{F}^{k_2} \to \mathbb{F}^{k_2}$ the prover chooses. Here E_i $(i = 1, 2)$ is invertible; namely there is a function $D_i : \mathbb{F}^{k_i} \to \mathbb{F}^{k_i}$ so that

$$D_i \circ E_i = 1_{\mathbb{F}^{k_i}} = E_i \circ D_i,$$

where 1_S denotes the identity map of a set S. We denote by $\hat{s}_1$, $\hat{s}_2$ the secret inputs and outputs, respectively. More specifically, these are given by $\hat{s}_1 = E_1(s_1)$, $\hat{s}_2 = E_2(s_2)$.

The TTP is informed of encryption functions E_1, E_2. Another function $\hat{F} : \mathbb{F}^{d_1} \to \mathbb{F}^{d_2}$ is then converted into a constraint system $\hat{C}$. The function $\hat{F}$ is defined by

$$\hat{F} := (E_2 \times 1_{\mathbb{F}^{l_1}}) \circ F \circ (D_1 \times 1_{\mathbb{F}^{l_2}}). \tag{1}$$

Since $\hat{F}(\hat{s}_1, p_1) = (E_2 \times 1_{\mathbb{F}^{l_1}}) \circ F(s_1, p_1) = (\hat{s}_2, p_2)$, the data $\hat{s}_1$, p_1, $\hat{s}_2$, and p_2 are correct inputs and outputs of $\hat{F}$ (**Fig. 2**). The TTP also calculates a CRS for a proposition for $\hat{C}$; the CRS is sent to the worker and the verifier.

The worker derives a solution of the constraint system $\hat{C}$ from $\hat{s}_1$, $\hat{s}_2$, p_1, and p_2. Then the worker generates a proof $\hat{\pi}$ for $\hat{C}$.

The verifier checks the proof $\hat{\pi}$. It is difficult to generate a valid proof unless one has knowledge of correct inputs and outputs of $\hat{F}$, which is equivalent to F's. Hence the verifier is convinced that the prover knows correct inputs and outputs of F if the proof $\hat{\pi}$ is valid.

The larger size of a constraint system increases the computational cost for proof generation in typical zk-SNARKs. Therefore, for the sake of efficiency, we use simple encryption functions $E_i(x) := x + r_i$ $(i = 1, 2)$ with $r_i \in \mathbb{F}^{k_i}$. This being the case,

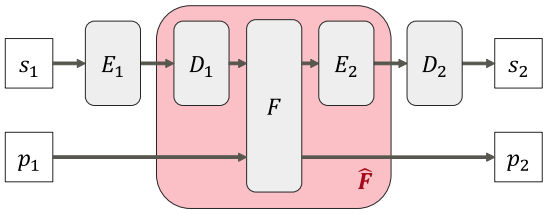

Fig. 2 The encrypted secret datum $\hat{s}_1$ are on the input wire to D_1 and $\hat{s}_2$ is on the output wires from E_2. Hence we see that $(\hat{s}_1, p_1)$ and $(\hat{s}_2, p_2)$ are correct inputs and outputs of $\hat{F}$, respectively.

the decryption function D_i is given by $x \mapsto x - r_i$.

4.2 Protocol

The protocol of our scheme is as follows:

Step 1 The prover picks random parameters r_1 and r_2 from $\mathbb{F}^{k_1}$ and $\mathbb{F}^{k_2}$, respectively, and sends them to the TTP.

Step 2 The prover calculates $\hat{s}_i \leftarrow E_i(s_i) := s_i + r_i$ $(i = 1, 2)$ and sends $\hat{s}_1$, $\hat{s}_2$ to the worker.

Step 3 The TTP converts the function $\hat{F}$ into a constraint system $\hat{C}$, and calculates a CRS for $\hat{C}$. Then the TTP informs $\hat{C}$ to the worker and sends the CRS to the worker and the verifier.

Step 4 The worker generates a proof $\hat{\pi}$ concerned with $\hat{C}$. Then the worker sends the proof $\hat{\pi}$ to the verifier.

Step 5 The verifier checks the proof $\hat{\pi}$.

4.3 Security

The prover's secret data s_1, s_2 can be recovered from $\hat{s}_1$, $\hat{s}_2$ using the random parameters as $s_i \leftarrow D_i(\hat{s}_i) = \hat{s}_i - r_i$ $(i = 1, 2)$. The secret data is therefore disclosed when a worker obtains the random parameters. If the worker knows F, the worker may infer the random parameters from $\hat{C}$. Therefore, to avoid disclosing the secret data to the worker, a better approach is to slightly adjust the scheme so that the worker communicates only with the prover.

4.4 Remarks on Implementation

In Step 3, the TTP calculates a CRS. The CRS generation procedure requires a high computational cost. Hence when applying our scheme to a case such as many users have to convince some party that they know correct inputs and outputs of the same function F, it is more practical for the TTP to prepare a number of tuples consisting of random parameters, a constraint system $\hat{C}$ for the function $\hat{F}$ determined by the random parameters, and a CRS for $\hat{C}$ in advance; each user then selects the random parameters.

4.5 Comparison with the Known Schemes

We compare our scheme with the other privacy-outsourcing schemes referred to in Section 2.

In our scheme, the secrets are disclosed when both the TTP and the worker are attacked. On the other hand, in Trinocchio [25] and the scheme of Rahami et al. [26], the secret data are disclosed when some of the workers compromise the information.

In terms of the computational and memory costs on the workers in total, our scheme is more efficient than Trinocchio and the scheme of Rahimi et al. In fact, Trinocchio requires each worker to compute the same load or more than a prover in Pinocchio.

The scheme of Rahimi et al. is more efficient than Trinicchio; however, to reduce each worker's task, it needs more and more trusted workers. On the other hand, in our scheme, the worker's computational and memory cost is almost the same as that on the prover if the provers generate proofs on their own.

We also note that our scheme works in a single-worker setting, while Trinocchio and the scheme of Rahimi et al. do not work in that setting.

5. Conclusions

We propose a privacy-preserving outsourcing scheme for zero-knowledge proof generation, which is an overhead for ZKPs to be applied in more real-world use cases assuming that a prover has only limited computational resources.

Our scheme can be applied to zk-SNARKs with a trusted setup. Compared to the known outsourcing schemes in the literature, our scheme is more practical in that the computational and memory load on the worker is almost the same as that on the prover in cases where the provers generate proofs on their own.

References

[1] Goldwasser, S., Micali, S. and Rackoff, C.: The knowledge complexity of interactive proof systems, *SIAM Journal on Computing*, Vol.18, No.1, pp.186–208 (1989).

[2] Gennaro, R., Gentry, C., Parno, B. and Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp.626–645, Springer (2013).

[3] Parno, B., Howell, J., Gentry, C. and Raykova, M.: Pinocchio: Nearly practical verifiable computation, *2013 IEEE Symposium on Security and Privacy*, pp.238–252, IEEE (2013).

[4] Groth, J.: On the size of pairing-based non-interactive arguments, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp.305–326, Springer (2016).

[5] Ben-Sasson, E., Bentov, I., Horesh, Y. and Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity, *IACR Cryptol. ePrint Arch.*, Vol.2018, p.46 (2018).

[6] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P. and Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more, *2018 IEEE Symposium on Security and Privacy*, pp.315–334, IEEE (2018).

[7] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J. and Walfish, M.: Doubly-efficient zkSNARKs without trusted setup, *2018 IEEE Symposium on Security and Privacy*, pp.926–943, IEEE (2018).

[8] Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S. and Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs, *Annual International Cryptology Conference*, pp.698–728, Springer (2018).

[9] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M. and Ward, N.P.: Aurora: Transparent succinct arguments for R1CS, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp.103–128, Springer (2019).

[10] Maller, M., Bowe, S., Kohlweiss, M. and Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings, *Proc. 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp.2111–2128 (2019).

[11] Gabizon, A., Williamson, Z.J. and Ciobotaru, O.: PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge, *IACR Cryptol. ePrint Arch.*, Vol.2019, p.953 (2019).

[12] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup, *Annual International Cryptology Conference*, pp.704–737, Springer (2020).

[13] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N. and Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp.738–768, Springer (2020).

[14] Fisch, B., Freund, D. and Naor, M.: Physical zero-knowledge proofs of physical properties, *Annual Cryptology Conference*, pp.313–336, Springer (2014).

[15] Garman, C., Green, M. and Miers, I.: Decentralized Anonymous Credentials, *NDSS* (2014).

[16] Naveh, A. and Tromer, E.: Photoproof: Cryptographic image authentication for any set of permissible transformations, *2016 IEEE Symposium on Security and Privacy*, pp.255–271, IEEE (2016).

[17] Panja, S. and Roy, B.K.: A secure end-to-end verifiable e-voting system using zero knowledge based blockchain, *IACR Cryptol. ePrint Arch.*, Vol.2018, p.466 (2018).

[18] Sahai, S., Singh, N. and Dayama, P.: Enabling privacy and traceability in supply chains using blockchain and zero knowledge proofs, *2020 IEEE International Conference on Blockchain*, pp.134–143, IEEE (2020).

[19] Gabay, D., Akkaya, K. and Cebe, M.: Privacy-preserving authentication scheme for connected electric vehicles using blockchain and zero knowledge proofs, *IEEE Trans. Vehicular Technology*, Vol.69, No.6, pp.5760–5772 (2020).

[20] Liu, J.K., Au, M.H., Yuen, T.H., Zuo, C., Wang, J., Sakzad, A., Luo, X. and Li, L.: Privacy-preserving COVID-19 contact tracing App: A zero-knowledge proof approach, *IACR Cryptol. ePrint Arch.*, Vol.2020, p.528 (2020).

[21] Electric Coin Company: Zcash (online), available from ⟨https://z.cash/⟩ (accessed 2021-06-16).

[22] Monero: Monero (online), available from ⟨https://www.getmonero.org/⟩ (accessed 2021-06-16).

[23] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C. and Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation, *Annual International Cryptology Conference*, pp.733–764, Springer (2019).

[24] Wu, H., Zheng, W., Chiesa, A., Popa, R.A. and Stoica, I.: DIZK: A distributed zero knowledge proof system, *27th USENIX Security Symposium*, pp.675–692 (2018).

[25] Schoenmakers, B., Veeningen, M. and de Vreede, N.: Trinocchio: Privacy-friendly outsourcing by distributed verifiable computation, *IACR Cryptol. ePrint Arch.*, Vol.2015, p.480 (2015).

[26] Rahimi, A. and Maddah-Ali, M.A.: Multi-Party Proof Generation in QAP-based zk-SNARKs, *IEEE Journal on Selected Areas in Information Theory* (online), DOI: 10.1109/JSAIT.2021.3102267 (2021).

[27] Matter Labs: zkSync (online), available from ⟨https://zksync.io/⟩ (accessed 2021-06-16).

Makoto Nakamura received his B.S. and M.S. degrees in Mathematics from the University of Tokyo in 2014 and 2016, respectively. He joined Fujitsu Laboratories Limited in 2020. He is currently with Data and Security Research Laboratory, Fujitsu Limited.

Takeshi Miyamae received his B.E. degree in Engineering from the University of Tokyo in 1994. He joined Fujitsu Limited in 1994. He is currently with Data and Security Research Laboratory, Fujitsu Limited.

Masanobu Morinaga received his B.E. and M.E. degrees in Electronic Engineering from Kyushu University in 1991 and 1993, respectively. He joined Fujitsu Laboratories Limited in 1993. He is currently with Data and Security Research Laboratory, Fujitsu Limited.