QAP-based Simulation-Extractable SNARK with a Single Verification

Jihye Kim[1], Jiwon Lee[2], and Hyunok Oh[2]

[1] Kookmin University, Seoul, Korea,
jihyek@kookmin.ac.kr
[2] Hanyang University, Seoul, Korea,
{jiwonlee,hoh}@hanyang.ac.kr

Abstract. The pairing-based simulation-extractable succinct non-interactive arguments of knowledge (SE-SNARKs) are attractive since they enable a prover to generate a proof with the knowledge of the witness to an instance in a manner which is succinct - proofs are short and the verifier's computation is small, zero-knowledge - proofs do not reveal the witness, and simulation-extractable - it is only possible to prove instances to which a witness is known although a number of simulated proofs are provided. The state-of-the-art pairing-based SE-SNARK is based on a square arithmetic program (SAP), instead of a more generalized quadratic arithmetic program (QAP). In order to add simulation extractability, the SE-SNARK requires to verify an additional equation compared to the state-of-the-art SNARKs.
In this paper, we propose a QAP-based SE-SNARK which consists of only *3 group elements* for a *QAP circuit* and a *single* verification equation in asymmetric groups (Type III pairing). The proposed scheme is secure under concrete intractability assumptions in the random oracle model. Moreover, we propose a scheme with *two elements* as a proof and a single verifying equation, based on SAP in a symmetric group (Type I pairing).

Keywords: SNARK, non-interactive zero-knowledge proof, simulation-extractability, quadratic arithmetic program, square arithmetic program

1 Introduction

As digital privacy becomes more sensitive, the conflict between privacy and legitimacy often sets a barrier for recent real-life applications. One proper example is privacy-aware blockchain systems, such as an anonymous voting blockchain. Since the blockchain is well-known to provide robust integrity due to consensus and distribution, it is often considered as an ideal platform for the voting applications. The blockchain integrity provided by finalizing contents and distributing them to all participants, however, raises a privacy issue for the plain data. Alternatively, if the data (votes) is encrypted in a block, then it is hard to know whether the data provider (voter) is an authorized candidate or whether the

data itself was created in a legitimate way. The contradiction between privacy vs. legitimacy leads the privacy-aware blockchain applications to a dead-end[3].

A zero-knowledge proof system acts as a problem-solver to resolve the legitimacy problem of the private data. If the data provider includes a proof related to the legitimacy of the data, the public can verify it without knowing the data. In practice, anonymous blockchain cryptocurrencies such as Zcash [BCG+14], already deploy the zero-knowledge proof system in their applications. The main concern is practicality: the proof generation needs to be non-interactive when the applications target the unlimited, non-specific, public verification and the proof size/verification time is desired to be scalable regardless of the complexity of the legitimacy.

In the recent history of zero-knowledge proofs, zk-SNARKs (zero-knowledge succinct non-interactive arguments of knowledge) have drawn significant attention for its practicality and theoretical advances. They enable a prover to generate a proof for NP statements in a manner where the proof is zero-knowledge about its witness and the proof size and the verification cost are succinct. For succinctness, it is often accepted if the size and the verifying computation are logarithmic to the circuit size. Thus the zk-SNARK terminology embraces various types of zero-knowledge proof systems, such as ZKBoo [GMO16] and vRAM [ZGK+18] which are an advanced from of traditional interaction-based proof systems with Fiat-Shamir transformation [FS86].

However, when applied to a massive public infrastructure such as blockchain, a logarithmic (sublinear) size might not be enough for succinctness. For example in Zcash [BCG+14], the membership test circuit has 64 hash functions (approximately 29,000 lines for each hash) which leads to a single proof size of 5MB by rough estimation in ZKBoo [GMO16][4]. Considering that innumerable transactions, each including a proof are distributed to the participants, a proof size of 5MB seems inadmissible as practical.

Therefore, for scalability, it is desirable to adopt zk-SNARKs with a *constant* size proof and verification, which is constructed in the paring-based elliptic curve group and Quadratic Arithmetic Program (QAP) [GGPR13]. In the QAP-based SNARKs such as [Gro16], by utilizing polynomial relations, a proof contains 3 group elements and the verification requires 3 pairings regardless of the circuit size. When this scheme is applied to the Zcash, the proof size becomes 60 bytes and verification takes 100ms. Consequently, we focus on the literature of QAP-based (and pairing-based) zk-SNARKs with constant size proofs and verification for the rest of the paper. Hereafter, we often use the term zk-SNARK or SNARK mixed with the "QAP-based (and pairing-based) zk-SNARK".

[3] There still are alternative solutions, such as setting a trusted manager or delicately narrowing down the blockchain data contents. However it is often complicated and does not solve the fundamental controversy.

[4] In ZKBoo, the experiment results show that the proof size is 835.91KB for a SHA-256 hash function. We multiply it by 6 (=log(64)) to estimate the 64 sequential executions of hash functions.

Simulation-extractability. Despite the practical functionality, a weakness of zk-SNARKs is that they are susceptible to man-in-the-middle attacks. Namely, an adversary who obtains valid proofs could forge a new valid proof without knowing the witness. In consequence, the zk-SNARK's implementation often requires an additional protection method against its malleability. The Zcash [BCG+14], for example, combines one-time signatures within the zk-SNARK circuit.

Groth and Maller [GM17] tackle the malleability problem of existing SNARKs, define the notion of *simulation-extractable*-SNARK (SE-SNARK) which indicates non-malleable zk-SNARKs, and propose the corresponding scheme called SE-SNARK. Briefly, it is possible to generate a new valid proof by performing point exponentiation and/or multiplication operations given a state-of-the-art SNARK [Gro16] proof which consists of three group elements (A, B, C) such that $e(A, B) = e(G^{\alpha}, H^{\beta})\ e(G^{\frac{f(\phi)}{\gamma}}, H^{\gamma})\ e(C, H^{\delta})$. Note that f in the IO statement ϕ is a known polynomial and $\alpha, \beta, \gamma, \delta$ are secret values. Without knowing the witness, the proof modification is possible in two ways as follows:

$$A' = A^r; B' = B^{\frac{1}{r}}; C' = C$$
$$A' = A; B' = BH^{r\delta}; C' = A^r C$$

Note that $A = G^a$ (in group $\mathbb{G}_1$), and $B = H^b$ (in group $\mathbb{G}_2$) are associated with the left input a and right input b of a multiplication gate, respectively, and $C = G^c$ (in $\mathbb{G}_1$) is connected with the output c of the gate. Let us simplify the verification equation for a better delivery of the idea: the verification confirms the relation $a \cdot b = c$ by evaluating the pairing equation $e(A, B) = e(C, H)$. When observing these attacks, it is either driven by modifying a, b to satisfy $a \cdot r \cdot b \cdot r^{-1} = c$, or modifying b, c to satisfy $a \cdot (b + r) = c + ar$. The attacks that adversaries can deliver are to eliminate the random values r, r^{-1} by multiplication $(r \cdot r^{-1})$, or to add r to elements b and c.

The main reason which makes these attacks possible is that the proof elements (A, B, C) are related only by the algebraic structure of the pairing function. Thus to prevent these attacks, it is recommended to bind the elements to have a tighter relation. Square arithmetic program (SAP) is a circuit where an arithmetic representation is in a square format, i.e., $a * a = c$, while $a * b = c$ in QAP. SE-SNARK [GM17] resolves the malleability issue by 1) adopting SAP to synchronize the exponent of $A = G^a$ and $B = H^{a'}$, and 2) applying an additional verification to check them (i.e. $a = a'$). In this format, the above man-in-the-middle attack based on eliminations by point exponentiation or multiplication does not work anymore because the exponents of A and B should keep the same value. This is why the supported language in [GM17] is degraded into a Squaring Arithmetic Program (SAP) where the left input and the right input of a multiplication gate are identical. In order to accomplish non-malleability, [GM17] pays a price: an increase in CRS size and proof computation/verification.

SNARK in NILP. In the recent works [Gro16,GM17], the authors attuned existing SNARKs to the non-interactive linear proofs (NILP), renamed after

the original notion of linear interactive proofs (LIP) [BCI+13]. In the NILP frame, the proof matrix is a matrix that includes a witness-related vectors; the matrix is multiplied to the common reference string (CRS) when generating a proof. The proof matrix is independent of the CRS σ, and it is noted as $\Pi \leftarrow \mathsf{ProofMatrix}(R, \phi, w)$ with a function $\mathsf{ProofMatrix}$ where R is an relation and $(\phi, w) \in R$ is an instance. The proof is then computed by $\pi = \Pi\sigma$. The NILP frame efficiently covers all existing SNARKs.

It is proven that any SE-SNARK scheme in NILP necessarily requires at least 3 elements in a proof and 2 equations in the verification [GM17]. The state-of-the-art [GM17] in pairing-based SE-SNARKs achieves *3 elements* for a proof and *2 equations* for verification by using the SAP circuit.

Breaking the boundary. While 3 elements in a proof and 2 equations in verification are required at least in a pairing-based NILP SE-SNARK, they can be further optimized if the NILP is not assumed. An interesting observation is that the usage of hash functions can deviate from the NILP frame, and exploit a new possibility of breaking the existing boundaries. Given a proof tuple (A, B, C) as in [Gro16], C includes the hash of A and B in our approach, deviating from the NILP frame. In this approach, it is possible to achieve better results of 3 proof elements and a single verifying equation. Additionally, a QAP circuit is allowed in our SE-SNARK while a SAP circuit is necessary in [GM17].

Our contributions. In this paper, we first construct a QAP-based SE-SNARK with a single verifying equation in an asymmetric group (Type III pairing). Given three groups with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, our proofs consists of only 3 group elements from the source groups: two from $\mathbb{G}_1$ and one from $\mathbb{G}_2$. Additionally, we also propose a SAP-based SE-SNARK with 2 elements for a proof and a single verifying equation, in a symmetric group (Type I pairing). We summarize our contributions as follows:

- **QAP-based SE-SNARK with 3 elements (Type III)**
  We propose a first pairing-based SE-SNARK that utilizes QAP circuits, instead of SAP, while maintaining 3 elements for a proof. Note that the SAP circuit size is theoretically double of the QAP circuit size.

- **SAP-based SE-SNARK with 2 elements (Type I)**
  We show that our construction can reduce the number of proof elements to 2 (with utilizing SAP) in symmetric pairing (Type I). Note that this result surpasses the theoretical boundary for SE-SNARKs proven in [GM17].

- **Single verifying equation**
  Our SE-SNARK construction verifies the proof with a *single* verifying equation. By utilizing the hash function to bind the unique proof tuple (A, B, C), we eliminate the additional equation for the malleability check.

Table 1 summarizes and compares the overall size and performance of our QAP-based SE-SNARK with the state-of-the-art schemes of zk-SNARK [Gro16] and SE-SNARK [GM17].

Table 1: Comparison for arithmetic circuit satisfiability with l element instance, m wires, n multiplication gates. Since SE-SNARK uses squaring gates, $2n$ squaring gates and $2m$ wires are considered instead of n multiplication gates and m wires; Units: $\mathbb{G}$ stands for group elements, E stands for exponentiations and P stands for pairings.

	zk-SNARK [Gro16]	**SE-SNARK [GM17]**	**Our SE-SNARK**
Circuit base	QAP	SAP	QAP
CRS size	$(m+2n+3)\mathbb{G}_1 + (n+3)\mathbb{G}_2$	$(2m+4n+5)\mathbb{G}_1 + (2n+3)\mathbb{G}_2$	$(m+3n+6)\mathbb{G}_1 + (n+3)\mathbb{G}_2$
Proof size	$2\mathbb{G}_1 + \mathbb{G}_2$	$2\mathbb{G}_1 + \mathbb{G}_2$	$2\mathbb{G}_1 + \mathbb{G}_2$
Prover computation	$(m+3n-l+3)E_1 + (n+1)E_2$	$(2m+4n-l)E_1 + 2nE_2$	$(m+4n-l+3)E_1 + (n+1)E_2$
Verifier computation	$lE_1 + 3P$	$lE_1 + 5P$	$lE_1 + 3P$
Verifying equation	1	2	1

Related work. In the history of proof systems and verifiable computations, there are various NIZK arguments with different types which do not leverage QSP (Quadratic Span Program) or QAP (Quadratic Arithmetic Program) circuits [GKR08,CMT12,WJB+17,WTTW18,BBB+18,ZGK+18,BSCTV14]. A well-known branch comes from the sum-check protocol [GKR08], which gains a sublinear proof from the fiat-shamir transform [FS86]. Nonetheless, they do not support the constant time verification; the verification time is sublinear to the size of the circuits.

Since Gennaro et al. [GGPR13] introduced the Quadratic Span Program(QSP) and Quadratic Arithmetic Program(QAP), zk-SNARK gained a constant proof size and verification. In 2013, Parno et al. [PHGR13] proposed a zk-SNARK scheme called Pinocchio and provided a first practical implementation of zk-SNARK. After Pinocchio, many works added and enhanced some functionalities, such as multiple-function control, additional anonymity for the I/O, or proof scalability [CFH+15,DLFKP16,KPP+14,FFG+16,BBFR15,BSCTV17].

Later, Groth [Gro16] proposed a more efficient zk-SNARK scheme. Compared with Pinocchio [PHGR13], the proof size was reduced from 8 group elements to 3 group elements. Also the number of pairing operations required to verify the proof was reduced from 11 to 3. Recently these SNARK protocols are implemented as an open source [KPS18,BSCG+13] to be used in real applications. By exploiting the short proof sizes and the short verification times, zk-SNARKs can be used as a key component in various cryptographic applications such as anonymous cryptocurrencies [BCG+14,KMS+16,GGM16].

The Zcash [BCG+14], one of the anonymous cryptocurrencies based on blockchain technology, utilized a zk-SNARK to hide transaction information and to provide an efficient verification process. However, since zk-SNARKs [Gro16,PHGR13] do not provide simulation-extractability, Zcash has to add extra cryptographic primitives such as one-time signatures to avoid malleability attacks.

The SE-SNARK scheme [GM17] defines and provides the simulation-extractable SNARK (SE-SNARK), with a similar notion to the Signatures of knowledge [CL06]. While maintaining an efficient proof size of [Gro16], it can prevent the malleability attacks due to the simulation-extractability.

Recently, Bowe and Gabizon [BG18] put an effort to make Groth's scheme [Gro16] simulation-extractable by utilizing random oracle model, with additional hash in proofs and verification. However, unlike their intentions, the proof in their scheme is still malleable (i.e. not simulation-extractable); we show the concrete malleability attack in Appendix B.

Orthogonal to the simulation-extractability, a zk-SNARK with updatable CRS solves the trust issue of CRS by letting the users independently update the CRS [GKM+18]. The traditional limitation of SNARKs is that they all require trusted CRS generation. Through the updating approach, users who distrust a CRS can rely on self-updating.

In this paper, we focus on the simulation-extractable SNARK, specifically pairing-based SE-SNARK. The rest of the paper proceeds as follows: Section 2 provides some necessary notions and backgrounds; Section 3 defines a bilinear group and assumptions; in Section 4, we present our QAP-based SE-SNARK with a single verification; in Section 5, we propose a symmetric SAP-based SE-SNARK with 2 proof elements; Section 6 draws a conclusion.

2 Preliminaries

2.1 Notation

We denote the security parameter with $\lambda \in \mathbb{N}$. For functions $f, g : \mathbb{N} \to [0; 1]$ we write $f(\lambda) \approx g(\lambda)$ if $|f(\lambda) - g(\lambda)| = \lambda^{-\omega(1)}$. We say a function f is negligible if $f(\lambda) \approx 0$, and overwhelming if $f(\lambda) \approx 1$. We implicitly assume all participants and the adversary know the security parameter. If S is a set, $x \leftarrow S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some proper input and returning output x. For an algorithm $\mathcal{A}$ we define $\mathsf{trans}_{\mathcal{A}}$ to be a list containing all of $\mathcal{A}$'s inputs and outputs, including random coins. We use games in security definitions and proofs. A game $\mathcal{G}$ has a main procedure whose output is the output of the game. The notation $\Pr[\mathcal{G}]$ denotes the probability that the output is 1.

2.2 Relations

Given a security parameter 1^λ, a relation generator $\mathcal{R}$ returns a polynomial time decidable relation $R \leftarrow \mathcal{R}(1^\lambda)$. For $(\boldsymbol{\phi}, \boldsymbol{w}) \in R$ we say $\boldsymbol{w}$ is a witness to the instant $\boldsymbol{\phi}$ being in the relation. We denote with $\mathcal{R}_\lambda$ the set of possible relations that $\mathcal{R}(1^\lambda)$ might output.

2.3 Zero-Knowledge Succinct Non-interactive Arguments of Knowledge

Definition 1. *A zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) for $\mathcal{R}$ is a set of four algorithms $Arg = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{SimProve})$ working as follows:*

- $(\mathbf{crs}, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R)$*: the setup algorithm is a PPT algorithm which takes a relation $R \in \mathcal{R}_\lambda$ as input and returns a common reference string* $\mathbf{crs}$ *and a simulation trapdoor* $\boldsymbol{\tau}$*.*
- $\boldsymbol{\pi} \leftarrow \mathsf{Prove}(\mathbf{crs}, \boldsymbol{\phi}, \boldsymbol{w})$*: the prover algorithm is a PPT algorithm which takes a common reference string* $\mathbf{crs}$ *as input for a relation R and $(\boldsymbol{\phi}, \boldsymbol{w}) \in R$ and returns a proof* $\boldsymbol{\pi}$*.*
- $0/1 \leftarrow \mathsf{Vfy}(\mathbf{crs}, \boldsymbol{\phi}, \boldsymbol{\pi})$*: the verifier algorithm is a deterministic polynomial time algorithm which takes a common reference string* $\mathbf{crs}$*, an instance* $\boldsymbol{\phi}$ *and a proof* $\boldsymbol{\pi}$ *as input and returns 0 (reject) or 1 (accept).*
- $\boldsymbol{\pi} \leftarrow \mathsf{SimProve}(\mathbf{crs}, \boldsymbol{\tau}, \boldsymbol{\phi})$*: the simulator is a PPT algorithm which takes a common reference string* $\mathbf{crs}$*, a simulation trapdoor* $\boldsymbol{\tau}$ *and an instance* $\boldsymbol{\phi}$ *as input and returns a proof* $\boldsymbol{\pi}$*.*

It satisfies completeness, knowledge soundness, zero-knowledge, and succinctness as described below:

Perfect Completeness: Perfect completeness states that given a true statement, a prover with a witness can convince the verifier. For all $\lambda \in \mathbb{N}$, for all $R \in \mathcal{R}_\lambda$ and for all $(\boldsymbol{\phi}, \boldsymbol{w}) \in R : Pr[(\mathbf{crs}, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R); \boldsymbol{\pi} \leftarrow \mathsf{Prove}(\mathbf{crs}, \boldsymbol{\phi}, \boldsymbol{w}) : \mathsf{Vfy}(\mathbf{crs}, \boldsymbol{\phi}, \boldsymbol{\pi}) = 1] = 1$.

Computational Knowledge Soundness: Computational knowledge soundness says that the prover must know a witness and such knowledge can be efficiently extracted from the prover by a knowledge extractor. Proof of knowledge requires that for every adversarial prover $\mathcal{A}$ generating an accepting proof, there must be an extractor $\chi_\mathcal{A}$ that, given the same input of $\mathcal{A}$, outputs a valid witness. Formally, we define $\mathbf{Adv}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) = Pr[\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)]$ where the game $\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}$ is defined as follows.

$$
\begin{array}{l}
\underline{\text{MAIN } \mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)} \\
R \leftarrow \mathcal{R}(1^\lambda) \\
(crs, \tau) \leftarrow \mathsf{Setup}(R) \\
(\phi, \pi) \leftarrow \mathcal{A}(crs) \\
\omega \leftarrow \chi_\mathcal{A}(trans_\mathcal{A}) \\
assert\ (\phi, \omega) \notin R \\
return\ \mathsf{Vfy}(crs, \phi, \pi)
\end{array}
$$

An argument system Arg is computationally considered as knowledge sound if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\chi_\mathcal{A}$, such that $\mathbf{Adv}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) \approx 0$.

Perfect Zero-Knowledge: Perfect zero-knowledge states that the system does not leak any information besides the truth of the instance. This is modelled by a simulator that does not know the witness but has some trapdoor information that enables it to simulate proofs. Formally, we define $\mathbf{Adv}^{zk}_{Arg,\mathcal{A}}(\lambda) = 2Pr[\mathcal{G}^{zk}_{Arg,\mathcal{A}(\lambda)}] - 1$ where the game $\mathcal{G}^{zk}_{Arg,\mathcal{A}}$ is defined as follows:

$\underline{\text{MAIN } \mathcal{G}^{zk}_{Arg,\mathcal{A}}(\lambda)}$
$R \leftarrow \mathcal{R}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{Setup}(R)$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{P^b_{crs,\tau}}(crs)$
$return\ 1\ if\ b = b'\ and$
$return\ 0\ otherwise$

$\underline{P^b_{crs,\tau}(\phi_i, w_i)}$
$assert(\phi_i, w_i) \in R$
$\pi_i \leftarrow \mathsf{Prove}(crs, \phi, w)\ if\ b = 0$
$\pi_i \leftarrow \mathsf{SimProve}(crs, \tau, \phi)\ if\ b = 1$
$return\ \pi_i$

The argument system is perfectly zero knowledge if for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}^{zk}_{Arg,\mathcal{A}}(\lambda) = 0$.

Succinctness: Succinctness states that the argument generates the proof of polynomial size in the security parameter, and of the verifier's computation time is polynomial in the security parameter and in instance size.

Definition 2. *A simulation-extractable SNARK system (SE-SNARK) for $\mathcal{R}$ is a zk-SNARK system (*Setup*,* Prove*,* Vfy*,* SimProve*) with simulation-extractability described below:*

Simulation-Extractability [GM17]: Simulation-extractability states that for any adversary $\mathcal{A}$ that sees a simulated proof for a false instance should not allowed to modify the proof into another proof for a false instance. Non-malleability of proofs prevents cheating in the presence of simulated proofs. Formally, we define $\mathbf{Adv}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) = Pr[\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)]$ where the game $\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}$ is defined as follows:

$\underline{\text{MAIN } \mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)}$
$R \leftarrow \mathcal{R}(1^\lambda); Q = \emptyset$
$(crs, \tau) \leftarrow \mathsf{Setup}(R)$
$(\phi, \pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}_{crs,\tau}}(crs)$
$\omega \leftarrow \chi_\mathcal{A}(trans_\mathcal{A})$
$assert\ (\phi, \pi) \notin Q$
$assert\ (\phi, \omega) \notin R$
$return\ \mathsf{Vfy}(crs, \phi, \pi)$

$\underline{\mathsf{SimProve}_{crs,\tau}(\phi_i)}$
$\pi_i \leftarrow \mathsf{SimProve}(crs, \tau, \phi_i)$
$Q = Q \cup \{(\phi_i, \pi_i)\}$
$return\ \pi_i$

An argument is simulation-extractable if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\chi_\mathcal{A}$ such that $\mathbf{Adv}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) \approx 0$.

We note that simulation-extractability implies knowledge soundness, since simulation-extractability corresponds to knowledge soundness where the adversary is allowed to use the simulation oracle SimProve.

When knowledge soundness and simulation-extractability are applied for a succinct argument, extractors are inherently non-black-box. As in [GM17] we assume the relationship generator is benign[5], such that the relation (and the potential auxiliary inputs included in it) are distributed in such a way that the SNARKs we construct can be simulation-extractable.

2.4 Forking Lemma

In our SE-SNARK, we leverage the hash function within the random oracle model. Thus we make use of the following version of the forking lemma [BN06], which is specifically analyzed at NIZKs in a random oracle model [FKMV12].

Lemma 1. (General forking lemma). *Fix an integer Q and a set $\mathcal{H}$ of size $h \geq 2$. Let $\mathcal{P}$ be a randomized program that on input $y, h_1, \cdots, h_Q$ returns a pair, the first element of which is an integer in the range $0, \cdots, Q$ and the second element of which we refer to as a* side output*. Let IG be a randomized algorithm that we call the input generator. The accepting probability of $\mathcal{P}$, denoted as* acc*, is defined as the probability that $J \geq 1$ in the experiment $y \leftarrow IG; h_1, \cdots, h_Q \leftarrow \mathcal{H}; (J, s) \leftarrow \mathcal{P}(y, h_1, \cdots, h_Q)$.*

The forking algorithm $\mathcal{F}_{\mathcal{P}}$ associated to $\mathcal{P}$ is a randomized algorithm that on input y proceeds as follows:

Algorithm $\mathcal{F}_{\mathcal{P}}(y)$
Pick coins ρ for $\mathcal{P}$ at random
$h_1, \cdots, h_Q \leftarrow \mathcal{H}$
$(I, s) \leftarrow \mathcal{P}(y, h_1, \cdots, h_Q; \rho)$
If $I = 0$ *return* $(0, \perp, \perp)$
$h'_I, \cdots, h'_Q \leftarrow \mathcal{H}$
$(I', s') \leftarrow \mathcal{P}(y, h_1, \cdots, h_{I-1}, h'_I, \cdots, h'_Q; \rho)$
If $(I = I') \wedge (h_I \neq h'_I)$ *return* $(1, s, s')$ *else return* $(0, \perp, \perp)$

Let $ext = Prob[b = 1 : y \leftarrow \mathcal{IG}; (b, s, s') \leftarrow \mathcal{F}_{\mathcal{P}}(y)]$, *then* $ext \geq acc(\frac{acc}{Q} - \frac{1}{h})$.

3 Bilinear Groups and Assumptions

In this section, we present the basic bilinear groups and intractability assumptions required for our SE-SNARK, which are adopted from [GM17].

[5] The non-falsifiable *knowledge of exponent* assumption is a necessary ingredient in building a SNARK with witness extraction. In Bitansky's analysis [BCI+13,BCPR16], there are some counter examples and observations; auxiliary inputs may affect the extraction of the witness in extractable one-way functions. However they also observe that the extractability still holds with respect to common auxiliary input that is taken from specific distributions that may be conjectured to be "benign", e.g. the uniform distribution.

Definition 3. *A bilinear group generator $\mathcal{BG}$ takes a security parameter as input in unary and returns a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$ consisting of cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of prime order p and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ possibly together with some auxiliary information (aux) such that:*

- *there are efficient algorithms for computing group operations, evaluating the bilinear map, deciding membership of the groups, and for sampling the generators of the groups;*
- *the map is bilinear, i.e., for all $G \in \mathbb{G}_1$ and $H \in \mathbb{G}_2$ and for all $a, b \in \mathbb{Z}$ we have*
$$e(G^a, H^b) = e(G, H)^{ab};$$
- *and the map is non-degenerate (i.e., if $e(G, H) = 1$ then $G = 1$ or $H = 1$).*

Usually bilinear groups are constructed from elliptic curves equipped with a pairing, which can be tweaked to yield a non-degenerate bilinear map. There are many ways to set up bilinear groups, both as symmetric bilinear groups, where $\mathbb{G}_1 = \mathbb{G}_2$, and as asymmetric bilinear groups, where $\mathbb{G}_1 \neq \mathbb{G}_2$. We will be working in the asymmetric setting, in what Galbraith, Paterson, and Smart [GPS08] call the Type III setting where there is no efficiently computable non-trivial homomorphism in either direction between $\mathbb{G}_1$ and $\mathbb{G}_2$. Type III bilinear groups are the most efficient type of bilinear groups and hence the most relevant for practical applications.

3.1 Intractability Assumptions

We will now specify the intractability assumptions used later to prove our pairing-based SE-SNARK as secure. While variant Power Knowledge of Exponent (PKE) assumptions in a symmetric group are adopted in many SNARK systems [PHGR13], a generalized PKE assumption in asymmetric groups called eXtended PKE has been proposed and utilized in [GM17]. In this paper, we adopt both $XPKE$ assumptions in a symmetric group and in asymmetric groups.

We consider an adversary that gets access to source group elements that have discrete logarithms that are polynomials evaluated on secret random variables. The assumption then says that the only way the adversary can produce group elements in the two source groups for asymmetric groups or type III pairing with matching discrete logarithms. $G^a \in \mathbb{G}_1$ and $H^b \in \mathbb{G}_2$ with $a = b$, is if it knows that b is the evaluation of a known linear combination of the polynomials. Similarly in the symmetric group assumption, the adversary can produce group elements of different generators in the same source group with matching discrete logarithms (i.e. for unknown δ, $G^a, G^{b\delta} \in \mathbb{G}_1$ with $a = b$) if it knows that b is the evaluation of a known linear combination of the polynomials. We denote an asymmetric $XPKE$ assumption as $XPKE_a$ and a symmetric $XPKE$ assumption as $XPKE_s$.

Assumption 1. *Let $\mathcal{A}$ be an adversary and let $\chi_{\mathcal{A}}$ be an extractor. Define in the asymmetric group the advantage* $\mathbf{Adv}^{XPKE_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = Pr[\mathcal{G}^{XPKE_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)]$

where $\mathcal{G}^{XPKE_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}$ *is defined as shown below and* Q_2 *is the set of polynomials* $h_j(X_1,\ldots,X_q)$ *queried to* $\mathcal{O}^2_{H,\boldsymbol{x}}$.

$$\frac{\mathsf{MAIN}\ \mathcal{G}^{XPKE_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)}{(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,aux)\leftarrow\mathcal{BG}(1^\lambda);}$$
$$G\leftarrow\mathbb{G}_1^*;H\leftarrow\mathbb{G}_2^*;\boldsymbol{x}\leftarrow(\mathbb{Z}_p^*)^q$$
$$(G^a,H^b)\leftarrow\mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,aux)$$
$$\boldsymbol{\eta}\in(\mathbb{Z}_p)^{|Q_2|}\leftarrow\chi_{\mathcal{A}}(trans_{\mathcal{A}});$$
$$return\ 1\ if\ a=b\ and\ b\neq\sum_{h_j\in Q_2}\eta_j h_j(\boldsymbol{x})$$
$$else\ return\ 0$$

Define in the symmetric group the advantage $\mathbf{Adv}^{XPKE_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)=Pr[\mathcal{G}^{XPKE_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)]$ *where* $\mathcal{G}^{XPKE_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}$ *is defined as below and* Q_2 *is the set of polynomials* $h_j(X_1,\ldots,X_q)$ *queried to* $\mathcal{O}^2_{H,\boldsymbol{x}}$.

$$\frac{\mathsf{MAIN}\ \mathcal{G}^{XPKE_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)}{(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,aux)\leftarrow\mathcal{BG}(1^\lambda);}$$
$$G\leftarrow\mathbb{G}_1^*;\delta\leftarrow\mathbb{Z}_p;H=G^\delta;\boldsymbol{x}\leftarrow(\mathbb{Z}_p^*)^q$$
$$(G^a,H^b)\leftarrow\mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,aux)$$
$$\boldsymbol{\eta}\in(\mathbb{Z}_p)^{|Q_2|}\leftarrow\chi_{\mathcal{A}}(trans_{\mathcal{A}});$$
$$return\ 1\ if\ a=b\ and\ b\neq\sum_{h_j\in Q_2}\eta_j h_j(\boldsymbol{x})$$
$$else\ return\ 0$$

$$\frac{\mathcal{O}^1_{G,\boldsymbol{x}}(g_i)}{assert\ g_i\in\mathbb{Z}_p[X_1,\ldots,X_q]}$$
$$assert\ deg(g_i)\leq d$$
$$return\ G^{g_i(\boldsymbol{x})}$$

$$\frac{\mathcal{O}^2_{H,\boldsymbol{x}}(h_j)}{assert\ h_j\in\mathbb{Z}_p[X_1,\ldots,X_q]}$$
$$assert\ deg(h_j)\leq d$$
$$return\ H^{h_j(\boldsymbol{x})}$$

The $(d(\lambda),q(\lambda))-XPKE_a$ *and* $(d(\lambda),q(\lambda))-XPKE_s$ *assumptions hold relative to* $\mathcal{BG}$ *if for all PPT adversaries* $\mathcal{A}$*, there exists a PPT algorithm* $\chi_{\mathcal{A}}$ *such that* $\mathbf{Adv}^{XPKE_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$ *and* $\mathbf{Adv}^{XPKE_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$ *are negligible in* λ.

The computational polynomial (Poly) assumption is also adopted in [GM17]. In the univariate case, the Poly assumption states that for any $G\in\mathbb{G}_1^*$, given $G^{g_1(\boldsymbol{x})},\ldots,G^{g_I(\boldsymbol{x})}$, an adversary cannot compute $G^{g(\boldsymbol{x})}$ for a polynomial g that

is linearly independent from $g_1, \ldots, g_I$ - even if it knows $H^{g(\boldsymbol{x})}$ for $H \in G_2^*$. In this paper, we also adopt two Poly assumptions in asymmetric and symmetric groups where $Poly_a$ and $Poly_s$ denote each assumption, respectively.

Assumption 2. *Let $\mathcal{A}$ be an adversary and define the advantage $\mathbf{Adv}^{Poly_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda) = Pr[\mathcal{G}^{Poly_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)]$ where $\mathcal{G}^{Poly_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}$ is defined as below and Q_1 is the set of polynomials $g_j(X_1, \ldots, X_q)$ queried to $\mathcal{O}^1_{H,\boldsymbol{x}}$.*

$\underline{\mathsf{MAIN}\ \mathcal{G}^{Poly_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)}$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux) \leftarrow \mathcal{BG}(1^\lambda);$
$G \leftarrow \mathbb{G}_1^*; H \leftarrow \mathbb{G}_2^*; \boldsymbol{x} \leftarrow (\mathbb{Z}_p^*)^q$
$(G^a, g(X_1, \ldots, X_q)) \leftarrow \mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}}(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$
return 1 *if* $a = g(\boldsymbol{x})$ *and* $g \notin span\{Q_1\}$
else return 0

Define the advantage $\mathbf{Adv}^{Poly_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda) = Pr[\mathcal{G}^{Poly_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)]$ where $\mathcal{G}^{Poly_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}$ is defined as below and Q_1 is the set of polynomials $g_j(X_1, \ldots, X_q)$ queried to $\mathcal{O}^1_{H,\boldsymbol{x}}$.

$\underline{\mathsf{MAIN}\ \mathcal{G}^{Poly_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)}$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux) \leftarrow \mathcal{BG}(1^\lambda);$
$G \leftarrow \mathbb{G}_1^*; \delta \leftarrow \mathbb{Z}_p; H = G^\delta; \boldsymbol{x} \leftarrow (\mathbb{Z}_p^*)^q$
$(G^a, g(X_1, \ldots, X_q)) \leftarrow \mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}}(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$
return 1 *if* $a = g(\boldsymbol{x})$ *and* $g \notin span\{Q_1\}$
else return 0

$\underline{\mathcal{O}^1_{G,\boldsymbol{x}}(g_i)}$
assert $g_i \in \mathbb{Z}_p[X_1, \ldots, X_q]$
assert $deg(g_i) \leq d$
return $G^{g_i(\boldsymbol{x})}$

$\underline{\mathcal{O}^2_{H,\boldsymbol{x}}(h_j)}$
assert $h_j \in \mathbb{Z}_p[X_1, \ldots, X_q]$
assert $deg(h_j) \leq d$
return $H^{h_j(\boldsymbol{x})}$

The $(d(\lambda), q(\lambda)) - Poly_a$ and $(d(\lambda), q(\lambda)) - Poly_s$ assumptions hold relative to $\mathcal{BG}$ if for all PPT adversaries $\mathcal{A}$, we have $\mathbf{Adv}^{Poly_a}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)$ and $\mathbf{Adv}^{Poly_s}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)$ are negligible in λ.

4 QAP-based SE-SNARK Scheme

4.1 Main Idea

As an example of how standard zk-SNARKs can be modified, suppose for a statement ϕ that (A, B, C) $(= (G^a, H^b, G^c))$ are three group elements in a proof that satisfies the verification equations of Groth's zk-SNARK in [Gro16]. Then

$$e(A, B) = e(G^{\alpha}, H^{\beta})e(G^{\frac{f(\phi)}{\gamma}}, H^{\gamma})e(C, H^{\delta}) \tag{1}$$

for a known polynomial f in ϕ and some secret $\alpha, \beta, \gamma, \delta$.

There are two methods to generically randomize a proof A, B, C that satisfies (1). An adversary can either set

$$A' = A^r; B' = B^{\frac{1}{r}}; C' = C$$

or they can set

$$A' = A; B' = BH^{r\delta}; C' = A^r C$$

In the proposed approach, we devise a new way to neutralize the two attacks using the hash of A and B in C. The verification equation is required to detect the changes of A and B. If A or B changes, it should be possible to extract the exponent values a and b from the revised proof. Therefore, the scheme includes two hash functions attached with a and b, respectively. As a result, C is revised as follows:

$$C^* = C \cdot G^{\frac{aH_2(A,B)}{\delta} + bH_1(A,B) + H_1(A,B)H_2(A,B)}$$

where $A = G^a$, $B = H^b$, and H_1 and H_2 are hash functions.

According to the revised C^*, the verification is revised by adding proper additional terms to A and B as follows:

$$e(A \cdot G^{\delta H_1(A,B)}, B \cdot H^{H_2(A,B)}) = e(G^{\alpha}, H^{\beta})e(G^{\frac{f(\phi)}{\gamma}}, H^{\gamma})e(C^*, H^{\delta})$$

Let $h_1 = H_1(A, B)$, $h_2 = H_2(A, B)$, $h'_1 = H_1(A', B')$, $h'_2 = H_2(A', B')$. If A, B change A', B' then C^* should be revised to $C^* \cdot G^{\frac{a(h'_2 - h_2)}{\delta} + b(h'_1 - h_1) + h'_1 h'_2 - h_1 h_2}$. However, since $G^{\frac{a}{\delta}}$ and G^b are only computable if a witness is known, an adversary cannot forge the proof. Note that if A or B changes then hash results $h'_1 = H_1(A, B)$ and $h'_2 = H_2(A, B)$ change. Hence, $G^{\frac{a}{b}}$ and G^b should be known to compute a valid C^* according to h'_1 and h'_2.

4.2 Quadratic Arithmetic Programs

In our SE-SNARK, we will formally adopt the quadratic arithmetic programs (QAP) [GGPR13,Gro16] in a relation R, which is as follows:

$$R = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, l, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X))$$

The bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ defines the finite field $\mathbb{Z}_p$, $1 \leq l \leq m$, and the polynomials $u_i(X), v_i(X), w_i(X)$ represent each linearly independent polynomial set in the QAP with the definition below:

$$\sum_{i=0}^{m} s_i u_i(X) \cdot \sum_{i=0}^{m} s_i v_i(X) \equiv \sum_{i=0}^{m} s_i w_i(X) + h(X)t(X)$$

where $u_i(X), v_i(X), w_i(X)$ have a strictly lower degree than n, which is the degree of $t(X)$. By defining s_0 as 1, the following definition describes the relation R.

$$R = \left\{ (\phi, w) \middle| \begin{array}{l} \phi = (s_1, \cdots, s_l) \in \mathbb{Z}_p^l \\ w = (s_{l+1}, \cdots, s_m) \in \mathbb{Z}_p^{m-l} \\ \\ \exists h(X) \in \mathbb{Z}_p[X], deg(h) \leq n-2 : \\ \qquad \sum_{i=0}^{m} s_i u_i(X) \cdot \sum_{i=0}^{m} s_i v_i(X) \equiv \sum_{i=0}^{m} s_i w_i(X) + h(X)t(X) \end{array} \right\}$$

We say $\mathcal{R}$ is a relation generator for the QAP, given the relation R with field size larger than $2^{\lambda-1}$.

4.3 Construction

– $(crs, \tau) \leftarrow \mathsf{Setup}(R)$: Select generators $G \leftarrow \mathbb{G}_1^*, H \leftarrow \mathbb{G}_2^*$, hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and parameters $\alpha, \beta, \gamma, \delta, x \leftarrow \mathbb{Z}_p^*$, such that $t(x) \neq 0$, and set

$$\tau = (G, H, \alpha, \beta, \gamma, \delta, x)$$

$$crs = \begin{pmatrix} R, H_1, H_2, G, G^{\alpha}, G^{\beta}, G^{\delta}, G^{\alpha\delta}, H, H^{\beta}, H^{\delta} \\ \{G^{\gamma x^i}, H^{\gamma x^i}, G^{\gamma^2 t(x) x^i}, G^{\gamma\delta x^i}\}_{i=0}^{n-1}, \{G^{\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x)}\}_{i=0}^{l}, \\ \{G^{\gamma^2 w_i(x) + \beta\gamma u_i(x) + \alpha\gamma v_i(x)}\}_{i=l+1}^{m} \end{pmatrix}$$

– $\pi \leftarrow \mathsf{Prove}(crs, \phi, w)$: Set $s_0 = 1$ and parse ϕ as $(s_1, \ldots, s_l) \in \mathbb{Z}_p^l$ and w as $(s_{l+1}, \ldots, s_m) \in \mathbb{Z}_p^{m-l}$. Use the witness to compute $h(X)$ from the QAP,

choose $r, s \leftarrow \mathbb{Z}_p$ and compute $\pi = (A, B, C) = (G^a, H^b, G^c)$ such that

$$\begin{aligned} a =& \alpha + \gamma \sum_{i=0}^{m} s_i u_i(x) + r \\ b =& \beta + \gamma \sum_{i=0}^{m} s_i v_i(x) + s \\ c =& \sum_{i=l+1}^{m} s_i(\gamma^2 w_i(x) + \beta\gamma u_i(x) + \alpha\gamma v_i(x)) + \gamma^2 t(x)h(x) + sa + rb - rs \\ & + \delta a h_2 + b h_1 + \delta h_1 h_2 \end{aligned}$$

where $h_1 = H_1(A, B)$ and $h_2 = H_2(A, B)$.

– $0/1 \leftarrow \mathsf{Vfy}(crs, \phi, \pi)$: Parse ϕ as $(s_1, \ldots, s_l) \in \mathbb{Z}_p^l$ and π as $(A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$. Set $s_0 = 1$ and accept the proof if and only if the following equation is satisfied:

$$e(AG^{h_1}, BH^{\delta h_2}) = e(G^{\alpha}, H^{\beta}) e(G^{\sum_{i=0}^{l} s_i(\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x))}, H^{\gamma}) e(C, H)$$

where $h_1 = H_1(A, B)$ and $h_2 = H_2(A, B)$.

– $\pi \leftarrow \mathsf{SimProve}(crs, \tau, \phi)$: Choose $\mu, \nu \leftarrow \mathbb{Z}_p$ and compute $\pi = (A, B, C)$ such that

$$A = G^{\mu}, B = H^{\nu}, C = G^{\mu\nu - \alpha\beta - \gamma \sum_{i=0}^{l} s_i(\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x)) + \delta\mu h_2 + \nu h_1 + \delta h_1 h_2}$$

where $h_1 = H_1(A, B)$ and $h_2 = H_2(A, B)$.

4.4 Security Proof

Theorem 1. *The protocol given above is a non-interactive zero-knowledge argument of knowledge with perfect completeness and perfect zero-knowledge. It is simulation-extractable (implying it also has knowledge soundness) provided that the $XPKE_a$, $XPKE_s$, $Poly_a$, and $Poly_s$ assumptions hold in the random oracle model.*

Proof. Perfect Completeness:

We demonstrate that the prover can compute the proof (A, B, C) as described from the common reference string. The prover can compute the coefficients of

$$h(X) = \frac{(\sum_{i=0}^{m} s_i u_i(X))(\sum_{i=0}^{m} s_i v_i(X)) - (\sum_{i=0}^{m} s_i w_i(X))}{t(X)} = \sum_{j=0}^{n-2} h_j X^j.$$

Now, the proof elements can be computed as follows:

$$A = G^{\alpha} \prod_{j=0}^{n-1} (G^{\gamma x^j})^{u_j} \cdot G^{r}$$

$$B = H^{\beta} \prod_{j=0}^{n-1} (H^{\gamma x^j})^{v_j} \cdot H^{s}$$

$$C = \prod_{i=l+1}^{m} G^{s_i(\gamma^2 w_i(x) + \beta\gamma u_i(x) + \alpha\gamma v_i(x))} \cdot A^{s} A'^{h_2} B'^{(r+h_1)} \cdot G^{-rs} \cdot G^{\delta h_1 h_2} \cdot \prod_{j=0}^{n-1} (G^{\gamma^2 t(x) x^j})^{h_j}$$

where $A' = A^{\delta} = G^{\alpha\delta} \prod_{j=0}^{n-1} (G^{\delta\gamma x^j})^{u_j} \cdot G^{\delta r}$ and $B' = G^{\beta} \prod_{j=0}^{n-1} (G^{\gamma x^j})^{v_j} \cdot G^{s}$.

This computation provides us the proof elements specified in the construction

$$A = G^{\alpha + \gamma \sum_{i=0}^{m} s_i u_i(x) + r}$$

$$B = H^{\beta + \gamma \sum_{i=0}^{m} s_i v_i(x) + s}$$

$$C = G^{\sum_{i=l+1}^{m} s_i(\gamma^2 w_i(x) + \beta\gamma u_i(x) + \alpha\gamma v_i(x)) + \gamma^2 t(x) h(x) + sa + rb - rs + \delta a h_2 + b h_1 + \delta h_1 h_2}.$$

Here we show that the verification equation holds.

$$e(AG^{h_1}, BH^{\delta h_2}) = e(G^{\alpha}, H^{\beta}) e(G^{\sum_{i=0}^{l} s_i(\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x))}, H^{\gamma}) e(C, H)$$

Taking discrete logarithms, checking the verification equation is equivalent to showing that

$$\begin{aligned}
&(a+h_1) \cdot (b + \delta h_2) = ab + \delta a h_2 + b h_1 + \delta h_1 h_2 \\
&= (\alpha + \gamma \sum_{i=0}^{m} s_i u_i(x) + r) \cdot (\beta + \gamma \sum_{i=0}^{m} s_i v_i(x) + s) + \delta a h_2 + b h_1 + \delta h_1 h_2 \\
&= \alpha\beta + \gamma^2 (\sum_{i=0}^{m} s_i u_i(x))(\sum_{i=0}^{m} s_i v_i(x)) + \sum_{i=0}^{m} s_i(\beta\gamma u_i(x) + \alpha\gamma v_i(x)) \\
&\quad + rb + sa - rs + \delta a h_2 + b h_1 + \delta h_1 h_2 \\
&= \alpha\beta + \sum_{i=0}^{m} s_i(\gamma^2 w_i(x) + \beta\gamma u_i(x) + \alpha\gamma v_i(x)) + \gamma^2 t(x) h(x) \\
&\quad + rb + sa - rs + \delta a h_2 + b h_1 + \delta h_1 h_2 \\
&= \alpha\beta + \gamma \sum_{i=0}^{l} s_i(\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x)) + \sum_{i=l+1}^{m} s_i(\gamma^2 w_i(x) + \beta\gamma u_i(x) + \alpha\gamma v_i(x)) \\
&\quad + \gamma^2 t(x) h(x) + rb + sa - rs + \delta a h_2 + b h_1 + \delta h_1 h_2 \\
&= \alpha\beta + \gamma \sum_{i=0}^{l} s_i(\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x)) + c
\end{aligned}$$

where $A = G^{a}$, $B = H^{b}$ and $C = G^{c}$.

Note that since the vector $(s_{l+1}, \ldots, s_m)$ is a valid witness for the instance $(s_1, \ldots, s_l)$, $(\sum_{i=0}^{m} s_i u_i(X))(\sum_{i=0}^{m} s_i v_i(X)) = \sum_{i=0}^{m} s_i w_i(X) + h(X)t(X)$ for all $X \in \mathbb{Z}_p$.

ZERO KNOWLEDGE:

For the zero knowledge, notice that the construction already provides the simulation SimProve which always produces verifying proofs. It can be observed that we get the same distribution over the real proof and the simulated proof, with the choice of random r, s in real proofs and the choice of random μ, ν in simulated proofs.

SIMULATION-EXTRACTABILITY:

Assume that adversary $\mathcal{A}$ succeeds to forge a proof (A, B, C). In the ROM, hash queries must exist on input (A, B) to hash functions H_1 and H_2 respectively. Then we rewind the random tape before the hash query of $H_1(A, B)$ and return a new hash value h_1'. Then, by the forking lemma, a new proof (A, B, C') is obtained. Since $\frac{C'}{C} = G^{b(h_1'-h_1)+h_2(h_1'-h_1)\delta}$, and $(h_1'-h_1)$, h_2, and G^δ are known, we can compute G^b. Similarly, we can compute $G^{\delta a}$ by rewinding the random tape and changing the hash query response for $H_2(A, B)$.

To show simulation-extractability, we will show that if there exists any adversary that breaks simulation-extractability for our scheme we can construct an adversary to break $(2n+2, q+4) - XPKE_a$, $(2n+2, q+4) - XPKE_s$, $(2n+2, q+4) - Poly_a$, or $(2n+2, q+4) - Poly_s$ assumptions, where n is the degree of $t(x)$ defined in the relations, and q is a polynomial upper bound on the number of simulation queries the adversary asks. To put this formally in terms of the games $\mathcal{G}^{prove-ext}$, $\mathcal{G}^{XPKE_a}$, $\mathcal{G}^{XPKE_s}$, $\mathcal{G}_a^{Poly}$ and $\mathcal{G}_s^{Poly}$, we observe that the relation generator R corresponds to a bilinear group generator where the values l, $\{u_i(X), v_i(X), w_i(X)\}_{i=0}^m$, $t(X)$ are auxiliary information. Assuming the maximum number of hash queries to H_1 (in G_1) is Q_1 and the maximum number of hash queries to H_2 (in G_2) is Q_2, we will show (in the ROM) that for all PPT adversaries $\mathcal{A}$ there exists a PPT algorithm $\mathcal{B}_1$ (or $\mathcal{B}_2$), and for all PPT extractors $\chi_{\mathcal{B}_1}$ (or $\chi_{\mathcal{B}_2}$) there exist PPT algorithms $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{D}$, $\chi_{\mathcal{A}}$, such that

$$
\begin{aligned}
&acc(\frac{acc}{Q_1} - \frac{1}{h})(\frac{acc}{Q_2} - \frac{1}{h}) \leq \mathbf{Adv}^{XPKE_a}_{\mathcal{R},2n+2,q+4,\mathcal{B}_1,\chi_{\mathcal{B}_1}}(\lambda) + \mathbf{Adv}^{XPKE_s}_{\mathcal{R},2n+2,q+4,\mathcal{B}_2,\chi_{\mathcal{B}_2}}(\lambda) \\
&+ \mathbf{Adv}^{Poly_a}_{\mathcal{R},2n+2,q+4,\mathcal{C}_1}(\lambda) + \mathbf{Adv}^{Poly_s}_{\mathcal{R},2n+2,q+4,\mathcal{C}_2}(\lambda) + \mathbf{Adv}^{Poly_s}_{\mathcal{R},2n+2,q+4,\mathcal{D}}(\lambda) \\
&where\ \mathbf{Adv}^{prove-ext}_{Arg,\mathcal{A},\chi_{\mathcal{A}}}(\lambda) \leq acc
\end{aligned}
\tag{2}
$$

According to the $XPKE_a$ and $XPKE_s$ assumptions, we can choose $\chi_{\mathcal{B}_1}$ and $\chi_{\mathcal{B}_2}$ such that $\mathbf{Adv}^{XPKE_a}_{\mathcal{R},2n+2,q+4,\mathcal{B}_1,\chi_{\mathcal{B}_1}}(\lambda)$ and $\mathbf{Adv}^{XPKE_s}_{\mathcal{R},2n+2,q+4,\mathcal{B}_2,\chi_{\mathcal{B}_2}}(\lambda)$ are negligible. Combining these with the Poly assumptions makes the latter three advantages negligible. Then the probability of $acc(\frac{acc}{Q_1} - \frac{1}{h})(\frac{acc}{Q_2} - \frac{1}{h})$ becomes negligible. Hence, acc is negligible since the number of hash queries Q_1 and Q_2 are polynomial size and $1/h$ is negligible. We then have that $\mathbf{Adv}^{prove-ext}_{Arg,\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$

is negligible. We will now show how to get the inequality in (2). Consider an execution of the game $\mathcal{G}^{proof-ext}$ with PPT algorithms $\mathcal{A}$,$\chi_{\mathcal{A}}$, which defines $\mathbf{Adv}^{proof-ext}(\lambda)$, using our construction. Our common reference string consists of group generators G, H raised to exponents that are polynomials in X_α, X_β, X_γ, X_δ, X_x evaluated on secret values $\alpha, \beta, \gamma, \delta, x$. Moreover, whenever $\mathcal{A}$ queries the simulation oracle, it gets back a simulated proof, which is a set of three group elements that can be computed by raising G,H to polynomials in indeterminates X_α, X_β, X_γ, X_δ, $X_x, X_{\mu_1}, X_{\nu_1}, \ldots, X_{\mu_q}, X_{\nu_q}$ where we plug in randomly generated $\mu_1, \nu_1, \ldots, \mu_q, \nu_q$ for the latter ones. Let us therefore define a PPT algorithm $\mathcal{E}$ that emulates the execution of $\mathcal{A}$ using access to the oracles described in $\mathcal{G}^{XPKE_a}$, $\mathcal{G}^{XPKE_s}$, $\mathcal{G}^{Poly_a}$, and $\mathcal{G}^{Poly_s}$ for exponentiating G, H to polynomials in secret values.

$$
\begin{array}{l}
\underline{\mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H_1(\cdot),H_2(\cdot)}(R)} \\
G \leftarrow \mathcal{O}^1_{G,\boldsymbol{x}}(X_1); G^\alpha \leftarrow \mathcal{O}^1_{G,\boldsymbol{x}}(X_\alpha); G^\beta \leftarrow \mathcal{O}^1_{G,\boldsymbol{x}}(X_\beta); \ldots \\
H \leftarrow \mathcal{O}^2_{H,\boldsymbol{x}}(X_1); H^\beta \leftarrow \mathcal{O}^2_{H,\boldsymbol{x}}(X_\beta); \ldots \\
crs = (R, G, G^\alpha, G^\beta, \ldots) \\
Q \leftarrow \emptyset \\
(\phi, (G^a, H^b, G^c)) \leftarrow \mathcal{A}^{ZSimProve_{crs,\tau},H_1(\cdot),H_2(\cdot)}(crs) \\
assert\ \mathsf{ZVfy}(crs, \phi, (G^a, H^b, G^c)) = 1 \\
assert\ (\phi, (G^a, H^b, G^c)) \notin Q \\
return\ (\phi, (G^a, H^b, G^c), trans_{\mathcal{A}})
\end{array}
$$

$$
\begin{array}{l}
\underline{ZSimProve^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}_{crs,\tau}(\phi_j)} \\
parse\ \phi_j = (s_1, \ldots, s_l) \\
A_j = G^{\mu_j} \leftarrow \mathcal{O}^1_{G,\boldsymbol{x}}(X_{\mu_j}) \\
B_j = H^{\nu_j} \leftarrow \mathcal{O}^2_{H,\boldsymbol{x}}(X_{\nu_j}) \\
h_{1,j} = H_1(A_j, B_j) \\
h_{2,j} = H_2(A_j, B_j) \\
C_j = G^{\mu_j\nu_j - \alpha\beta - \gamma\sum_{i=0}^{l} s_i(\gamma w_i(x) + \beta u_i(x) + \alpha v_i(x)) + \delta\mu_j h_{2,j} + \nu_j h_{1,j} + \delta h_{1,j} h_{2,j})} \\
\phi \leftarrow \mathcal{O}^1_{G,\boldsymbol{x}}(X_{\mu_j} X_{\nu_j} - X_\alpha X_\beta + \ldots) \\
Q = Q \cup \{(\phi, (A_j, B_j, C_j))\} \\
return (A_j, B_j, C_j)
\end{array}
$$

For the hash queries, the oracle proceeds as follows: if $H_1(A, B)$ is already available in a hash table then return the value. Otherwise, choose a random $h \leftarrow \mathbb{Z}_p^*$ and set $H_1(A, B)$ as h in the hash table and return h. For query $H_2(A, B)$, the same simulation is applied.

With this definition of $\mathcal{E}$ we have

$$\mathbf{Adv}^{proof-ext}_{Arg,\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = Pr\left[\begin{array}{l} R \leftarrow \mathcal{R}(1^\lambda); G \leftarrow \mathbb{G}_1^*; H \leftarrow \mathbb{G}_2^*; \\ \boldsymbol{x} = (\alpha, \beta, \gamma, \delta, x, \mu_1, \ldots, \mu_q, \nu_1, \ldots, \nu_q) \leftarrow (\mathbb{Z}_p^*)^{2q+5}; \\ (\phi, (G^a, H^b, G^c), trans_{\mathcal{A}} \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}, H_1(\cdot), H_2(\cdot)}(R)); w \leftarrow \chi_{\mathcal{A}}(trans_{\mathcal{A}}) \\ : (\phi, w) \notin R \end{array}\right]$$

Given $\mathcal{E}$ and the PPT algorithms $\chi_{\mathcal{B}_1}$ and $\chi_{\mathcal{B}_2}$ we now define algorithms $\mathcal{B}_1, \mathcal{B}_2, \mathcal{C}_1, \mathcal{C}_2, \mathcal{D}, \chi_{\mathcal{A}}$ below using the same oracles $\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}$ as before. Observe that $\mathcal{E}$ and $\mathcal{A}$ see exactly the same group elements, so it is possible to convert a transcript from one to a transcript from the other. Moreover, since the algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ just delete some information from the output of $\mathcal{E}$, the transcripts of $\mathcal{A}$ can be transformed into transcripts of $\mathcal{B}_1$ and $\mathcal{B}_2$. We write the transformations as $trans_{\mathcal{B}_1} = T_1(trans_{\mathcal{A}}, trans'_{\mathcal{A}})$ and $trans_{\mathcal{B}_2} = T_2(trans_{\mathcal{A}}, trans'_{\mathcal{A}})$.

In the following equations, we note that $h_1 = H_1(A, B)$, and $h_2 = H_2(A, B)$. Similarly, $h'_1 = H'_1(A, B)$, and $h'_2 = H'_2(A, B)$. Note that $H_1(\cdot)$ (resp. $H_2(\cdot)$) and $H'_1(\cdot)$ (resp. $H'_2(\cdot)$) return the same hash results for every input except input A, B.

$$\begin{array}{l} \mathcal{B}_1^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}}(R) \\ \hline (\phi, (G^a, H^b, G^c), trans_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}, H_1(\cdot), H_2(\cdot)}(R) \\ (\phi, (G^a, H^b, G^{c'}), trans'_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}, H'_1(\cdot), H_2(\cdot)}(R) \\ G^b = (G^{c'}/G^c)^{(h'_1 - h_1)^{-1}} G^{-h_2 \delta} \\ return\ (G^b, H^b) \end{array}$$

$$\begin{array}{l} \mathcal{B}_2^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}}(R) \\ \hline (\phi, (G^a, H^b, G^c), trans_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}, H_1(\cdot), H_2(\cdot)}(R) \\ (\phi, (G^a, H^b, G^{c'}), trans'_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}, H_1(\cdot), H'_2(\cdot)}(R) \\ G^{\delta a} = (G^{c'}/G^c)^{(h'_2 - h_2)^{-1}} G^{-h_1 \delta} \\ return\ (G^a, G^{\delta a}) \end{array}$$

$\mathcal{C}_1^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(R)$

$(\phi,(G^a,H^b,G^c),trans_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H_1(\cdot),H_2(\cdot)}(R)$

$(\phi,(G^a,H^b,G^{c'}),trans'_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H'_1(\cdot),H_2(\cdot)}(R)$

$G^b = (G^{c'}/G^c)^{(h'_1-h_1)^{-1}} G^{-h_2\delta}$

$trans_{\mathcal{B}_1} = T_1(trans_{\mathcal{A}}, trans'_{\mathcal{A}})$

$\boldsymbol{\eta} \leftarrow \chi_{\mathcal{B}_1}(trans_{\mathcal{B}_1})$

$parse\ \boldsymbol{\eta} = (\eta_1,\ldots,\eta_{n+3+q}) \in \mathbb{Z}_p^{n+3+q}$

$assert\ H^b = \prod_{h_j \in Q_2} (H^{h_j(\boldsymbol{x})})^{\eta_j}$

$let\ b(X) = \sum_{h_j \in Q_2} \eta_j h_j(X)$

$return\ (G^b, b(X))$

$\mathcal{C}_2^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(R)$

$(\phi,(G^a,H^b,G^c),trans_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H_1(\cdot),H_2(\cdot)}(R)$

$(\phi,(G^a,H^b,G^{c'}),trans_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H_1(\cdot),H'_2(\cdot)}(R)$

$G^{\delta a} = (G^{c'}/G^c)^{(h'_2-h_2)^{-1}} G^{-h_1\delta}$

$trans_{\mathcal{B}_2} = T_2(trans_{\mathcal{A}}, trans'_{\mathcal{A}})$

$\boldsymbol{\eta} \leftarrow \chi_{\mathcal{B}_2}(trans_{\mathcal{B}_2})$

$parse\ \boldsymbol{\eta} = (\eta_1,\ldots,\eta_{n+3+q}) \in \mathbb{Z}_p^{n+3+q}$

$assert\ G^a = \prod_{g_j \in Q_1} (G^{g_j(\boldsymbol{x})})^{\eta_j}$

$let\ a(X) = \sum_{g_j \in Q_1} \eta_j g_j(X)$

$return\ (G^a, a(X))$

$$
\begin{array}{l}
\underline{\mathcal{D}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(R)} \\
(\phi,(G^a,H^b,G^c),trans_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H_1(\cdot),H_2(\cdot)}(R) \\
(\phi,(G^a,H^b,G^{c'}),trans'_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H'_1(\cdot),H_2(\cdot)}(R) \\
G^b = (G^{c'}/G^c)^{(h'_1-h_1)^{-1}} G^{-h_2\delta} \\
trans_{\mathcal{B}_1} = T_1(trans_{\mathcal{A}}, trans'_{\mathcal{A}}) \\
\boldsymbol{\eta} \leftarrow \chi_{\mathcal{B}_1}(trans_{\mathcal{B}_1}) \\
parse\ \boldsymbol{\eta} = (\eta_1,\ldots,\eta_{n+3+q}) \in \mathbb{Z}_p^{n+3+q} \\
assert\ H^b = \prod_{h_j \in Q_2} (H^{h_j(\boldsymbol{x})})^{\eta_j} \\
let\ b(X) = \sum_{h_j \in Q_2} \eta_j h_j(X) \\
(\phi,(G^a,H^b,G^{c''}),trans''_{\mathcal{A}}) \leftarrow \mathcal{E}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}},H_1(\cdot),H'_2(\cdot)}(R) \\
G^{\delta a} = (G^{c''}/G^c)^{(h'_2-h_2)^{-1}} G^{-h_1\delta} \\
trans_{\mathcal{B}_2} = T_2(trans_{\mathcal{A}}, trans''_{\mathcal{A}}) \\
\boldsymbol{\eta'} \leftarrow \chi_{\mathcal{B}_2}(trans_{\mathcal{B}_2}) \\
parse\ \boldsymbol{\eta'} = (\eta'_1,\ldots,\eta'_{n+3+q}) \in \mathbb{Z}_p^{n+3+q} \\
assert\ G^a = \prod_{g_j \in Q_1} (G^{g_j(\boldsymbol{x})})^{\eta'_j} \\
let\ a(X) = \sum_{g_j \in Q_1} \eta'_j g_j(X) \\
let\ c(X) = a(X)\cdot b(X) - X_\alpha X_\beta + X_\delta a(X) h_2 + b(X) h_1 + X_\delta h_1 \cdot h_2 \\
- \sum_{i=0}^{l} s_i (X_\gamma^2 w_i(X_x) + X_\alpha X_\gamma v_i(X_x) + X_\beta X_\gamma u_i(X_x)) \\
return (G^c, c(X))
\end{array}
$$

$$
\begin{array}{l}
\underline{\chi_{\mathcal{A}}(trans_{\mathcal{A}})} \\
trans_{\mathcal{B}_1} = T_1(trans_{\mathcal{A}}) \\
\boldsymbol{\eta} \leftarrow \chi_{\mathcal{B}_1}(trans_{\mathcal{B}_1}) \\
parse\ \boldsymbol{\eta} = (\eta_1,\ldots,\eta_{n+3+q}) \in \mathbb{Z}_p^{n+3+q} \\
let\ a(X) = \sum_{h_j \in Q_2} \eta_j h_j(X) \\
assert\ a(X) \in span\{X_\alpha, X_\gamma u_0(X_x),\ldots,X_\gamma u_m(X_x), X_1\} \\
write\ a(X) = X_\alpha + X_\gamma \sum_{i=0}^{m} s_i u_i(X_x) + r \\
return\ (s_{l+1},\ldots,s_m)
\end{array}
$$

Now, let us evaluate $\mathbf{Adv}^{prove-ext}_{Arg,\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$ using the probability with $\mathcal{E}$ given above. Since $\mathcal{E}$ checks the proof is valid and is not coming from a previous query, the adversary wins exactly when the extractor $\chi_{\mathcal{A}}$ fails to extract a valid witness. We will show that this probability is bounded by $\mathbf{Adv}^{XPKE_a}_{\mathcal{R},2n+2,q+4,\mathcal{B}_1,\chi_{\mathcal{B}_1}}(\lambda)$ + $\mathbf{Adv}^{XPKE_s}_{\mathcal{R},2n+2,q+4,\mathcal{B}_2,\chi_{\mathcal{B}_2}}(\lambda)$ + $\mathbf{Adv}^{Poly_a}_{\mathcal{R},2n+2,q+4,\mathcal{C}_1}(\lambda)$ + $\mathbf{Adv}^{Poly_s}_{\mathcal{R},2n+2,q+4,\mathcal{C}_2}(\lambda)$ + $\mathbf{Adv}^{Poly_s}_{\mathcal{R},2n+2,q+4,\mathcal{D}}(\lambda)$.

First, consider the possibility that $\chi_{\mathcal{A}}$ gets an invalid witness. It is the same as $\mathbf{Adv}^{XPKE_a}_{\mathcal{R},2n+2,q+4,\mathcal{B}_1,\chi_{\mathcal{B}_1}}(\lambda)$ + $\mathbf{Adv}^{XPKE_s}_{\mathcal{R},2n+2,q+4,\mathcal{B}_2,\chi_{\mathcal{B}_2}}$.

Second, consider the possibility that $\chi_{\mathcal{A}}$ gets the correct $\boldsymbol{\eta}$, but the polynomial $b(X)$ or $a(X)$ are not the span. This probability is $\mathbf{Adv}^{Poly_a}_{\mathcal{R},2n+2,q+4,\mathcal{C}_1}(\lambda)$ + $\mathbf{Adv}^{Poly_s}_{\mathcal{R},2n+2,q+4,\mathcal{C}_2}(\lambda)$.

Third, there is a possibility that $\chi_{\mathcal{A}}$ gets the correct $\boldsymbol{\eta}$ and $a(X)$ and $b(X)$ are the span of Q_1 and Q_2, respectively. However, $c(X) \notin span\{Q_1\}$. The probability is $\mathbf{Adv}^{Poly_s}_{\mathcal{R},2n+2,q+4,\mathcal{D}}(\lambda)$.

Finally, there is a possibility that $\boldsymbol{\eta}$ defines polynomials $a(X), c(X) \in span\{Q_1\}$ and $b(X) \in span\{Q_2\}$ as satisfying

$$(a(X)+h_1)(b(X)+h_2X_\delta) = X_\alpha X_\beta + X_\gamma \sum_{i=0}^{l} a_{s_i}(X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) + c(X) \tag{3}$$

$$\begin{aligned}
a(X) &= a_0 + a_\alpha X_\alpha + a_\beta X_\beta + a_\delta X_\delta + a_{\alpha\delta} X_\alpha X_\delta + \sum\nolimits_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum\nolimits_{i=0}^{n-1} a_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} a_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} a_{s_i}(X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\
&+ \sum_{i=l+1}^{m} a_{s_i}(X_\gamma^2 w_i(X_x) + X_\beta X_\gamma u_i(X_x) + X_\alpha X_\gamma v_i(X_x)) + \sum_{j=0}^{q} a_{A_j} X_{\mu_j} \\
&+ \sum_{j=0}^{q} a_{C_j}(X_{\mu_j} X_{\nu_j} - X_\alpha X_\beta - X_\gamma \sum_{i=0}^{l} s_{j,i}(X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\
&+ h_2 X_\delta X_{\mu_j} + h_1 X_{\nu_1} + h_1 h_2 X_\delta)
\end{aligned}$$

$$b(X) = b_0 + b_\beta X_\beta + b_\delta X_\delta + \sum\nolimits_{i=0}^{n-1} b_{\gamma x^i} X_\gamma X_x^i + \sum_{j=0}^{q} b_{B_j} X_{\nu_j}$$

$$\begin{aligned} c(X) &= c_0 + c_\alpha X_\alpha + c_\beta X_\beta + c_\delta X_\delta + c_{\alpha\delta} X_\alpha X_\delta + \sum\nolimits_{i=0}^{n-1} c_{\gamma x^i} X_\gamma X_x^i \\ &+ \sum\nolimits_{i=0}^{n-1} c_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} c_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\ &+ \sum_{i=0}^{l} c_{s_i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\ &+ \sum_{i=l+1}^{m} c_{s_i} (X_\gamma^2 w_i(X_x) + X_\beta X_\gamma u_i(X_x) + X_\alpha X_\gamma v_i(X_x)) + \sum_{j=0}^{q} c_{A_j} X_{\mu_j} \\ &+ \sum_{j=0}^{q} c_{C_j} (X_{\mu_j} X_{\nu_j} - X_\alpha X_\beta - X_\gamma \sum_{i=0}^{l} s_{j,i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\ &+ h_2 X_\delta X_{\mu_j} + h_1 X_{\nu_j} + h_1 h_2 X_\delta) \end{aligned}$$

We will now show that in order to satisfy the formal polynomials equations above, either the adversary must recycle an instance and a proof, or alternatively $\chi_{\mathcal{A}}$ manages to extract a witness. First, suppose we have some $a_{A_k} \neq 0$. Since there are only X_{μ_k}, $X_{\mu_k} X_\delta$, and $X_{\mu_k} X_{\nu_k}$ related with X_{μ_k} in the right form, $b(X) = b_0 + b_\delta X_\delta + b_{B_k} X_{v_k}$. Similarly, since there are only X_{ν_k} and $X_{\mu_k} X_{\nu_k}$ in the right form, $a(X) = a_0 + a_{A_k} X_{\mu_k}$. Plugging this into (3) gives us,

$$\begin{aligned} &(a_0 + a_{A_k} X_{\mu_k} + h_1')(b_0 + b_\delta X_\delta + b_{B_k} X_{v_k} + h_2' X_\delta) \\ &= X_\alpha X_\beta + X_\gamma \sum_{i=0}^{l} a_{s_i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) + c(X) \end{aligned}$$

The only way this is possible is by setting

$$\begin{aligned} c(X) &= c_0 + c_{C_k} (X_{\mu_k} X_{\nu_k} - X_\alpha X_\beta - X_\gamma \sum_{i=0}^{l} s_{k,i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\ &+ h_2 X_\delta X_{\mu_k} + h_1 X_{\nu_k} + h_1 h_2 X_\delta) + c_{A_k} X_{\mu_k} \end{aligned}$$

This implies $c_{C_k} = 1$ due to $X_\alpha X_\beta$. $a_{A_k}(b_\delta + h_2') = h_2$ and $(a_0 + h_1')b_{B_k} = h_1$ due to X_{μ_k} and X_{ν_k}. Since h_2' depends on A and B, it is hard to find h_2' such that $h_2' \neq h_2$ to satisfy $a_{A_k}(b_\delta + h_2') = h_2$. Similarly, it is hard to find a_0 and b_{B_k} such that $(a_0 + h_1')b_{B_k} = h_1$ except that $h_1 = h_1'$ (i.e., $A = G^{\mu_k}$ and $B = H^{\nu_k}$). Therefore, $a_0 = 0$, $a_{A_k} = 1$, $b_0 = 0$, $b_\delta = 0$ and b_{B_k}=1 to meet that $h_1 = h_1'$ and $h_2 = h_2'$. Hence, $c_0 = 0$ and $c_{A_k} = 0$. Since $u_i(X_x)_{i=1}^{l}$ are linearly independent, we see for $i = 1, \ldots, l$ that $s_i = s_{k,i}$. In other words, the adversary has recycled the k-th instance $\pi = \pi_k$ and the proof $(A, B, C) = (A_k, B_k, C_k)$. The same conclusion is obtained if $b_{B_k} \neq 0$.

Next, suppose for all $j = 1, \ldots, q$ that $a_{A_j} = b_{B_j} = 0$. If $b_\beta = 0$ then $\sum_{j=0}^{q} -a_{C_j} X_\alpha X_\beta = X_\alpha X_\beta - \sum_{j=0}^{q} c_{C_j} X_\alpha X_\beta$. However since $a_{C_j} X_{\mu_j} X_{\nu_j} = c_{C_j} X_{\mu_j} X_{\nu_j}$, $\sum_{j=0}^{q} -a_{C_j} X_\alpha X_\beta \neq X_\alpha X_\beta - \sum_{j=0}^{q} c_{C_j} X_\alpha X_\beta$. Hence $b_\beta \neq 0$. In the right form, there are only X_β, $X_\beta X_\gamma$, $X_\beta X_\alpha$, and $X_\beta u_i(X_x)$ related with X_β, $a(X) = a_0 + a_\alpha X_\alpha + \sum_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i$. Since $c_{C_j} = 0$, $a_\alpha b_\beta = 1$ and $a_\alpha \neq 0$. Finally, $b(X) = b_0 + b_\beta X_\beta + b_\delta X_\delta + \sum_{i=0}^{n-1} b_{\gamma x^i} X_\gamma X_x^i$. We are now left with

$$
\begin{aligned}
c(X) &= c_0 + c_\alpha X_\alpha + c_\beta X_\beta + c_\delta X_\delta + c_{\alpha\delta} X_\alpha X_\delta + \sum_{i=0}^{n-1} c_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} c_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} c_{s_i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\
&+ \sum_{i=l+1}^{m} c_{s_i} (X_\gamma^2 w_i(X_x) + X_\beta X_\gamma u_i(X_x) + X_\alpha X_\gamma v_i(X_x))
\end{aligned}
$$

In (3),

$$
\begin{aligned}
&(a_0 + a_\alpha X_\alpha + \sum_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i + h_1')(b_0 + b_\beta X_\beta + \sum_{i=0}^{n-1} b_{\gamma x^i} X_\gamma X_x^i + (b_\delta + h_2') X_\delta) \\
&= X_\alpha X_\beta + X_\gamma \sum_{i=0}^{l} a_{s_i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\
&+ c_0 + c_\alpha X_\alpha + c_\beta X_\beta + c_\delta X_\delta + c_{\alpha\delta} X_\alpha X_\delta + \sum_{i=0}^{n-1} c_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} c_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} c_{s_i} (X_\gamma w_i(X_x) + X_\beta u_i(X_x) + X_\alpha v_i(X_x)) \\
&+ \sum_{i=l+1}^{m} c_{s_i} (X_\gamma^2 w_i(X_x) + X_\beta X_\gamma u_i(X_x) + X_\alpha X_\gamma v_i(X_x))
\end{aligned}
$$

Define for $i = l+1, \ldots, m$ that $s_i = c_{s_i}$. The terms involving $X_\beta X_\gamma X_x^i$ now give us $b_\beta \sum_{i=0}^{n-1} a_{\gamma x^i} X_x^i = \sum_{i=0}^{m} s_i u_i(X_x)$. In addition, the terms involving $X_\alpha X_\gamma X_x^i$ provide $a_\alpha \sum_{i=0}^{n-1} b_{\gamma x^i} X_x^i = \sum_{i=0}^{m} s_i v_i(X_x)$. Note that $a_\alpha b_\beta = 1$. Finally,

the terms involving X_γ^2 produce

$$X_\gamma \sum_{i=0}^{m} s_i u_i(X_x) \cdot X_\gamma \sum_{i=0}^{m} s_i v_i(X_x) = X_\gamma^2 a_\alpha b_\beta \sum_{i=0}^{n-1} a_{\gamma x^i} X_x^i \sum_{i=0}^{n-1} b_{\gamma x^i} X_x^i$$
$$= X_\gamma^2 (\sum_{i=0}^{m} s_i w_i(X_x) + t(X_x) \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_x^i)$$

Defining $h(X_x) = \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_x^i$ we see that this means $(s_{l+1}, \ldots, s_m)$ is a witness for the instance $(s_1, \ldots, s_l)$ (the extracted witness may be one of many possible valid witnesses).

5 QAP-based SE-SNARKs with Two Group Elements as a Proof

5.1 Hardness of Two Group Elements SE-SNARKs

In the previous section, we propose an efficient SE-SNARKs scheme with three group elements as a proof. Now it is interesting to observe whether it is possible to build a similar SE-SNARK scheme with two group elements if adopting Type I pairing instead of Type III pairing. Since each multiplication gate $a \cdot b = c$ can be transformed to $(a+b)^2 - (a-b)^2 = 4c$ as a square arithmetic program (SAP), it is possible to get a 2-element for boolean circuit satisfiability by changing a multiplication gate to two squaring gates. If we assume a generic group model like [Gro16], then it is possible to build a scheme with two group elements. However, if concrete intractability assumptions such as $XPKE_a$ and $XPKE_s$ are considered instead of relying on the full generic group model, it is not a simple problem to construct a scheme with two elements even in a symmetric group.

Consider the SE-SNARK from SE-SNARK [GM17]. Since it is defined in asymmetric groups, if it is redefined in a symmetric group then the scheme may require two group elements as proof. Unfortunately, security cannot be proven in the symmetric group version of [GM17]. The scheme utilizes an SAP to guarantee that two elements have equivalent exponents or $A = G^b$ and $B = H^b$. By using the $XPKE_a$ assumption and an extractor, it is possible to extract coefficients to compute b which is the evaluation of a known linear combination of polynomials. In a symmetric group, for a two group elements scheme, the $XPKE_s$ assumption should be used instead of an $XPKE_a$ assumption. However, we cannot use $XPKE_s$ assumption for a proof of (A, C) $(= (G^a, G^c))$ since c includes the square (or quadratic) of a while c should be a linear (or δ times) of a to use the $XPKE_s$ assumption.

5.2 Square Arithmetic Programs

In the SE-SNARK with two group elements, we will work with square arithmetic programs (SAP) R, with the definitions adopted from [GM17].

$$R = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, l, \{u_i(X), w_i(X)\}_{i=0}^{m}, t(X))$$

The bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ defines the finite field $\mathbb{Z}_p$, $1 \leq l \leq m$, and the polynomials $u_i(X), w_i(X)$ represent each linearly independent polynomial set in the SAP with the definition below:

$$(\sum_{i=0}^{m} s_i u_i(X))^2 \equiv \sum_{i=0}^{m} s_i w_i(X) + h(X)t(X)$$

where $u_i(X), w_i(X)$ have a strictly lower degree than n, which is the degree of $t(X)$. By defining s_0 as 1, the following definition describes the relation R.

$$R = \left\{ (\phi, w) \middle| \begin{array}{l} \phi = (s_1, \cdots, s_l) \in \mathbb{Z}_p^l \\ w = (s_{l+1}, \cdots, s_m) \in \mathbb{Z}_p^{m-l} \\ \\ \exists h(X) \in \mathbb{Z}_p[X], deg(h) \leq n-2 : \\ \qquad (\sum_{i=0}^{m} s_i u_i(X))^2 \equiv \sum_{i=0}^{m} s_i w_i(X) + h(X)t(X) \end{array} \right\}$$

We say $\mathcal{R}$ is a relation generator for the SAP, given the relation R with a field size larger than $2^{\lambda-1}$.

5.3 Proposed SE-SNARKs with Two Group Elements

In this section, we propose a scheme with two group elements as a proof in a symmetric group using SAP. In the proposed scheme using a hash function, it is possible to compute $(G^a, G^{\delta a})$ from (A, C) in ROM, since the square of the exponent of A in the exponent of C can be removed by rewinding the adversary (i.e., forking lemma). Consequently, we can apply the $XPKE_s$ and $Poly_s$ assumptions.

- $(crs, \tau) \leftarrow \mathsf{Setup}(R)$: Select a generator $G \leftarrow \mathbb{G}$, hash functions $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, and parameters $\alpha, \gamma, \delta, x \leftarrow \mathbb{Z}_p^*$, such that $t(x) \neq 0$, and set

$$\tau = (G, \alpha, \gamma, \delta, x)$$
$$crs = \begin{pmatrix} R, H, G, G^{\alpha}, G^{\delta}, G^{\alpha\delta} \\ \{G^{\gamma x^i}, G^{\gamma^2 t(x) x^i}, G^{\gamma \delta x^i}\}_{i=0}^{n-1}, \{G^{\gamma w_i(x) + 2\alpha u_i(x)}\}_{i=0}^{l}, \\ \{G^{\gamma^2 w_i(x) + 2\alpha\gamma u_i(x)}\}_{i=l+1}^{m} \end{pmatrix}$$

- $\pi \leftarrow \mathsf{Prove}(crs, \phi, w)$: Set $s_0 = 1$ and parse ϕ as $(s_1, \ldots, s_l) \in \mathbb{Z}_p^l$ and w as $(s_{l+1}, \ldots, s_m) \in \mathbb{Z}_p^{m-l}$. Use the witness to compute $h(X)$ from the SAP,

pick $r \leftarrow \mathbb{Z}_p$ and compute $\pi = (A, C) = (G^a, G^c)$ such that

$$a = \alpha + \gamma \sum_{i=0}^{m} s_i u_i(x) + r$$
$$c = \sum_{i=l+1}^{m} s_i(\gamma^2 w_i(x) + 2\alpha\gamma u_i(x)) + \gamma^2 t(x)h(x) + 2ra - r^2 + \delta aH(A)$$

– $0/1 \leftarrow \mathsf{Vfy}(crs, \phi, \pi)$: Parse ϕ as $(s_1, \ldots, s_l) \in \mathbb{Z}_p^l$ and π as $(A, C) \in \mathbb{G} \times \mathbb{G}$. Set $s_0 = 1$ and check that

$$e(AG^{\delta H(A)}, A) = e(G^\alpha, G^\alpha) e(G^{\sum_{i=0}^{l} s_i(\gamma w_i(x) + 2\alpha u_i(x))}, G^\gamma) e(C, G)$$

Accept the proof if and only if the test passes.

– $\pi \leftarrow \mathsf{SimProve}(crs, \tau, \phi)$: Pick $\mu \leftarrow \mathbb{Z}_p$ and compute $\pi = (A, C)$ such that

$$A = G^\mu, C = G^{\mu^2 - \alpha^2 - \gamma \sum_{i=0}^{l} s_i(\gamma w_i(x) + 2\alpha u_i(x)) + \delta\mu H(A)}$$

Theorem 2. *The protocol given above is a non-interactive zero-knowledge argument of knowledge with perfect completeness and perfect zero-knowledge. It is simulation-extractable (implying it also has knowledge soundness) provided that the $XPKE_s$ and $Poly_s$ assumptions hold in the random oracle model.*

6 Conclusion

In this paper, we propose the first quadratic arithmetic program based simulation-extractable succinct non-interactive arguments of knowledge (QAP-based SE-SNARK) with 3 group elements, which requires a single equation for verification. Our scheme is constructed beyond the existing non-interactive linear proofs (NILP), by utilizing random oracles. The proposed scheme is proven under concrete intractability assumptions (not generic group model), even with leveraging QAP. We also propose an SE-SNARK scheme with 2 elements as proof with the SAP representation in a symmetric group, although it is difficult to construct a scheme with 2 elements as proof from existing SE-SNARKs.

References

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Bulletproofs: Short Proofs for Confidential Transactions and More*, page 0. IEEE, 2018.

[BBFR15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M Reischuk. Adsnark: nearly practical and privacy-preserving proofs on authenticated data. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 271–286. IEEE, 2015.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography*, pages 315–333. Springer, 2013.

[BCPR16] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. *SIAM Journal on Computing*, 45(5):1910–1952, 2016.

[BG18] Sean Bowe and Ariel Gabizon. Making groth's zk-snark simulation extractable in the random oracle model. Cryptology ePrint Archive, Report 2018/187, 2018. https://eprint.iacr.org/2018/187.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399. ACM, 2006.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[BSCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796, 2014.

[BSCTV17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy (SP)*, pages 253–270. IEEE, 2015.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 78–96, 2006.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM, 2012.

[DLFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby x. 509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 235–254. IEEE, 2016.

[FFG+16] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1304–1316. ACM, 2016.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *International Conference on Cryptology in India*, pages 60–79. Springer, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[GGM16] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *International Conference on Financial Cryptography and Data Security*, pages 81–98. Springer, 2016.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Annual International Cryptology Conference*, pages 698–728. Springer, 2018.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 113–122. ACM, 2008.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from Simulation-Extractable SNARKs. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083, 2016.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[Gro16] Jens Groth. On the size of Pairing-Based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.

[KMS+16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[KPP+14] Ahmed E Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F Sayed, Elaine Shi, and Nikos Triandopoulos. Trueset: Faster verifiable set computations. In *USENIX Security Symposium*, pages 765–780, 2014.

[KPS18] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. xjsnark: A framework for efficient verifiable computation. In *xJsnark: A Framework for Efficient Verifiable Computation*, page 0. IEEE, 2018.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252, 2013.

[WJB+17] Riad S Wahby, Ye Ji, Andrew J Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2071–2086. ACM, 2017.

[WTTW18] Riad S Wahby, Ioanna Tzialla, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. 2018.

[ZGK+18] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vram: Faster verifiable ram with program-independent preprocessing. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)*, 2018.

Appendix

A Security Proof for SAP-based SE-SNARK

We show the security proof of our proposed SE-SNARK with two group elements in section 5.3.

Theorem 2. *The protocol given above is a non-interactive zero-knowledge argument of knowledge with perfect completeness and perfect zero-knowledge. It is simulation-extractable (implying it also has knowledge soundness) provided that the $XPKE_s$ and $Poly_s$ assumptions hold in the random oracle model.*

Proof. Perfect Completeness:

First, we state that the prover can compute the proof (A, C) as described from the common reference string. The prover can compute the coefficients of

$$h(X) = \frac{(\sum_{i=0}^{m} s_i u_i(X))^2 - (\sum_{i=0}^{m} s_i w_i(X))}{t(X)} = \sum_{j=0}^{n-2} h_j X^j.$$

It can now compute the proof elements as

$$A = G^{\alpha} \prod_{j=0}^{n-1} (G^{\gamma x^j})^{u_j} \cdot G^{r}$$

$$C = \prod_{i=l+1}^{m} G^{s_i(\gamma^2 w_i(x) + 2\alpha\gamma u_i(x))} \cdot A'^{H(A)} \cdot G^{-r^2} \cdot \prod_{j=0}^{n-1} (G^{\gamma^2 t(x) x^j})^{h_j}$$

where $A' = A^{\delta} = G^{\alpha\delta} \prod_{j=0}^{n-1} (G^{\delta\gamma x^j})^{u_j} \cdot G^{\delta r}$.

This computation provides us the proof elements specified in the construction

$$A = G^{\alpha + \gamma \sum_{i=0}^{m} s_i u_i(x) + r}$$

$$C = G^{\sum_{i=l+1}^{m} s_i(\gamma^2 w_i(x) + 2\alpha\gamma u_i(x)) + \gamma^2 t(x) h(x) + 2ra - r^2 + \delta a H(A)}$$

Here we show that the verification equation holds.

$$e(AG^{\delta H(A)}, A) = e(G^{\alpha}, G^{\alpha}) e(G^{\sum_{i=0}^{l} s_i(\gamma w_i(x) + 2\alpha u_i(x))}, G^{\gamma}) e(C, G)$$

Taking discrete logarithms, this is equivalent to showing that

$$
\begin{aligned}
&(a+\delta H(A)) \cdot a = a^2 + \delta a H(A) \\
&=(\alpha + \gamma \sum_{i=0}^{m} s_i u_i(x) + r)^2 + \delta a H(A) \\
&=\alpha^2 + \gamma^2 (\sum_{i=0}^{m} s_i u_i(x))^2 + 2\alpha\gamma \sum_{i=0}^{m} s_i u_i(x) + 2ra - r^2 + \delta a H(A) \\
&=\alpha^2 + \sum_{i=0}^{m} s_i(\gamma^2 w_i(x) + 2\alpha\gamma u_i(x)) + \gamma^2 t(x)h(x) + 2ra - r^2 + \delta a H(A) \\
&=\alpha^2 + \gamma \sum_{i=0}^{l} s_i(\gamma w_i(x) + 2\alpha u_i(x)) \\
&\quad + \sum_{i=l+1}^{m} s_i(\gamma^2 w_i(x) + 2\alpha\gamma u_i(x)) + \gamma^2 t(x)h(x) + 2ra - r^2 + \delta a H(A) \\
&=\alpha^2 + \gamma \sum_{i=0}^{l} s_i(\gamma w_i(x) + 2\alpha u_i(x)) + c
\end{aligned}
$$

where $A = G^a$, and $C = G^c$.

Note that since the vector $(s_{l+1}, \ldots, s_m)$ is a valid witness for the instance $(s_1, \ldots, s_l)$, $(\sum_{i=0}^{m} s_i u_i(X))^2 = \sum_{i=0}^{m} s_i w_i(X) + h(X)t(X)$ for all $X \in \mathbb{Z}_p$.

SIMULATION-EXTRACTABILITY:

Finally, there is a possibility that $\boldsymbol{\eta}$ defines polynomials $a(X), c(X) \in span\{Q_1\}$ as satisfying

$$
(a(X)+\delta H(A)) \cdot a(X) = X_\alpha^2 + X_\gamma \sum_{i=0}^{l} s_i(X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) + c(X) \tag{4}
$$

$$
\begin{aligned}
a(X) &= a_0 + a_\alpha X_\alpha + a_\delta X_\delta + a_{\alpha\delta} X_\alpha X_\delta + \sum\nolimits_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum\nolimits_{i=0}^{n-1} a_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} a_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} a_{s_i} (X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) \\
&+ \sum_{i=l+1}^{m} a_{s_i} (X_\gamma^2 w_i(X_x) + 2X_\alpha X_\gamma u_i(X_x)) + \sum_{j=0}^{q} a_{A_j} X_{\mu_j} \\
&+ \sum_{j=0}^{q} a_{C_j} (X_{\mu_j}^2 - X_\alpha^2 - X_\gamma \sum_{i=0}^{l} s_{j,i} (X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) + H(G^{X_{\mu_j}}) X_\delta X_{\mu_j}) \\
c(X) &= c_0 + c_\alpha X_\alpha + c_\delta X_\delta + c_{\alpha\delta} X_\alpha X_\delta + \sum\nolimits_{i=0}^{n-1} c_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum\nolimits_{i=0}^{n-1} c_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} c_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} c_{s_i} (X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) \\
&+ \sum_{i=l+1}^{m} c_{s_i} (X_\gamma^2 w_i(X_x) + 2X_\alpha X_\gamma u_i(X_x)) + \sum_{j=0}^{q} c_{A_j} X_{\mu_j} \\
&+ \sum_{j=0}^{q} c_{C_j} (X_{\mu_j}^2 - X_\alpha^2 - X_\gamma \sum_{i=0}^{l} s_{j,i} (X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) + H(G^{X_{\mu_j}}) X_\delta X_{\mu_j})
\end{aligned}
$$

We will now show that in order to satisfy the formal polynomials equations above, either the adversary must recycle an instance and a proof, or alternatively $\chi_{\mathcal{A}}$ manages to extract a witness. First, suppose we have some $a_{A_k} \neq 0$. Since there are only X_{μ_k}, $X_{\mu_k} X_\delta$, and $X_{\mu_k}^2$ related with X_{μ_k} and there is no X_δ^2 in the right form, $a(X) = a_0 + a_{A_k} X_{\mu_k}$. Plugging this into (4) gives us,

$$
\begin{aligned}
&(a_0 + a_{A_k} X_{\mu_k} + \delta H(A))(a_0 + a_{A_k} X_{\mu_k}) \\
&= X_\alpha^2 + X_\gamma \sum_{i=0}^{l} a_{s_i} (X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) + c(X)
\end{aligned}
$$

The only way this is possible is by setting

$$
\begin{aligned}
c(X) &= c_0 + c_{A_k} X_{\mu_k} + c_{C_k} (X_{\mu_k}^2 - X_\alpha^2 - X_\gamma \sum_{i=0}^{l} s_{k,i} (X_\gamma w_i(X_x) + 2X_\alpha u_i(X_x)) \\
&+ H(G^{X_{\mu_k}}) X_\delta X_{\mu_k})
\end{aligned}
$$

This implies $c_{C_k} = 1$ due to X_α^2. $a_{A_k} H(A) = H(G^{X_{\mu_k}})$ where $A = G^{a_0 + A_k X_{\mu_k}}$. Since it is hard to find a_0 and A_k to satisfy $a_{A_k} H(A) = H(G^{X_{\mu_k}})$ in hash H except queried hash value for $A = G^{\mu_k}$, $a_0 = 0$ and $a_{A_k} = 1$. Therefore, $c_0 = 0$ and $c_{A_k} = 0$. Since $u_i(X_x)_{i=1}^{l}$ are linearly independent, we see for $i = 1, \ldots, l$ that $s_i = s_{k,i}$. In other words, the adversary has recycled the k-th instance $\pi = \pi_k$ and proof $(A, C) = (A_k, C_k)$.

Next, suppose for all $j = 1, \ldots, q$ that $a_{A_j} = 0$. Since there is X_α^2 in the right form, $a_\alpha^2 = 1$. In the right form, there are only X_α, X_α^2, $X_\alpha X_\gamma$, $X_\alpha X_\delta$, and $X_\alpha u_i(X_x)$ related with X_α and there is no X_δ^2, $a(X) = a_0 + a_\alpha X_\alpha + \sum_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i$. We are now left with

$$
\begin{aligned}
c(X) &= c_0 + c_\alpha X_\alpha + c_\delta X_\delta + c_{\alpha\delta} X_\alpha X_\delta + \sum_{i=0}^{n-1} c_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} c_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} c_{s_i} (X_\gamma w_i(X_x) + 2 X_\alpha u_i(X_x)) \\
&+ \sum_{i=l+1}^{m} c_{s_i} (X_\gamma^2 w_i(X_x) + 2 X_\alpha X_\gamma u_i(X_x))
\end{aligned}
$$

In (4),

$$
\begin{aligned}
&(a_0 + a_\alpha X_\alpha + \textstyle\sum_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i + H(A) X_\delta)(a_0 + a_\alpha X_\alpha + \textstyle\sum_{i=0}^{n-1} a_{\gamma x^i} X_\gamma X_x^i) \\
&= X_\alpha^2 + X_\gamma \sum_{i=0}^{l} a_{s_i} (X_\gamma w_i(X_x) + 2 X_\alpha u_i(X_x)) \\
&+ c_0 + c_\alpha X_\alpha + c_\delta X_\delta + c_{\alpha\delta} X_\alpha X_\delta + \sum_{i=0}^{n-1} c_{\gamma x^i} X_\gamma X_x^i \\
&+ \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_\gamma^2 t(X_x) X_x^i + \sum_{i=0}^{n-1} c_{\gamma\delta x^i} X_\gamma X_\delta X_x^i \\
&+ \sum_{i=0}^{l} c_{s_i} (X_\gamma w_i(X_x) + 2 X_\alpha u_i(X_x)) \\
&+ \sum_{i=l+1}^{m} c_{s_i} (X_\gamma^2 w_i(X_x) + 2 X_\alpha X_\gamma u_i(X_x))
\end{aligned}
$$

Define for $i = l+1, \ldots, m$ that $s_i = c_{s_i}$. The terms involving $X_\alpha X_\gamma X_x^i$ now give us $a_\alpha \sum_{i=0}^{n-1} a_{\gamma x^i} X_x^i = \sum_{i=0}^{m} s_i u_i(X_x)$. Finally, the terms involving X_γ^2

produce

$$(X_\gamma \sum_{i=0}^{m} s_i u_i(X_x))^2 = X_\gamma^2 a_\alpha^2 (\sum_{i=0}^{n-1} a_{\gamma x^i} X_x^i)^2$$
$$= X_\gamma^2 (\sum_{i=0}^{m} s_i w_i(X_x) + t(X_x) \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_x^i)$$

Defining $h(X_x) = \sum_{i=0}^{n-1} c_{\gamma^2 t x^i} X_x^i$ we see that this means that $(s_{l+1}, \ldots, s_m)$ is a witness for the instance $(s_1, \ldots, s_l)$ (the extracted witness may be one of many possible valid witnesses).

B Malleability attacks on [BG18]

Recently, Bowe and Gabizon [BG18] made an effort to make Groth's SNARK scheme [Gro16] simulation-extractable, by utilizing random oracle model. However, unlike their intention, the scheme is still vulnerable to the malleability attack, i.e., not simulation-extractable.

Below, we will show that a new valid proof can be forged from an existing proof, which denotes that [BG18] is malleable.

In [BG18], the construction is as follows:

Prove: Choose d randomly and compute $\delta' = d \cdot \delta$. Compute the proof A, B, C similar to [Gro16], except that δ' is used instead of δ. Then set $z = \mathcal{H}(G^A, H^B, G^C, H^{\delta'}) \cdot d$. The prover outputs the proof as $\pi_1 = (G^A, G^C, G^z), \pi_2 = (H^B, H^{\delta'})$.

Verify: Given $G^A, H^B, G^C, G^{\delta'}, G^z$, check that:

1. $e(G^A, H^B) = e(G^\alpha, H^\beta) \cdot e(G^{\mathsf{ic}}, H^\gamma) \cdot e(G^C, H^{\delta'})$
2. $e(G, H^{\delta' \cdot H(G^A, H^B, G^C, H^{\delta'})}) = e(G^z, H^\delta)$

where $\mathsf{ic} = \frac{\sum_{i=0}^{l} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\gamma}$, which is the element relating to the primary QAP inputs $(a_0 = 1, a_1, \cdots, a_l)$.

Forge: We can generate a new valid proof $\pi' = (G^{A \cdot r}, H^{B \cdot r^{-1}}, G^C, H^{\delta'}, G^{z \cdot y_s^{-1} \cdot y'_s})$ with a random r, from an existing proof $(G^A, H^B, G^C, H^{\delta'}, G^z)$ where $y_s = \mathcal{H}(G^A, H^B, G^C, H^{\delta'})$ and $y'_s = \mathcal{H}(G^{A \cdot r}, H^{B \cdot r^{-1}}, G^C, H^{\delta'})$.

The forged proof π' can also pass the verification as below:

1. $e(G^(A \cdot r), H^(B \cdot r^{-1})) = e(G^A, H^B) = e(G^\alpha, H^\beta) \cdot e(G^{\mathsf{ic}}, H^\gamma) \cdot e(G^C, H^{\delta'})$
2. $e(G^{\mathcal{H}(A \cdot r, B \cdot r^{-1}, C, \delta')}, H^{\delta'}) = e(G^{y'_s}, H^{\delta'}) = e(G^{y'_s}, H^{\frac{z \cdot \delta}{y_s}}) = e(G^{y'_s \cdot y_s^{-1} \cdot z}, H^\delta)$

since $\delta' = z \cdot \frac{\delta}{H(A,B,C,\delta')}$.

Therefore, since a forged π' can pass the verification, [BG18] is still malleable (not simulation-extractable). In fact, this is a good example to show that it is not trivial to construct a non-malleable SNARK scheme even if a random oracle is adopted.