

Simplified Square Span Program

1 Background

1.1 Span Programs

A Span Program (SP) is a linear-algebraic model of computation introduced by Karchmer and Wigderson[4][6].

Definition 1. A SP over a field \mathbb{F} consists of a nonzero target vector \mathbf{t} over \mathbb{F} , a set of vectors $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_m\}$, a partition of the indices $\mathcal{I} = \{1, \dots, m\}$ into two sets $\mathcal{I}_{labeled}$ and \mathcal{I}_{free} , and a further partition of $\mathcal{I}_{labeled}$ as $\cup_{i \in [n], j \in \{0,1\}} \mathcal{I}_{ij}$.

The SP is said to “compute” a function f if the following is true for all input assignment $u \in \{0,1\}^n$: the target vector is in the span of the vectors that “belong” to the input assignments u – namely, the vectors with indices in $\mathcal{I}_u = \mathcal{I}_{free} \cup_i \mathcal{I}_{i,u_i}$ – iff $f(u) = 1$. The size of the span program is m .

1.2 Circuit Checker Function

Suppose C is a Boolean circuit that computes a function f .

Definition 2. Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a function whose Boolean circuit C has s gates. Let $N = n + s$. Suppose $\phi : \{0,1\}^N \rightarrow \{0,1\}$ is a function that outputs ‘1’ iff the input is a valid assignment of C ’s wires (wires that fan out are considered one wire) with output wire set to ‘1’. We say that ϕ is the circuit checker function for f .

2 Verifiable Computation

A public verifiable computation (VC) scheme allows a computationally limited client to outsource the computation of a function F on input u to an untrusted worker, and then verify the correctness of the returned result $F(u)$. Critically, the outsourcing and verification procedures must be significantly more efficient for the client than performing the computation by itself.[7]

A public verifiable computation scheme provides *public delegation*, which allows arbitrary parties to submit inputs for delegation; and *public verifiability*, which allows arbitrary parties (not just the delegator) to verify the correctness of the results returned by the worker. The following definition captures these two properties.

Definition 3. A public verifiable computation scheme \mathcal{VC} consists of a set of three polynomial-time algorithms (KeyGen, Compute, Verify) defined as follows:

- $(EK_F, VK_F) \leftarrow \text{KeyGen}(F, 1^\lambda)$: The randomized key generation algorithm takes the function F to be outsourced and security parameter λ . It outputs a public evaluation key EK_F , and a public verification key VK_F
- $(y, \pi_y) \leftarrow \text{Compute}(EK_F, u)$: The deterministic worker algorithm uses the public evaluation key EK_F and input u . It outputs $y \leftarrow F(u)$ and a proof π_y of y 's correctness.
- $\{0, 1\} \leftarrow \text{Verify}(VK_F, u, y, \pi_y)$: Given the verification key VK_F , the deterministic verification algorithm outputs 1 if $F(u) = y$, and 0 otherwise.

3 Simplified Square Span Program

We define SSSP somewhat similarly to SSP.

Definition 4. A simplified square span program (SSSP) Q over the field \mathbb{F} consists of $m + 1$ polynomials $v_0(x), v_1(x), \dots, v_m(x)$ and a target polynomial $t(x)$ such that $\deg(v_i(x)) \leq \deg(t(x))$ for all $i = 0, \dots, m$.

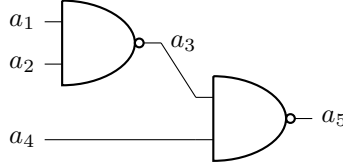
$$t(x) \text{ divides } \left(\sum_{i=0}^m a_i v_i(x) \right)^2 - 1$$

Specifically, $a_0 = 1$.

We say that Q verifies a boolean function $f : \{0, 1\}^l \rightarrow \{0, 1\}$ if it accepts exactly those inputs $\mathbf{a} \in \mathbb{F}^l$ that satisfy $\mathbf{a} \in \{0, 1\}^l$ and $f(\mathbf{a}) = 1$. We may see f as a binary circuit.

3.1 Transformation From Circuit Satisfiability to SSSPs

Consider a circuit consisting of two NAND gates, as is shown below:



It satisfies that

$$a_3 = \neg(a_1 \wedge a_2)$$

$$a_5 = \neg(a_3 \wedge a_4)$$

To guarantee $a_1, a_2, \dots, a_5 \in \{0, 1\}$, we use the constraints

$$(2a_i - 1)^2 = 1, \quad i = 1, 2, \dots, 5$$

To linearize[5] the NAND gate with input a_1, a_2 and output a_3 , writing \bar{c} for $1 - c$, we have

$$\begin{aligned}
a_3 = \neg(a_1 \wedge a_2) &\iff a_1 + a_2 - 2a_3 \in \{0, 1\} \\
&\iff (2(a_1 + a_2 - 2(1 - a_3)) - 1)^2 = 1 \\
&\iff (2a_1 + 2a_2 + 4a_3 - 5)^2 = 1
\end{aligned}$$

Similarly, we have

$$(2a_3 + 2a_4 + 4a_5 - 5)^2 = 1$$

The satisfiability of the circuit can therefore be represented by 7 quadratic equations:

$$\begin{aligned}
(2a_1 - 1)^2 &= 1 & (2a_2 - 1)^2 &= 1 & \dots & (2a_5 - 1)^2 &= 1 \\
(2a_1 + 2a_2 + 4a_3 - 5)^2 &= 1 \\
(2a_3 + 2a_4 + 4a_5 - 5)^2 &= 1
\end{aligned}$$

Corresponding to $(\mathbf{a}V)^2 = 1$.

Then we can represent the constraints as

$$\mathbf{a}V = (1, a_1, a_2, a_3, a_4, a_5) \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -5 & -5 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 2 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 & 4 \end{pmatrix}$$

To get a SSSP, let p be a prime and r_1, r_2, \dots, r_7 be 7 distinct elements in \mathbb{Z}_p . Pick degree 5 polynomials $v_0(x), v_1(x), \dots, v_5(x)$ such that

$$\begin{pmatrix} v_0(r_1) & v_0(r_2) & \dots & v_0(r_7) \\ v_1(r_1) & v_1(r_2) & \dots & v_1(r_7) \\ \vdots & \ddots & & \vdots \\ v_5(r_1) & & \ddots & v_5(r_7) \end{pmatrix} = V = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -5 & -5 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 2 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 & 4 \end{pmatrix}$$

Let $t(x) = (x-r_1)(x-r_2)\dots(x-r_7)$ to get a SSSP $(v_0(x), v_1(x), \dots, v_5(x), t(x))$ for the circuit such that

$$t(x) \text{ divides } \left(\sum_{i=0}^5 a_i v_i(x) \right)^2 - 1, \quad \text{where } a_0 = 1$$

If and only if a_1, a_2, \dots, a_5 satisfy the circuit.

4 Building Verifiable Computation from SSSPs

4.1 Bilinear group

We use the following notation[2]:

1. \mathbb{G} and \mathbb{G}_T are two (multiplicative) cyclic groups of prime order q .
2. G is a generator of \mathbb{G} .
3. e is a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. That is, for all $U, V \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, we have $e(U^a, V^b) = e(U, V)^{ab}$. We also require that $e(G, G)$ is a generator of \mathbb{G}_T .

We say that \mathbb{G} is a **bilinear group** if there exists a group \mathbb{G}_T and a bilinear map as above.

THE q -POWER KNOWLEDGE OF EXPONENT ASSUMPTION. The knowledge of exponent (KEA) assumption[3] says that given g, g^α it is infeasible to create c, \hat{c} such that $\hat{c} = c^\alpha$ without knowing a such that $c = g^a$ and $\hat{c} = (g^\alpha)^a$. Bellare and Palacio[1] extended this to the KEA3 assumption, which says that given $g, g^x, g^\alpha, g^{\alpha x}$ it is infeasible to create c, \hat{c} such that $\hat{c} = c^\alpha$ without knowing a_0, a_1 such that $c = g^{a_0}(g^x)^{a_1}$ and $\hat{c} = g^{\alpha a_0}(g^{\alpha x})^{a_1}$.

The q -power knowledge of exponent assumption is a generalization of KEA and KEA3 in bilinear groups. It says that given $(g, g^x, \dots, g^{x^q}, g^\alpha, g^{\alpha x}, \dots, g^{\alpha x^q})$ it is infeasible to create c, \hat{c} such that $\hat{c} = c^\alpha$ without knowing a_0, \dots, a_q such that $c = \prod_{i=0}^q (g^{x^i})^{a_i}$ and $\hat{c} = \prod_{i=0}^q (g^{\alpha x^i})^{a_i}$.

We will now give the formal definition of the q -power knowledge of exponent assumption. We write $(y; z) \leftarrow (\mathcal{A} \parallel \mathcal{X}_\mathcal{A})(x)$ when \mathcal{A} on input x output y and $\mathcal{X}_\mathcal{A}$ on the same input output z .

Definition 5. (q -PKE). *Let k be a security parameter. The $q(k)$ -power knowledge of exponent assumption holds for \mathcal{G} if for every non-uniform probabilistic polynomial time adversary \mathcal{A} there exists a non-uniform probabilistic polynomial time extractor $\mathcal{X}_\mathcal{A}$ such that*

$$Pr \left[\begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathbb{G}(1^k); g \leftarrow \mathbb{G} \setminus \{1\}; \alpha, s \leftarrow \mathbb{Z}_p^*; \\ \sigma = (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, \dots, g^{s^q}, g^\alpha, g^{\alpha s}, \dots, g^{\alpha s^q}); \\ (c, \hat{c}; a_0, \dots, a_q) \leftarrow (\mathcal{A} \parallel \mathcal{X}_\mathcal{A})(\sigma) : \hat{c} = c^\alpha \wedge c \neq \prod_{i=0}^n g^{a_i s^i} \end{array} \right] \approx 0.$$

for any auxiliary information $z \in \{0, 1\}^{\text{poly}(k)}$ that is generated independently of α .

The q -PKE assumption seems plausible even if the extractor is prohibited from viewing some computations, as long as those computations are not dependent on α . In our statement of the assumption, we capture this notion by providing \mathcal{A} and $\mathcal{X}_\mathcal{A}$ with auxiliary information $z \in \{0, 1\}^{\text{poly}(k)}$. \mathcal{A} and $\mathcal{X}_\mathcal{A}$ do not know how z was generated, but they are given the assurance that is not dependent on α . This assurance seems necessary: e.g., if z included an encoded pair

$(r, r\alpha)$ whose representation with respect to $\{s^i\}$ is unknown, then we could not hope for an efficient extractor \mathcal{X}_A , assuming problems such as discrete-log that underlie the encoding scheme are hard. Note that we do not require that z be independent of s . Even if z includes the actual value s , q -PKE still seems plausible.

THE q -POWER DIFFIE-HELLMAN ASSUMPTION. The q -power Diffie-Hellman assumption says given $g, g^s, \dots, g^{s^q}, g^{s^{q+2}}, \dots, g^{s^{2q}}$ it is hard to compute the missing element $g^{s^{q+1}}$.

Definition 6. (q -PDH). *Let k be a security parameter. The $q(k)$ -power Diffie-Hellman assumption holds for \mathcal{G} if for all non-uniform probabilistic polynomial time adversaries \mathcal{A} we have*

$$\Pr \left[\begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathbb{G}(1^k); g \leftarrow \mathbb{G} \setminus \{1\}; s \leftarrow \mathbb{Z}_p^*; \\ \sigma = (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, \dots, g^{s^q}, g^{s^{q+2}}, \dots, g^{s^{2q}}); \\ y \leftarrow (\mathcal{A})(\sigma) : y = g^{s^{q+1}} \end{array} \right] \approx 0.$$

4.2 Verifiable Computation from SSSPs

The following construction shows how to instantiate a verifiable computation scheme from a SSSP for any functions l_u, l_w and families $\{\mathcal{R}\}_\lambda$ of relations R of pairs $(u, w) \in \{0, 1\}^{l_u(\lambda)} \times \{0, 1\}^{l_w(\lambda)}$ that can be computed by polynomial size circuits with $m(\lambda)$ wires and $n(\lambda)$ gates for a total size of $d(\lambda) = m(\lambda) + n(\lambda)$.

- $(\sigma, \tau) \leftarrow \text{Setup}(1^\lambda, R)$: Run $gk := (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$. Parse R as a boolean circuit $C_R : \{0, 1\}^{l_u} \times \{0, 1\}^{l_w} \rightarrow \{0, 1\}$. Generate a SSSP $Q = (v_0(x), \dots, v_m(x), t(x))$ that verifies C_R over \mathbb{Z}_p . Pick $G \leftarrow \mathbb{G}^*$ and $\beta, s \leftarrow \mathbb{Z}_p^*$ such that $t(s) \neq 0$. Return

$$\begin{aligned} \sigma &= (gk, G, \dots, G^{s^d}, \{G^{\beta v_i(s)}\}_{i > l_u}, G^{\beta t(s)}, Q) \\ \tau &= (\sigma, \beta, s) \end{aligned}$$

- $\pi \leftarrow \text{Prove}(\sigma, u, w)$: Parse u as $(a_1, \dots, a_{l_u}) \in \{0, 1\}^{l_u}$ and use w to compute a_{l_u+1}, \dots, a_m such that $t(x)$ divides $\left(\sum_{i=0}^m a_i v_i(x) \right)^2 - 1$.

Let

$$h(x) = \frac{\left(\sum_{i=0}^m a_i v_i(x) \right)^2 - 1}{t(x)}$$

Use linear combinations of the elements in σ to compute

$$\begin{aligned} H &= G^{h(s)} & V_w &= G^{\sum_{i > l_u}^m a_i v_i(s)} \\ B_w &= G^{\beta \left(\sum_{i > l_u}^m a_i v_i(s) \right)} & V &= G^{\sum_{i=0}^m a_i v_i(s)} \end{aligned}$$

and return $\pi = (H, V_w, B_w, V)$.

- $\{0, 1\} \leftarrow \text{Verify}(\sigma, u, \pi)$: Parse u as $(a_1, \dots, a_{l_u}) \in \{0, 1\}^{l_u}$ and π as $(H, V_w, B_w, V) \in \mathbb{G}^4$. Return 1 if and only if

$$e(H, G^{t(s)}) = e(V^2/G) \quad e(V_w, G^\beta) = e(B_w, G).$$

5 Security

Here we prove the security of the ... The intuition is that it is hard for the prover, who knows ... but not ..., to output any ... unless the prover knows Knowledge of exponent assumptions (KEAs) formalize this intuition: they say that for any algorithm that outputs a pair of encoded elements with ratio ..., there is an extractor that "watches" the algorithm's computation that outputs the representation (linear combination). In the security proof, extractors for the ... terms extract out polynomials ... that are in the spans of If the proof verifies, then...

References

1. Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 48–62. Springer, 2004.
2. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.
3. Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *Annual International Cryptology Conference*, pages 445–456. Springer, 1991.
4. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizes without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.
5. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):11, 2012.
6. Mauricio Karchmer and Avi Wigderson. On span programs. In *Structure in Complexity Theory Conference, 1993., Proceedings of the Eighth Annual*, pages 102–111. IEEE, 1993.
7. Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference*, pages 422–439. Springer, 2012.