A High-Speed Architecture for the Reduction in VDF Based on a Class Group

Yifeng Song, Danyang Zhu, Jing Tian, and Zhongfeng Wang
School of Electronic Science and Engineering, Nanjing University, Nanjing, China
Email: 141120091@smail.nju.edu.cn,zhudanyang10@foxmail.com, jingtian_nju@sina.com, zfwang@nju.edu.cn

***Abstract*—Due to the enormous energy consuming involved in the proof of work (POW) process, the resource-efficient blockchain system is urged to be released. The verifiable delay function (VDF), being slow to compute and easy to verify, is believed to be the kernel function of the next-generation blockchain system. In general, the reduction over a class group, involving many complex operations, such as the large-number division and multiplication operations, takes a large portion in the VDF. In this paper, for the first time, we propose a high-speed architecture for the reduction by incorporating algorithmic transformations and architectural optimizations. Firstly, based on the fastest reduction algorithm, we present a modified version to make it more hardware-friendly by introducing a novel transformation method that can efficiently remove the large-number divisions. Secondly, highly parallelized and pipelined architectures are devised respectively for the large-number multiplication and addition operations to reduce the latency and the critical path. Thirdly, a compact state machine is developed to enable maximum overlapping in time for computations. The experiment results show that when computing 209715 reduction steps with the input width of 2048 bits, the proposed design only takes 137.652ms running on an Altera Stratix-10 FPGA at 100MHz frequency, while the original algorithm needs 3278ms when operating over an i7-6850K CPU at 3.6GHz frequency. Thus we have obtained a drastic speedup of nearly 24x over an advanced CPU.**

***Index Terms*—Verifiable delay function, blockchain, reduction, hardware architecture, FPGA**

I. Introduction

A verifiable delay function (VDF) is a function which is slow to evaluate and easy to verify [1], [2]. A number of specified sequential steps are required to acquire the evaluation result, while the verification procedure is efficient and public. There are a lot of applications of VDF in decentralized systems, such as generating a verifiable randomness beacon in a trust-less environment and constructing resource-efficient blockchains. For instance, a classic approach to get a randomness beacon is to apply an extractor function to a public entropy source, such as stock prices [2]. This approach is unreliable when the attacker tries to control the beacon results by influencing the stock price according to their simulations. Due to the heavy computation of the evaluation function, a VDF can be applied to make it unachievable to get the simulation results before the beacon is announced. As for the resource-efficient blockchain, also called next-generation blockchain, is much different from the proof-of-work (POW) blockchain like Bitcoin or Ethereum. POW blockchains, which consume a large (and growing) amount of energy, are gradually replaced by some resource-efficient blockchains such as the proof-of-stake [3]–[7], proof-of-storage [8], and proof-of space [9], [10] blockchains. VDFs are applied to these projects to resist simulation attacks from which the resource-efficient blockchains suffer a lot. Some other applications of VDFs were also introduced in [2].

Two brilliant VDFs are proposed recently by Wesolowski and Pietrzak respectively [1], [11], [12]. The evaluation functions are the same in both schemes and the verification procedures are different. Both of them construct evaluation functions with repeated squaring in a group of unknown order, which is believed cannot be sped up in parallel. This is what we focus on in this paper, openly proposing the fastest hardware design to prevent the blockchain systems from being attacked. Benefiting from the convenience of generating the appropriate parameters without any trusted party, using a class group of Binary quadratic forms is considered as a good choice for evaluation. The reduction and squaring of binary quadratic forms are the computations in the evaluation function. For the original version of the evaluation, 90% of the time is taken by the reduction operations. Thanks to the competition hosted by the CHIA company, many simplifications for the reduction algorithm have been proposed. By approximating the large integers using small (like 64-bit) numbers, Akashnil Dutta improved the reduction by almost 5x [13]. With the help of the fast reduction algorithm, reduction time is decreased to about 60% of the overall time budget. However, there are still exist large integer divisions, which are not friendly for hardware implementation.

In order to make the VDF more effective, a modified fast reduction algorithm and the corresponding hardware architecture are proposed in this paper. By introducing a group of selective parameters, large-number divisions are discarded, which makes it more friendly for hardware implementation. Highly-parallelized architectures are also devised for large-number multiplication and addition to reduce the long latency and the critical path. Moreover, a compact state machine is developed to enable maximum overlapping in time for computations.

The rest of this work is structured as follows. Section II summaries the definition of VDF, binary quadratic forms,

This work was supported by the National Natural Science Foundation of China under Grant 61604068, the Fundamental Research Funds for the Central Universities under Grant 021014380065, the Key Research Plan of Jiangsu Province of China under Grant BE2019003-4.

and the fast reduction algorithm. The proposed modified fast reduction algorithm is presented in Section III. Hardware architectures are shown in Section IV. Experiment results and comparisons are shown in Section V. Finally, Section VI draws the conclusion.

II. Background

A. VDF

A verifiable delay function (VDF) is a function $f : \mathcal{X} \to \mathcal{Y}$ that takes a prescribed time to compute and cannot be parallelly processed. However, the output can be quickly verified by public when the result is computed. Moreover, every input $x \in \mathcal{X}$ must have a unique valid output $y \in \mathcal{Y}$. In more detail, a VDF consists of three algorithms: $Setup(\lambda, T)$, $Eval(pp, x)$, and $Verify(pp, x, y, \pi)$.

- $Setup(\lambda, T)$ takes a security parameter λ and a time bound T, and generates public parameters pp.
- $Eval(pp, x)$ takes a number of sequential steps to output a y and a proof π.
- $Verify(pp, x, y, \pi)$ outputs $accept$ if y is indeed the correct output for x, or outputs $reject$.

Repeated squaring in a group of unknown order is the best-known method for achieving a non-parallelizable sequential operation in the evaluation function [14]. An unknown group order prevents the repeated squaring computation being shorted by modulo the order of the group. The traditional RSA group scheme uses a multiplicative group $\mathbb{Z}/N$, where N is the product of two unknown primes. When $N = pq$, the order of the group equals to $(p-1)(q-1)$. It is extremely difficult to calculate the factors of N, so the group order is considered as unknown. However, a trusted third-party is needed to generate the order N and destroy the prime factors once N is created. Therefore, class groups of binary quadratic forms are presented to get a group of unknown order more effectively and more securely, without any trusted party.

B. Binary Quadratic Forms

Consider a class group of a negative prime discriminant d, where $|d| \equiv 3 \bmod 4$. When $|d|$ is sufficiently large, the order of the group is so complicated to be computed that it can be considered as unknown. The class group is explained clearly in [14], [15], and we only introduce several basic concepts and algorithms which are needed in our work.

- A binary quaratic form is $f(x, y) = ax^2 + bxy + cy^2$, where $a, b, c \in \mathbb{R}$ and they are not all equal to zero. And $f = (a, b, c)$ is called a form. The discriminant of a form is $\Delta(f) = b^2 - 4ac$.
- For a given $f(a, b, c)$ and n, a solution $(x, y) \in \mathbb{Z}^2$ to the equation $n = ax^2 + bxy + cy^2$ is called a representation of n by f.
- A form $f = (a, b, c)$ is called normal if $-a < b \leq a$. If $f(a, b, c)$ is normal and $a \leq c$ (if $a = c$ then $b \geq 0$), f is called reduced.

The normalization algorithm and reduction algorithm are shown in Alg. 1 and Alg. 2, respectively. A intermediate value r is computed with a and b, and is used to update $f(a, b, c)$ to a normalization form. Similar operation is repeated after a normalization to get the reduced form.

Algorithm 1 Normalization

Input: $f(a, b, c)$, $\Delta < 0$, $a > 0$
Output: $f(a, b, c)$, $-a < b \leq a$
1: $r = \lfloor \frac{a-b}{2a} \rfloor$;
2: $\eta(f) = (a, b + 2ra, ar^2 + br + c)$;
3: Update $f = \eta(f)$

Algorithm 2 Reduction

Input: $f(a, b, c)$, $\Delta < 0$, $a > 0$
Output: $f(a, b, c)$, $-a < b \leq a$, $a \leq c$, and if $a = c$ then $b \geq 0$
1: Normalize f;
2: **repeat**
3: $s = \lfloor \frac{c+b}{2c} \rfloor$;
4: $\rho(f) = (c, -b + 2sc, cs^2 - bs + a)$;
5: Update$f = \rho(f)$;
6: **until** f is reduced

C. Fast Reduction Algorithm

Due to the heavy computation of large-number division and multiplication, the reduction of a binary quadratic form is extremely slow to compute. In the repeated squaring procedure, almost 90% of the time is taken in the reduction steps. Therefore, a competition was hosted by the CHIA company to find the simplification algorithm for reduction. The fast reduction algorithm proposed by Akashnil Dutta stood out in the first round competition. The main idea of this algorithm is using small numbers to simulate the large numbers and performing reduction on those small numbers. Intermediate variables are generated in the small reduction loops, and the large numbers are updated with these variables when the small reductions are done. These steps are repeated until a reduced form is gotten. The detailed algorithm is depicted in Alg. 3.

Algorithm 3 Fast Reduction

Input: $f(a, b, c)$,$\Delta < 0$, $a > 0$
Output: $f(a, b, c)$, $-a < b \leq a$, $a \leq c$ and if $a = c$ then $b \geq 0$
1: **if** f is reduced **then**
2: Return f;
3: **else**
4: **repeat**
5: Effective bits of a, b, c are computed and denoted as a_num, b_num and c_num.
6: The maximun number and the minimum one are chosen as max_num and min_num.
7: **if** $max\_num - min\_num > 31$ **then**
8: Normalize f;
9: **else**
10: (x, y, z) are generated by extracting the most significant 64 bits of (a, b, c).

$x \gg (max\_num - a\_num + 1)$;
$y \gg (max\_num - b\_num + 1)$;
$z \gg (max\_num - c\_num + 1)$;
Set $u = 1, v = 0, m = 0, n = 1$;
repeat
$\delta = y \geq 0\ ?\ (b+c)/(2c) : -(-b+c)/(2c)$
$(x', y', z') = (z, -y + 2\delta z, x - \delta y + \delta^2 z)$
$(u', v', m', n') = (v, -u + \delta v, n, -m + \delta n)$
Update(x, y, z) and (u, v, m, n);
until $x < z \parallel z > 0$
$a' = u^2 a + umb + m^2 c$;
$b' = 2uva + (un + vm)b + 2mnc$;
$c' = v^2 a + vnb + n^2 c$;
Update (a, b, c);
end if
until f is reduced
end if

There are some simple transformations when testing whether $f(a, b, c)$ is reduced in the fast reduction algorithm, which is shown in Alg. 4.

Algorithm 4 Test Reduction

Input: $f(a, b, c)$
Output: $f(a, b, c)$, $check$
if $|a| < |b| \parallel |c| < |b|$ **then**
Return $f(a, b, c)$ and $check = 0$
else
if $a > c$ **then**
$f(a', b', c') = f(c, -b, a)$
else if $a == c$ && $b < 0$ **then**
$f(a', b', c') = f(a, -b, c)$
else
$f(a', b', c') = f(a, b, c)$
end if
Return $f(a', b', c')$ and $check = 1$
end if

According to Akashnil's test, reduction steps in repeated squaring are sped up by almost 5 times by applying the fast reduction algorithm [13]. The reduction time decreases to about 60% of the overall time budget, which is the fastest scheme in the competition. Therefore, we choose this algorithm to implement our hardware accelerator.

III. Modified Fast Reduction Algorithm

The key advantage of fast reduction algorithm is that the multiplications and divisions of large numbers are decreased. Multiplication can be accelerated by parallel processing. But the division of large numbers is really unfriendly for hardware implementation. Hence, a modified fast reduction (MFR) algorithm is proposed to remove large number divisions and make the algorithm more friendly for hardware acceleration. We notice that the function of normalization in the fast reduction algorithm is to make variables (a, b, c) closer to each other. Therefore, we choose a group of specified parameters to update these variables, which achieves the same target without any division of large numbers. Only some large-number additions and subtractions are involved. These can also be accelerated by parallel processing. The detailed algorithm is depicted in Alg. 5.

Algorithm 5 Modified Fast Reduction

Input: $f(a, b, c)$,$\Delta < 0$, $a > 0$
Output: $f(a, b, c)$, $-a < b \leq a$, $a \leq c$ and if $a = c$ then $b \geq 0$
Test whether f is reduced.
if f is reduce **then**
Return f;
else
repeat
Effective bits of a, b, c are computed and denoted as a_num, b_num and c_num.
The maximun number and the minimum one are chosen as max_num and min_num.
if $max\_num - min\_num > 31$ **then**
$(a', b', c') = (c, c - b, a - b + c)$
Update(a, b, c);
else
(x, y, z) are generated by extracting the most significant 64 bits of (a, b, c).
$x \gg (max\_num - a\_num + 1)$;
$y \gg (max\_num - b\_num + 1)$;
$z \gg (max\_num - c\_num + 1)$;
Set $u = 1, v = 0, w = 0, x = 1$;
repeat
$\delta = y \geq 0\ ?\ (b+c)/(2c) : -(-b+c)/(2c)$
$(x', y', z') = (z, -y + 2\delta z, x - \delta y + \delta^2 z)$
$(u', v', m', n') = (v, -u + \delta v, n, -m + \delta n)$
Update(x, y, z) and (u, v, m, n);
until $x < z \parallel z > 0$
$a' = u^2 a + umb + m^2 c$;
$b' = 2uva + (un + vm)b + 2mnc$;
$c' = v^2 a + vnb + n^2 c$;
Update (a, b, c);
end if
until f is reduced
end if

IV. The Proposed Architecture

Based on the modified fast reduction algorithm (MFR) shown in Alg. 5, A well-designed hardware architecture is proposed in this section. Fig. 1 shows the top-level architecture of the MFR.

The whole architecture is composed of four main modules: 1) Test-Reduction; 2) Pre-Calculation; 3) App-Reduce; and 4) Tran-Calculation. The Test-Reduction module tests whether the input form is reduced. The Pre-Calculation module counts the effective bits and extracts the most significant 64 bits to represent the large number. When the signal $Check$ equals "1", these small numbers from the Pre-Calculation module are sent to App-Reduce module to operate the reduction algorithm for 64-bit numbers. The Tran-Calculation module updates the old

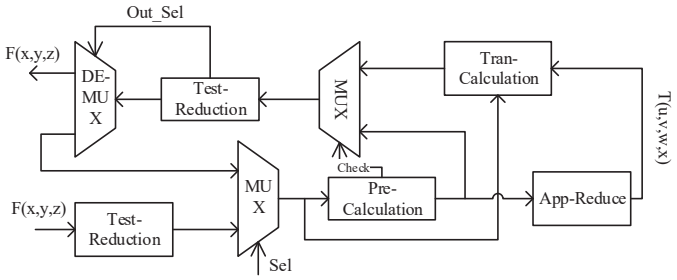

Fig. 1. Top-Level Entity

form using the transformation parameters $T(u, v, w, x)$ from the App-Reduce module. These four modules will be detailed in the following.

Test-Reduction In this module, the carry select adder(CAS) [16] architecture is adopted to reduce the critical path. For instance, the 2048-bit addition is divided into 32 64-bit additions, which does not cause too much delay. As shown in Fig. 2, $a_i + b_i$ and $a_i + b_i + 1$ are calculated at the same time. The final result add_i is decided by $c_i - 1$ which is the highest bit of add_{i-1}. After all mid-values are calculated, the sign bits of the result are generated by adding the sign bit of a, the sign bit of b, and the possible carry bit from the previous adder. The total delay of this architecture is one 64-bit adder, 31 64-bit multiplexers (MUXs), and one full adder, which is much faster than a large 2049-bit adder.

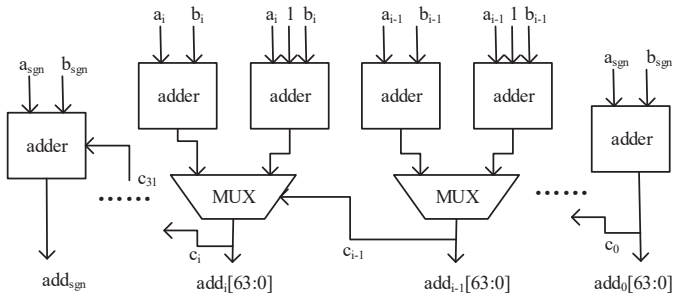

Fig. 2. Carry Select Adder

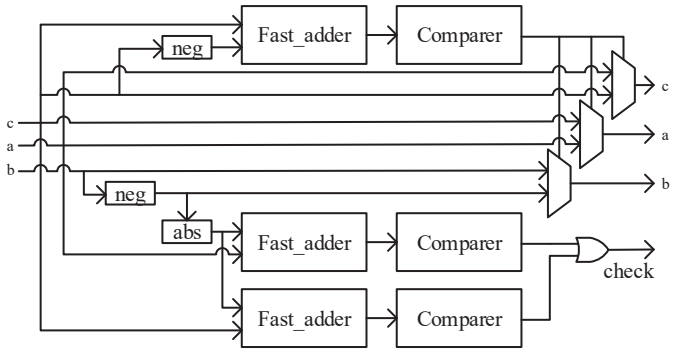

Fig. 3. Test Reduction

The top-level architecture of this module is shown in Fig. 3. A comparer is placed after the fast addition unit, which outputs "0" when the input equals to 0, outputs "1" when the MSB of input is 0, and outputs "2" when the most significant bit (MSB) is 1. According to the pseudo-code in Alg. 4, the outputs of these comparators are used with some MUXs to generate the final results.

Pre-Calculation The most significant bits of the triple-inputs should be firstly confirmed in this module. As shown in Fig. 4, a tree structure is adopted to find the MSB more efficiently. The whole procedure is cut into three stages to shrink calculation loads. In the first stage, the first nonzero element of 32 64-bit numbers is found and decomposed into 16 4-bit integers. In the second stage, the first nonzero 4-bit is acquired. And in the third stage, a small size lookup table is used to locate the exact MSB. At last, the numbers of effective bits are calculated by two shift operators and two adders.

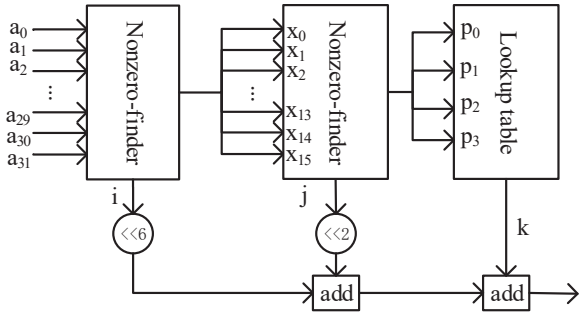

Fig. 4. Bit-Counter

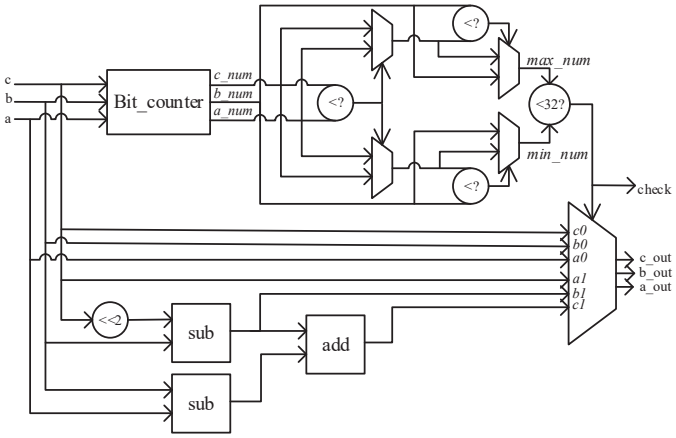

Fig. 5. Pre-Calculation

Fig. 5 shows the top-level architecture of the whole module. Bit-counter generates the lengths of these three inputs. The maximum one and the minimum one of (a_num, b_num, c_num) can be found out through four MUXs and three comparers. If $max\_num - min\_num > 31$, the signal $check$ will be set high. $4c-b$ and $a-b$ are calculated in parallel for high-speed design, and the results are added up to obtain c_1. $f(a_0, b_0, c_0)$ or $f(a_1, b_1, c_1)$ are chosen to be

output as $f(a\_out, b\_out, c\_out)$ by the signal $check$ through a MUX.

APP-Reduce A loop architecture is adopted in this module, which is shown in Fig. 6. $c - b$ and $c + b$ are calculated in parallel, and the results are selected out by the sign bit of b. The middle variable $delta$ is generated by a 64-bit divider with pipelines inserted. Then, $f(a, b, c)$ and $T(u, v, w, x)$ are updated with $delta$ using several adders and multipliers. Before the "App-reduce" is done, new $f(a, b, c)$ will be sent to the input MUX to get into loop again. a and c are compared to obtain the signal $done$, and $T(u, v, w, x)$ is exported once the signal $done$ gets high.

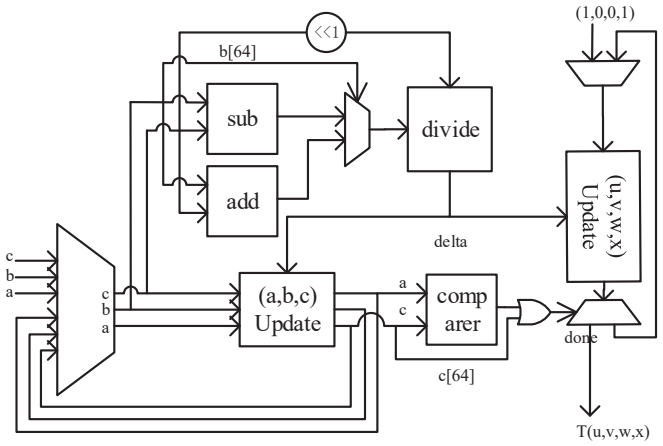

Fig. 6. App-Reduce

Tran-Calculation As declared by the pseudo-code in Alg. 5, multipliers for large numbers and the three-input adders for large numbers are needed in this module. The adder is similar to the architecture in Fig. 2, and small three-input adders are used to calculate the partial sum. Both $a_i + b_i + c_i$ and $a_i + b_i + c_i + 1$ are generated, and the exact partial sum is selected by the propagating carry with a series of MUXs. As for the large-number multiplier, the Karatsuba scheme is adopted. A large-number multiplication is divided into several small multiplications. Then the additions of partial products are performed with a carry-select architecture which is similar to the fast-add module.

V. Experiment Results and Comparison

TABLE I
Experiment Results of 209715 groups of Input for the Reduction

	Total Time	Loops
MFR (proposed)	3232ms	7230871
Fast Reduction	3278ms	7229027
Proportion	98.60%	100.03%

Due to the fact that there is no existing hardware implementation in the open literature for the reduction of binary quadratic forms, we compare our scheme with software implementation. We test Akashnil's code on a server with i7-6850K @3.6GHZ for comparison. Based on Akashnil's platform, our algorithm is also deployed on the same server. 209715 groups of input are used for the test. As shown in Table I, the MFR takes 3232ms while the original algorithm needs 3278ms. And, MFR takes 7230871 loops in total while Akashnil's code needs 7229027 loops. Benefiting from discarding the large-number division, our algorithm is a little faster than the fast reduction algorithm by 1.40%, although a little more loops are needed.

We code our architecture with RTL and implement it on quartus 18.0 platform with 1SM21BHU2F53E2VGS1 board. Table II shows the implementation results. ALM utilization is 189938, which takes about 27% resource of the board. Besides, 1483 DSPs and 333022 registers are consumed, which take 27% and 12% respectively. We get the timing closure at 100MHz and test the same work as what we have done with software. It takes 137.652ms to finish 209715 times of reductions, which is 24x faster than the software.

TABLE II
Implementation Results

Resource	Number	Total	Percentage
ALMs	189938	702720	27%
Registers	333022	2810880	12%
DSPs	1483	3960	37%

VI. Conclusion

In this paper, we propose a modified fast reduction algorithm for the VDF, which can effectively remove the large-number divisions. Software simulation shows that the proposed algorithm achieves a slight better performance than the state-of-art algorithm. Moreover, a well-designed hardware architecture based on the proposed reduction algorithm is also proposed. The multiplication and addition operations are highly parallelized and a compact state machine is presented. The experiment results show that when computing 209715 reduction steps with the input width of 2048 bits, the proposed design only takes 137.652ms running on the Altera Stratix-10 FPGA at 100MHz frequency, while the original algorithm needs 3278ms when operating over the i7-6850K server at 3.6GHz frequency. Thus we have obtained a drastic speedup of nearly 24 times over an advanced CPU.

References

[1] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions." *IACR Cryptology ePrint Archive*, vol. 2018, p. 712, 2018.

[2] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[3] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, vol. 2016, no. 919, 2016.

[4] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[5] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[6] S. King and S. Nadal, "Peercoin–secure & sustainable cryptocoin," *Aug-2012 [Online]. Available: https://peercoin. net/whitepaper ()*, 2012.

[7] S. Micali, "Algorand: the efficient and democratic ledger," *arXiv preprint arXiv:1607.01341*, 2016.

[8] S. Park, A. Kwon, G. Fuchsbauer, P. Gaži, J. Alwen, and K. Pietrzak, "Spacemint: A cryptocurrency based on proofs of space," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 480–499.

[9] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 475–490.

[10] Filecoin, "A decentralized storage network." Website, https://filecoin.io/filecoin.pdf.

[11] K. Pietrzak, "Simple verifiable delay functions," in *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[12] B. Wesolowski, "Efficient verifiable delay functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 379–407.

[13] A. Dutta, "Fast reduce," Website, https://github.com/Akashnil/chia-vdf-competition/tree/master/Entry1.

[14] L. Long, "Binary quadratic forms," Website, https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf.

[15] J. Buchmann and U. Vollmer, "Binary quadratic forms," in *Binary Quadratic Forms*. Springer, 2007, pp. 9–20.

[16] A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Trans. Comput.*, vol. 42, no. 10, p. 1163–1170, Oct. 1993. [Online]. Available: https://doi.org/10.1109/12.257703