An Efficient Hash Function for Imaginary Class Groups

Kostas Kryptos Chalkias[0000−0002−3252−9975],
Jonas Lindstrøm[0000−0002−1989−3019], and
Arnab Roy[0009−0005−3770−9982]

Mysten Labs
{kostas,jonas,arnab}@mystenlabs.com
mystenlabs.com

Abstract. This paper presents a new efficient hash function for imaginary class groups. Many class group based protocols, such as verifiable delay functions, timed commitments and accumulators, rely on the existence of an efficient and secure hash function, but there are not many concrete constructions available in the literature, and existing constructions are too inefficient for practical use cases.
Our novel approach, building on Wesolowski's initial scheme, achieves a 200 fold increase in computation speed, making it exceptionally practical for real-world applications. This optimisation is achieved at the cost of a smaller image of the hash function, but we show that the image is still sufficiently large for the hash function to be secure.

Keywords: Cryptographic Hash Functions · Imaginary class groups · Class group cryptography · Implementations · Verifiable delay functions · Accumulators · Timed commitments

1 Introduction

Imaginary class groups have recently become a focal point in cryptographic research due to a unique property: their order remains elusive. Recall that the order of an imaginary class group with a given discriminant is known as the *class number* and it is believed to be difficult to compute for large discriminants, which is why we may assume that the order of a class group, even if we know the discriminant, is unknown.

Assuming the factorization is not known, the order of an RSA group is also unknown, but the benefit of class groups over RSA groups is that sampling a new group with unknown order is easier for class groups, because one can simply sample a sufficiently large negative, prime discriminant Δ and publish it. For RSA groups, sampling a new group is much more difficult to do because it requires sampling a modulus $N = pq$ where p and q are two prime factors, and if you know these you also know the group order. So in RSA groups, for a trusted party to sample a group with unknown order *without*, more sophisticated and computationally expensive protocols are required, such as secure multi-party

computation (MPC), making class groups a more practical choice for certain cryptographic applications. See [8] for a recent example of this.

Groups of unknown order are used in many applications, most famously in the RSA signature scheme [1], but also for some implementations of accumulators [6], verifiable delay functions ([20,16]), and polynomial commitments [7]. We also note that specific groups of unknown order, including class groups, cannot replace RSA in applications which require the group order as a trapdoor - some examples are RSA-based time-lock puzzles [17], and random beacons [5].

When class groups are used in cryptography, sampling or hashing to a random element is an important primitive. This is for instance the case for some digital signature schemeas, where the plaintext has to be mapped to a group element, or for verifiable delay functions where a random input has to be sampled for each VDF instance.

Until recently, there have not been many concrete examples of hash functions for class groups. Wesolowski [20] presented a construction where the first coefficient a is restricted to be a prime. Recently, Seres, Burcsi and Kutas [18] have presented a set of new hash function schemes and have also shown that some previous constructions are insecure. Their work focuses in particular on how to construct a hash function with a uniform output in the class group and uses Bach's algorithm [3] to sample quadratic forms with composite a coefficients.

In [20], Wesoloski calls for optimizations to his hash construction, and this paper is a response to this. Our proposed scheme is significantly more efficient than both Wesolowski's construction and the constructions presented by Seres et al.

After a brief introduction to imaginary class groups in Section 2, we present the algorithm in Section 4 and prove the security of the hash function by proving that its image is uniform in a sufficiently large subgroup of a class group, and analyze its time complexity. The algorithm is implemented as part of the open-source *fastcrypto* Rust library and is used in the implementation of verifiable delay functions on the Sui blockchain. Finally, Section 5 discusses implementation remarks and benchmarks, before presenting future work developments in Section 7.

2 Background

We will only provide a few facts and definitions about imaginary class groups here and refer to Chapter 5 in [10] for a thorough introduction. The imaginary class group $Cl(\Delta)$ with discriminant $\Delta < 0$ where $\Delta \equiv 1 \bmod 4$ and $|\Delta|$ is prime may be represented by all quadratic forms $ax^2 + bxy + cy^2$, which we will write (a, b, c), with $\Delta = b^2 - 4ac$ where the group operation is known as *composition* of forms which was discovered by Gauss [13]. In this representation, two quadratic forms f and g represent the same group element if they are equivalent, e.g. if there is a matrix $U \in SL_2(\mathbb{Z})$ such that $f = g \circ U$. Each equivalence class contains exactly one *reduced* form so we use reduced quadratic forms to represent class group elements:

Definition 1 (Reduced quadratic form). *A quadratic form (a, b, c) with $a > 0$ and discriminant $\Delta = b^2 - 4ac$ is said to be* reduced *if $|b| \leq a \leq c$ and if $a \in \{|b|, c\}$ implies $b \geq 0$.*

As discussed in the introduction, the order of a class group is hard to compute, but we get as a consequence of Dirichlet's class group formula an approximation, namely that $\#Cl(\Delta) \approx \sqrt{|\Delta|}$. The size of the discriminant for cryptographic protocols will typically be at least 1024 bits, but a recent paper by Dobson, Galbraith and Smith [12] suggests that discriminants should be thousands of bits to ensure that it is sufficiently hard to find the order of a class group for a random discriminant of a given size.

The only result we need about quadratic forms is the following lemma which ensures that the output of our algorithms are reduced.

Lemma 1. *If $a < \frac{\sqrt{|\Delta|}}{2}$ and $-a < b \leq a$ then (a, b, c) is reduced.*

Proof. By assumption, we have $|b| \leq a$, so we just need to consider the bound on c, but this is true because

$$c = \frac{b^2 + |\Delta|}{4a} \geq \frac{|\Delta|}{4a} > \frac{a^2}{a} = a.$$

This also proves that $a \neq c$. We may still have $a = b$, but by assumption this only happens when $b \geq 0$ which proves that (a, b, c) is reduced.

3 Preliminaries

For multiple constructions we will need a function which samples a uniformly random integer in a range, that is a function $\textsc{RandomNumber}(C, \mathsf{seed})$ which is modelled as a random oracle on $\{1, \ldots, C-1\}$ for $\mathsf{seed} \in \{0,1\}^*$.

We assume there is a function $\textsc{ModSqrt}(k, m)$ for positive integers k and m, m being a prime number, which returns the smallest s such that $s^2 \equiv k \bmod m$ with $0 \leq s < m$, assuming this exists. In particular, it is the smallest s and $m - s$ for any square root $0 \leq s < m$.

Recall that $\pi : \mathbb{N} \to \mathbb{N}$ is the *prime-counting* functions such that $\pi(n) = \#\{p \leq n \mid p \text{ prime}\}$. We are also interested in the inverse of this, namely the n'th prime, which by the prime number theorem is approximately

$$\tilde{p}(n) := \lfloor n \ln n \rfloor.$$

4 Hash to class group

We now describe a hash algorithm and discuss its complexity and parallelizability.

4.1 Description

Our algorithm is an extension of Wesolowski's construction from [20]. The original construction samples a uniformly random prime a such that $a < \sqrt{|\Delta|}/2$ and $\left(\frac{\Delta}{a}\right) = 1$. This ensures that we can find a square root b such that $b^2 \equiv \Delta \bmod a$. If b is odd, $b^2 \equiv 1 \bmod 4$, so $4a$ divides $b^2 - \Delta$. If not, we can use the other square root, $a - b$, in place of b and compute the exact quotient $c = (b^2 - \Delta)/4a$ and return a quadratic form (a, b, c) with discriminant Δ. Lemma 1 ensures that the result is reduced.

Our algorithm is a natural extension to this construction where we first determine an upper bound on the size of the prime from a security parameter λ and then sample smaller primes and use their product as a.

The bottleneck for Wesolowski's construction is sampling random primes, and using a product of smaller primes allows for a speed-up and efficient parallelization at the cost of reducing the size of the image and also any claims of uniformity of the image, but the output is large enough for the hash function to be collision resistant, which we think will be sufficient for many use cases.

Algorithm 1 HASH2PRIME(l, q, seed)

Require: $l > 3$, d prime, $\mathsf{seed} \in \{0,1\}^*$.
Ensure: p prime with $\left(\frac{q}{p}\right) = 1$ and $p \leq 2\tilde{p}(l)$.
  $\mathsf{counter} \leftarrow 0$,
  repeat
    $p \leftarrow$ RANDOMNUMBER$(2\tilde{p}(l), \mathsf{counter}\|\mathsf{seed})$,
    $n \leftarrow n + 1$,
  until p prime and $\left(\frac{q}{p}\right) = 1$
  return p.

Lemma 2. *Let* $P(l, q) = \{$HASH2PRIME$(l, q, \mathsf{seed}) \mid \mathsf{seed} \in \{0,1\}^*\}$. *Then*

$$\#P(l, q) \sim l.$$

Furthermore, for fixed l *and* q, *the map* $\mathsf{seed} \mapsto$ HASH2PRIME(l, q, seed) *is a random oracle on* $P(l, q)$.

Proof. From the definition of $\tilde{p}$, there are $2l$ primes in the set p is chosen from, and half of these have q as a quadratic residue. The map is a random oracle since it is a rejection sampling based on RANDOMNUMBER, which is assumed to be a random oracle.

Lemma 3. *If we let* $C(l, \Delta) = \{$SAMPLEPRIMEFORM$(l, \Delta, \mathsf{seed}) \mid \mathsf{seed} \in \{0,1\}^*\}$, *then* $\#C(l, \Delta) \sim l$ *and* $\mathsf{seed} \mapsto$ SAMPLEPRIMEFORM$(l, \Delta, \mathsf{seed})$ *is a random oracle on* $C(l, \Delta)$ *for all* l.

Algorithm 2 SamplePrimeForm$(l, \Delta, \mathsf{seed})$

Require: $3 < \tilde{p}(l) \leq \frac{\sqrt{|\Delta|}}{2}$, $\Delta < 0, \Delta \equiv 1 \pmod 4$, $-\Delta$ is prime and $\mathsf{seed} \in \{0,1\}^*$.
Ensure: $(a, b, c) \in Cl(\Delta)$ reduced.
  $a \leftarrow$ HashPrime$(l, \Delta, \mathsf{seed})$
  $b \leftarrow$ ModSqrt(Δ, a) ▷ Efficiently computable because a is prime.
  return (a, b).

Algorithm 3 Hash2ClassGroup$(\lambda, k, \Delta, \mathsf{seed})$

Require: $k \geq 1$, $\Delta < 0, \Delta \equiv 1 \pmod 4$, $-\Delta$ is prime, $\tilde{p}(2^{2\lambda}) < \frac{\sqrt{-\Delta}}{2}$ and $x \in \{0,1\}^*$.
Ensure: $(a, b, c) \in Cl(\Delta)$ is a reduced quadratic form.
  $(a_0, b_0) \leftarrow$ SamplePrimeForm $\left(2^\lambda, \Delta, \mathsf{seed} \| 0\right)$
  $\mathsf{counter} \leftarrow 0$
  for $i = 1, \ldots, k$ **do**
    repeat
      $\mathsf{counter} \leftarrow \mathsf{counter} + 1$
      $(a_i, b_i) \leftarrow$ SamplePrimeForm $\left(2^{\lambda/k}, \Delta, \mathsf{seed} \| \mathsf{counter}\right)$ for $i = 1, \ldots, k$.
    until $a_i \neq a_j$ for all $j < i$
  end for
  $a \leftarrow \prod_{i=0}^{k} a_i$.
  Pick b such that $b \equiv b_i \bmod a_i$ for all i. ▷ Using the Chinese Remainder Theorem.
  if b even **then**
    $b \leftarrow a - b$,
  end if
  $c \leftarrow \frac{b^2 - \Delta}{4a}$, ▷ b odd $\implies b^2 \equiv 1 \bmod 4$
  return (a, b, c).

Proof. The size of $C(l, \Delta)$ follows directly from Lemma 3 since there is a 1-1 mapping between $P(l, \Delta)$ and $C(l, \Delta)$ given by the procedure SamplePrimeForm. This also proves that SamplePrimeForm is a random oracle on $C(l, \Delta)$.

Theorem 1. *The output of Algorithm 4.1 is a reduced quadratic form with discriminant Δ, and the image of the hash function has size 2^λ.*

Proof. The sampling method for a ensures that Δ is a quadratic residue modulo a and the Chinese Remainder Theorem ensures that b is a square root of Δ modulo a and that b is odd so $b^2 \equiv 1 \bmod 4$. This ensures that the division when computing c is exact and that (a, b, c) has discriminant Δ. We also see that $a < \sqrt{|\Delta|}/2$ and $-a < b < a$ so (a, b, c) is reduced by Lemma 1.

Ignoring the requirement that the prime factors must be distinct, the image size is $2^\lambda (2^{\lambda/k})^k = 2^{2\lambda}$.

Remark 1. If the output of the hash function is to be used as input to a VDF an adversary could, if the upper bounds of the prime forms used in Algorithm 3 is too small, precompute the output on some smaller forms and compute the output quickly as follows: If we define $c_i = \frac{b_i^2 - \Delta}{4a_i}$ for $i = 0, \ldots, k$, $(a, b, c) = \prod_i (a_i, b_i, c_i)$ as elements of $Cl(\Delta)$. Recall that a VDF on input $x \in Cl(\Delta)$ is x^{2^T} for some large T, and that computing this requires T operations to compute. However, if $x = x_0 \cdots x_k$,

$$x^{2^T} = x_0^{2^T} \cdots x_k^{2^T},$$

so if the adversary knows all x_i, the output can be computed using k operations.

This is handled in Algorithm 3 by ensuring that one of the factors, namely the first, is taken from a set of size $O(2^\lambda)$, so precomputing all elements from this has complexity 2^λ. If $k > 1$, the rest of the forms are smaller, but these are included to ensure that the image of the hash function is of size $2^{2\lambda}$ to ensure that the hash functions is collision resistant.

4.2 Theoretical complexity

The asymptotical complextiy of Algorithm 4.1 is dominated by the sampling of the prime factors. If we use the Miller-Rabin primality test[1], the complexity of checking an N bit number is $O(rN^3)$ where r is the number of rounds. Computing the Legendre symbol can be done in $O(N^2)$ steps using a Euclidean Algorithm-like implementation (see Algorithm 2.3.5 in [11]), and the modular square root is $O(N^4)$ but is only computed once per found factor. The expected number of candidates to consider before finding a N bit prime is $O(\log 2^N) = O(N)$ under the Cramér random model, and the probability that the Legendre symbol $= 1$, is $1/2$, so the expected complexity of the entire loop is $O(r\lambda^4)$, and does not

[1] In practice, the Miller-Rabin test should be combined with the Lucas primality tests to ensure security (see section 5.1), but for the complexity analysis here we will just consider the Miller-Rabin test.

depend on k directly, since the dominant part of the algorithm is the sampling of the first, large prime factor. We will see, however, that the parameter k makes a difference in practice in the benchmarks.

4.3 Parallelization

Algorithm 4.1 may be parallelized by running the loop to sample the a_i's in multiple threads. This requires some alterations to the algorithm:

1. We need to check that no a_i can be sampled more than once. This may be done either by checking it after the loop and resample any duplications or by making the list of a_i's synchronized such that only one thread can write to it at a time. In practice, it is very unlikely it will happen when the discriminant is large.
2. The counter variable needs to be handled such that the same value is not used twice. This may be done by allowing the i'th thread to only use use values for the counter that are of the form $qk + i$ for and then increment q instead.

5 Implementation and Benchmarks

The algorithms presented above have been implemented as part of the Rust language based high-efficiency fastcrypto library [15]. In this section we provide comments and considerations for a concrete implementation.

5.1 Primality testing

Both the theoretical complexity analysis and profiling of the actual application suggests that the main bottleneck of computing the hash function presented in this paper is primality testing, so this has to be designed carefully.

In our implementation, we use the Baillie-PSW probabilistic primality test [4]. This was chosen over using just a Miller-Rabin test because the latter is vulnerable to an attack when used on candidates that may have been chosen by an adversary [2].

5.2 Legendre symbol

In Algorithm 2, the Legendre symbol $\left(\frac{\Delta}{a}\right)$ is computed to ensure that Δ is a square modulo a. First note that this is indeed a Legendre symbol because a is a prime, so it may be computed as

$$\left(\frac{\Delta}{a}\right) = \Delta^{\frac{a-1}{2}} \bmod a. \tag{1}$$

Assuming that a multiplication of two numbers not larger than a takes $O(\log^2 a)$ operations, this formula gives an algorithm which runs in $O(\log^3 a)$ operations,

but there is a faster algorithm, similar to the Euclidean Algorithm (see e.g. [10, pp. 29–31] or [11, p. 98]), which takes $O(\log^2 a)$ operations. The latter is the approach used in our implementation.

There is an alternative way to compute the Legendre symbol which is faster but reduces the range of the hash function slightly: Recall that $\Delta < 0$ is the fixed discriminant and that $-\Delta$ is prime, so if we restrict the factors a_i of a to to $a_i \equiv 3 \bmod 4$ we get from the Law of Quadratic Reciprocity that

$$\left(\frac{\Delta}{a_i}\right) = \left(\frac{a_i}{-\Delta}\right) = a^{\frac{-\Delta-1}{2}} \mod (-\Delta).$$

Now the modulus is fixed, so we can use the Montgomery Exponentiation [14] to compute the exponent which avoids modular reduction in each step in the exponentiation loop.

5.3 Benchmarks

The implementation has been benchmarked on a MacBook Pro Laptop with an M1 Pro processor with 8 cores and 16 GB RAM, and the results are shown in Figure 1.

The target image size is 2^{256}, and the benchmarks show a 3.5 × improvement for the right choice of k, but also that increasing k beyond 2 does not give any additional benefits. The results also show that there is a small improvement in parallelisation with $k = 1$ but that it is insignificant for larger k. This is likely due to congestion and the fact that the bottleneck is the sampling of the first, large prime factor.

As discussed in section 4.1, Wesolowski's construction for a hash function samples a single prime a smaller than $\sqrt{|\Delta|}/2$ and constructs b and c from this, and this method takes, using the same program and discriminant as was used for these benchmarks, 495 ms to compute so our construction gives, at the cost of a smaller but still significantly large image for the hash function, an 200 × improvement.

6 Applications

Imaginary class groups are used in a wide array of applications, and for a lot of these, for example timed commitments [19], accumulators [6], and, perhaps most notably, verifiable delay functions (VDFs) [20,16], a secure and efficient hash function to the class group is an important primitive.

Recall that a VDF is a function $F : G \to G$ defined by $g \mapsto g^{2^T}$ for some large T. Computing this function takes T group operations if the order of the group is unknown, but it is also possible to derive a proof that the computation was done correctly, which is fast to verify.

As noted in Remark 3 in [20], using a hash function to pick the input of the VDF is important, because knowing the result of $F(x) = x^{2^T}$ makes it easy to

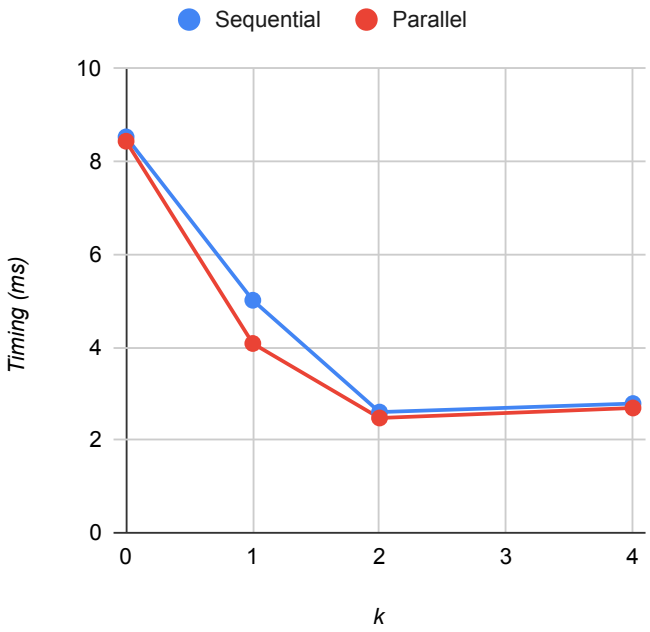

Fig. 1. Plots of performance for different choices of the parameter k. The class group used for the benchmarks has a 3072 bit discriminant, but this has no influence on the runtime.

compute, for example,

$$F(x^a) = (x^a)^{2^T} = (x^{2^T})^a = F(x)^a.$$

At the time of writing, the only VDF in production is run by the Chia Network [9], where VDFs are used in a proof-of-time consensus protocol. However, Chia Networks' deployment does not use a hash function to generate a random input to a VDF. Instead it uses a fixed input to the VDF and samples a new random discriminant for each VDF instance.

The Sui blockchain also has a VDF implementation, and while implementing this we have found that sampling sufficiently large (according to [12]) discriminants at random may take several seconds, which is too slow for on-chain usage. Due to this limitation, a fixed discriminant is used instead so the input to the VDF must be sampled for each instance instead based on user-provided randomness. This requires a hash function to which maps to a class group element, which is fast enough for it to be computed on-chain, but Wesolowski's hash function construction takes up to a second to compute, which is too slow for on-chain

usage, so we use the construction presented in this paper which computes a hash in as little as 2 ms (see Figure 1) for a large 3072 bit discriminant.

7 Future work

In certain applications, it is imperative to ensure that hash functions operate in constant time to prevent any leakage of information about the input. The challenge with the algorithm described in this paper lies in its dependency on probabilistically sampling random primes from a range, which inherently varies in time. Interesting lines of research could include:

(a) Uniform sampling techniques for developing an efficient, constant time algorithm,
(b) Pre-computations by preparing a pool of random primes in advance,
(c) Time-padding methods where the execution time is artificially extended to be fixed.

References

1. Adleman, L.M., Rivest, R.L., Shamir, A.: Cryptographic communications system and method. US Patent No. 4,405,829. (Sep 1983), https://www.google.com/patents/US4405829, patent filed 14 September 1977.
2. Albrecht, M.R., Massimo, J., Paterson, K.G., Somorovsky, J.: Prime and prejudice: Primality testing under adversarial conditions. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 281–298. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243787, https://doi.org/10.1145/3243734.3243787
3. Bach, E.: How to generate factored random numbers. SIAM Journal on Computing **17**(2), 179–193 (1988). https://doi.org/10.1137/0217012, https://doi.org/10.1137/0217012
4. Baillie, R., Wagstaff, S.S.: Lucas pseudoprimes. Mathematics of Computation **35**(152), 1391–1417 (1980), http://www.jstor.org/stable/2006406
5. Beaver, D., Chalkias, K., Kelkar, M., Kokoris-Kogias, L., Lewi, K., de Naurois, L., Nikolaenko, V., Roy, A., Sonnino, A.: Strobe: Streaming threshold random beacons. In: 5th Conference on Advances in Financial Technologies (AFT 2023). Schloss-Dagstuhl-Leibniz Zentrum für Informatik (2023)
6. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 561–586. Springer International Publishing, Cham (2019)
7. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 677–706. Springer (2020)
8. Chen, M., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., Shelat, A., Cohen, R.: Multiparty generation of an rsa modulus. Journal of Cryptology **35**(2), 12 (2022). https://doi.org/10.1007/s00145-021-09395-y, https://doi.org/10.1007/s00145-021-09395-y

9. Cohen, B., Pietrzak, K.: The chia network blockchain (2019), https://api.semanticscholar.org/CorpusID:209373416
10. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer Publishing Company, Incorporated (2010)
11. Crandall, R., Pomerance, C.: Prime numbers. A computational perspective. Springer-Verlag, New York (2001)
12. Dobson, S., Galbraith, S., Smith, B.: Trustless unknown-order groups. Mathematical Cryptology **1**(2), 25–39 (Mar 2022), https://inria.hal.science/hal-02882161, https://eprint.iacr.org/2020/196.pdf
13. Gauss, C., Waterhouse, W.: Disquisitiones Arithmeticae. Springer-Verlag (1986), https://books.google.dk/books?id=Y-49PgAACAAJ
14. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation **44**, 519–521 (1985), https://api.semanticscholar.org/CorpusID:119574413
15. Mysten Labs: fastcrypto, https://github.com/MystenLabs/fastcrypto
16. Pietrzak, K.: Simple verifiable delay functions. In: 10th innovations in theoretical computer science conference (itcs 2019). Schloss-Dagstuhl-Leibniz Zentrum für Informatik (2019)
17. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
18. Seres, I.A., Burcsi, P., Kutas, P.: How (not) to hash into class groups of imaginary quadratic fields? Cryptology ePrint Archive, Paper 2024/034 (2024), https://eprint.iacr.org/2024/034, https://eprint.iacr.org/2024/034
19. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient cca timed commitments in class groups. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 2663–2684. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484773, https://doi.org/10.1145/3460120.3484773
20. Wesolowski, B.: Efficient verifiable delay functions. J. Cryptol. **33**(4), 2113–2147 (oct 2020). https://doi.org/10.1007/s00145-020-09364-x, https://doi.org/10.1007/s00145-020-09364-x