Tiresias: Large Scale, Maliciously Secure Threshold Paillier

Zvika Brakerski[1,2], Offir Friedman[1], Avichai Marmor[1], Dolev Mutzari[1], Yehonatan C. Scaly[1], Yuval Spiizer[1], and Avishay Yanai

[1] dWallet Labs, research@dwalletlabs.com
[2] Weizmann Institute of Science

Abstract. In the threshold version of Paillier's encryption scheme, a set of parties collectively holds the secret decryption key through a secret sharing scheme. Whenever a ciphertext is to be decrypted, the parties send their decryption shares, which are then verified for correctness and combined into the plaintext. The scheme has been widely adopted in various applications, from secure voting to general purpose MPC protocols. However, among the handful existing proposals for a maliciously secure scheme, one must choose between an efficient implementation that relies on non-standard assumptions or a computationally expensive implementation that relies on widely acceptable assumptions.

In this work, we show that one can enjoy the benefits of both worlds. Specifically, we adjust a scheme by Damgård et al. (Int. J. Inf. Secur. 2010) to get a practical distributed key generation (DKG). While the original scheme was only known to be secure under ad-hoc non-standard assumptions, we prove that the adjusted scheme is in fact secure under the decisional composite residuosity (DCR) assumption alone, required for the semantic security of the Pallier encryption scheme itself. This is possible thanks to a *novel reduction technique*, from the soundness of a zero-knowledge proof of equality of discrete logs, to the factoring problem. Furthermore, we use similar ideas to prove that batching techniques by Aditya et al. (ACNS 2004), which allows a prover to *batch* several statements into a single proof, can be applied to our adjusted scheme. This enables a batched threshold Paillier decryption in the fully distributed setting for the first time.

Until now, verifying that a decryption share is correct was the bottleneck of threshold Paillier schemes and hindered real world deployments (unless one is willing to rely on a trusted dealer). Our work accumulates to shifting the bottleneck back to the plaintext reconstruction, just like in the semi-honest setting, and render threshold Paillier practical for the first time, supporting large scale deployments.

We exemplify this shift by implementing the scheme and report our evaluation with up to 1000 parties, in the dishonest majority setting. For instance, over an EC2 c6i machine, we get a throughput of about 50 and 3.6 decryptions per second, when run over a network of 100 and 1000 parties, respectively.

Keywords: Additive Homomorphic Encryption · Paillier Encryption · Threshold Encryption · Batched ZK Arguments

1 Introduction

The Paillier encryption scheme from 1999 [Pai99] has gained significant popularity due to its advantageous properties. It is a public-key encryption scheme renowned for its additive homomorphic property, enabling linear operations on encrypted data without requiring decryption. Additionally, the Paillier scheme supports a large message space, enabling useful operations on secrets.

Motivated by applications in voting systems, several authors have proposed *threshold* variants of the Paillier encryption scheme [FPS01, DJN10]. These variants are based on similar constructions for RSA signatures [Sho00].

A threshold encryption scheme facilitates a set of parties utilizing a public encryption key pk to encrypt messages while *collectively maintaining* the corresponding secret key sk for decrypting ciphertexts. In this scheme, each party P_j possesses a secret decryption key share sk_j. When the parties collectively decide to decrypt a ciphertext ct, they participate in a cryptographic protocol that ultimately reveals the message while ensuring the confidentiality of the secret decryption key. Typically in such protocols, each party P_j broadcasts a "decryption share" $\mathsf{ct}_j = Dec_{sk_j}(\mathsf{ct})$. If a sufficient number of parties, passing a certain pre-defined threshold, broadcast their decryption shares, those can then be locally combined by anyone to recover the plaintext $\mathsf{pt} = Dec_{sk}(\mathsf{ct})$.

The combination of homomorphic properties and threshold decryption capabilities has rendered the Paillier encryption scheme highly appealing in systems focused on privacy-preserving voting [KLM+20, DJN10] and data aggregation in general [MT21, BS21]. Moreover, threshold decryption of the Paillier scheme (and additively homomorphic encryption in general) serves as a foundational building block in other cryptographic protocols like threshold signatures [GGN16] and secure multiparty computation (MPC) in general [DN03, DPSZ12]. In many prior works, such as [FPS01, GGN16], the focus was primarily on the threshold Paillier decryption feature. However, these approaches often relied on a trusted dealer for key generation and secret key share distribution. This reliance reintroduced a security risk that originally prompted the use of threshold encryption in the first place. To this end, fully-fledged threshold schemes have been proposed, (e.g., [DK01]). In these schemes, the involvement of a trusted dealer is entirely eliminated, and the generation and distribution of keys are carried out by the participating parties themselves.

Similar to other RSA-based primitives, like signature [RSA78] and verifiable delay function (VDF) [BBBF18] schemes, the public key of the Paillier encryption scheme consists of a modulus N that is the product of two large prime numbers P and Q. Therefore, many works (see [BDF+23] and references within) deal with distributed RSA modulus as a stand alone and independent building block, leaving additional necessary cryptographic material to be generated by the specific application, be it the RSA signature, RSA encryption, VDF, or Paillier encryption schemes. The additional cryptographic material may be different from scheme to scheme. For example, in the threshold RSA signature scheme, P_j's signature share on message m is computed by $\sigma_j = H(m)^{d_j} \mod N$ where $d = \sum d_j$ is the secret signing key shared among the parties, and the final sig-

nature is $\sigma = \prod \sigma_j$. If one of the parties computes an incorrect partial signature $\sigma'_j \neq \sigma_j$ then it is evident to everyone, since the final signature σ' will not verify. In contrast, in the context of threshold encryption, without employing some verifiability mechanism it might not be possible to tell whether a decryption share was computed correctly or not. To this end, proposals for threshold Paillier devise such a verifiability mechanism, in the form of a zero knowledge (zk) proof. That is, in addition to the aforementioned decryption share, each party provides a zero knowledge proof for the claim that the decryption share is computed correctly using its secret key share. In that sense, the proposed threshold Paillier protocols differ mostly in the way that the zero knowledge proof is implemented, offering a trade off between efficiency and security. Specifically, these protocols offer a trade off in the three metrics below:

- **Key generation efficiency** refers to whether distributed key generation is practical.
- **Proof efficiency** refers to the size and the time it takes to generate/verify the zero knowledge proof of the correctness of the decryption share.
- **Strength of assumptions** refers to the cryptographic assumptions underlying the soundness of the proof.

Considering only protocols with a feasible key generation phase, one has to choose between a protocol with an efficient proof system that relies on non-standard assumptions (such as [DJN10] using the assumptions in [DK01]), and a protocol whose proof system is inefficient but relies on standard widely accepted assumptions (e.g., [FS01, HMR+19]). This raises the following question:

Is it possible for a threshold Paillier encryption scheme to incorporate an efficient key generation and proof while relying solely on standard assumptions?

In this work we answer this question in the affirmative. We depart from a protocol that has an efficient key generation and proof, but relies on non-standard assumptions, and present a novel reduction technique that is applied to the proof of soundness of the zero knowledge proof of correct decryption share. This, for the first time, allows using an efficient version of threshold Paillier without compromising on security.

1.1 Previous Work: Efficiency vs. Security

In the following, we provide a more detailed overview of the trade-off discussed above, which involves a tension between efficiency and the level of leniency associated with relying on non-standard assumptions. On one extreme, the protocol by Algesheimer et al. [ACS02] allows distributed generation of a *bi-prime* public key $N = PQ$ consisting of *safe primes* (i.e., with $(P-1)/2$ and $(Q-1)/2$ primes). The fact that N is a product of safe primes enables an efficient zero knowledge protocol for the correctness of the threshold decryption, while also achieving security under standard assumptions. Generating N as a product of safe primes is commonly adopted by works that assume a trusted dealer [FPS01], since a dealer can easily generate such a key. However, *distributed* generation of safe primes remains an infeasible task, which is evident by the fact that [ACS02] has never been implemented.

On the other extreme, Damgård et al. [DK01] proposed a protocol for threshold RSA signatures that lowers the bar by generating a public key N that is a product of 'general' primes[3] P, Q (i.e., they are not necessarily safe), and using the same efficient zk-proof as [FPS01]. This result is transferable to threshold Paillier as mentioned in [DJN10]. However, since the generated primes may lack some important properties that exist in safe primes, the soundness of that zero knowledge proof has to rely on *non-standard assumptions* (which are discussed later in Section 1.3).

In order to bridge between the above mentioned extremes, Fouque and Stern [FS01] proposed a new protocol in which the key generation produces primes P, Q that are only *almost safe primes*, meaning that $\frac{P-1}{2}$ and $\frac{Q-1}{2}$ are *B-rough numbers* and *co-prime*. Unlike general primes, the B-roughness property facilitates a proof of soundness for the zk-protocol without relying on new assumptions. Nevertheless, while distributed generation of almost safe primes can be done, it is impractical for most use-cases (we estimate that generation of almost safe primes would be 1000x slower than general ones). Moreover, the technique incurs a degradation of the efficiency of the zero knowledge proof. As reported by the authors, the proof efficiency is about $30\times$ worse compared to [DK01].

Another bridging attempt is due to Hazay et al. [HMR+19]. Their protocol, similar to [DK01], builds on a public key N that is a product of general primes, but uses a different zero knowledge protocol (using the cut-and-choose technique) than the one used in [DK01]. Their proof, while relying on standard assumptions, suffers from poor soundness $(1/2)$, which means that it has to be repeated κ times to meet real security requirements.

Since safe prime generation as in [ACS02] is infeasible, we are left with the choice between accepting the extra assumptions in [DK01] and getting an efficient proof (and therefore threshold decryption), or sticking to the widely accepted assumptions but suffering an inefficient proof, as in [FS01] and [HMR+19].

It worth mentioning a different approach, proposed by Baum et al. [BDTZ16], which removes the zero knowledge proof from threshold Paillier decryption altogether. That is, in order to decrypt the ciphertext $c = \mathsf{Enc}_{pk}(m; r)$ the parties first reconstruct the randomness r, which in turn enables learning the plaintext m. This approach, however, apart from revealing the randomness r to the adversary, enables an attacker to anonymously cheat in the reconstruction of r and thus to deny decryption. Both of the above issues are problematic in most cases, yet tolerable in some scenarios[4].

Lastly we would like to mention previous work by Seres and Burcsi [SB21] which upgrades the security assumptions of Pietrzak's proof of exponentiation.

[3] We remark that distributed generation of a product of general primes is due to Boneh and Franklin [BF97] and its improvements [DdSGMRT21, CHI+21, BDF+23].

[4] Specifically, in [BDTZ16] such decryption is happening only in the preprocessing phase of a generic MPC protocol, in which case, neither an abort nor leakage of r gives the adversary any advantage. Alternatively, the same work proposes a method to avoid denial of decryption at the cost of two additional rounds and assuming the primes are safe, a property we wish to avoid in the first place.

While the reductions share some ideas there are meaningful differences. Most importantly they assume that $\phi(N)$ does not have factors in a certain range, which is okay in their context since a single party holding the factorization of N can provide a proof of this fact. In our context it is much more involved to produce such a proof since the factorization of N is shared among the parties.

1.2 Our Contribution

- We present the first practical, efficient, large-scale threshold Paillier encryption protocol, supporting efficient distributed key generation and threshold decryption, built upon the same decisional composite residuosity assumption as the standard Paillier encryption. The protocol is secure in the presence of a malicious adversary who statically corrupts $t < n$ parties.
- At the heart of our contribution lies a novel proof for the soundness of the standard proof system of equality of discrete logs over the group QR_{N^2} of quadratic residues modulo N^2, *even when N is not a product of safe primes*. Such proof may be of independent interest: First, the same proof can be used in threshold protocols over RSA groups, such as threshold RSA signature. Second, the requirement of safe primes in various cryptographic primitives can be re-assessed, which is left to future work.[5] The ramification of that proof is that distributed Paillier key generation can be implemented using *any* distributed bi-prime modulus generation (for 'general' primes), and in particular, we can leverage recent advances, like Diogenes [CHI+21], for key generation by thousands of parties.
- In a real-world system required to continuously process many ciphertexts, the parties need to verify decryption shares received from other parties for each of these ciphertexts. Verification of decryption shares from many parties across multiple ciphertexts can easily dominate the overall cost of decryption. To address this issue, we use a batching technique due to [APB+04], which allows a prover to *batch* several decryption shares into a single proof. Specifically, proving and verifying B statements using the batched proof system requires a single 'large' exponentiation and $\mathcal{O}(B)$ 'small' exponentiations, rather than $\mathcal{O}(B)$ large exponentiations (where 'large' refers to the size of the shares, e.g., 4096 bits, and 'small' refers to the computational security parameter, e.g., 128 bits). The original proof of this batching technique has also assumed safe primes. We provide a new proof which avoids this issue. This results in the first batched proof system for threshold Paillier decryption when N is not a product of safe primes. Overall it introduces a significantly more efficient proof system for the equality of discrete logs (as demonstrated in Section 5). It reduces the total time complexity of threshold decryption to roughly that of a semi-honest model with no proof system. E.g., for 1000 parties proof overhead is about 5%.

[5] That being said, while similar techniques may be applied to remove the requirement of safe primes in other cases as well, in some protocols the requirement that the primes are safe might be crucial, so every protocol must be analyzed on its own.

Although our contributions are concentrated around threshold decryption, we briefly discuss the protocol for distributed key generation as well (in Section 3 and in more detail in F), for completeness of the exposition.

Performance comparison. In Table 1 we compare the performance between this work and previous works that rely on standard assumptions only, namely [FS01] and [HMR+19]. As the performance is dominated by exponentiations by a large exponent (thousands bit-length) the table focuses on that metric. Note that the large exponents in all the protocols are of the same size. Importantly, our protocol is the only one that supports batch decryption (which essentially requires of aggregation of proofs). Thus, in [FS01] and [HMR+19] the overhead of decrypting a batch of ciphertexts grows linearly with the batch size, whereas in our protocol the overhead remains constant.

	Number of Exponentiations		
Work	In general	When $\kappa = 128, \sigma = 40, B = 2^{16}$	Supports batching
[FS01]	$2 \cdot \frac{\kappa}{\log B} \cdot (\lceil \frac{\sigma}{\log B} \rceil + 1)$	64	No
[HMR+19]	κ	128	No
This work	**2**	**2**	Yes

Table 1: The number of exponentiations required for proof generation and verification in each protocol, where κ represents computational security (i.e., 2^{κ} is infeasible), and σ represents statistical security (i.e., $2^{-\sigma}$ is negligible). For [FS01], we utilize the guaranteed B-roughness of $\frac{P-1}{2} \cdot \frac{Q-1}{2}$.

1.3 Technical Overview

Let us begin with the high-level overview of the threshold Paillier decryption. As mentioned above, Paillier's public key is a modulus N that is a product of two large primes P and Q. The secret key d is derived from these primes, and it is assumed to be computationally infeasible to obtain it from N. In threshold Paillier key generation protocols, the parties first obtain a sharing of P and Q, and then derive a sharing of d as well as the product $N = PQ$ in plain. To be more specific, the generated primes P and Q are required to satisfy $\gcd(\phi(N), N) = 1$, where $\phi(N) = (P-1)(Q-1)$. Using this fact, the secret key is computed[6] by $d = \phi(N) \cdot [\phi(N)^{-1} \mod N] \in \mathbb{Z}$ and shared among the parties using a secret sharing scheme over the integers[7], such that party P_j obtains a share d_j. In addition to the public modulus N, the parties generate a public verification key v_j for every P_j, such that $v_j = g^{d_j}$ for some basis element g from the group of quadratic residues modulo N^2 (denoted QR_{N^2}). Then, when the parties agree to decrypt a ciphertext ct, party P_j sends the decryption share $\mathsf{ct}_j = \mathsf{ct}^{d_j}$ along with a zero-knowledge proof that ct_j is computed correctly using the public ct and the secret d_j.[8] This proof is a proof of *equality of discrete logs* of the values

[6] There are several choices for the exact form of the secret key, which are all variants of the one described above.

[7] Some works assume secret sharing over the ring $\mathbb{Z}_{N\phi(N)}$ but this is harder to achieve without a trusted dealer.

[8] When the threshold is smaller than the number of parties each exponent is multiplied by the appropriate Lagrange coefficient.

ct_j and v_j over QR_{N^2}, with respect to the bases ct and g. That is, if P_j computes its decryption share correctly then $\log_{\mathsf{ct}}(\mathsf{ct}_j) = \log_g(v_j) = d_j$.

In the following we present in more detail why safe primes are powerful for efficient proofs, and later we explain how we achieve the same efficiency without using safe primes and without resorting to additional assumptions.

Suppose that the generated modulus N is a product of two safe primes P, Q, meaning that $P' = \frac{P-1}{2}$ and $Q' = \frac{Q-1}{2}$ are primes as well. There are three main benefits from the assumption that P' and Q' are primes:

1. In the context of proofs for equality of discrete logs, it is commonly assumed that the group is cyclic. For any pair P, Q of distinct safe primes, the group QR_{N^2} is guaranteed to be cyclic, since $\mathsf{QR}_{N^2} \cong \mathsf{QR}_{P^2} \times \mathsf{QR}_{Q^2}$ and the orders of QR_{P^2} and QR_{Q^2} are co-prime. However, in the general case these orders may share a common factor and the group QR_{N^2} may not be cyclic.
2. When the group QR_{N^2} happens to be cyclic, meaning that there exist an element g, called a generator, such that every element in QR_{N^2} equals g^i for some i. In fact, the probability of a random element $g \leftarrow \mathsf{QR}_{N^2}$ to not be a generator, is close to the probability of guessing one of the factors of N. Having a generator helps when arguing security for zero knowledge protocols. However, in the general case (where the primes are not safe), even when QR_{N^2} is cyclic, the probability of a random element to be a generator is not negligible. This problem is exacerbated by the fact that no known algorithm exist for finding a generator or determine if an element is one.
3. The standard proof of soundness (see [FPS01]) obtains an equation of the form $x^e = 1 \mod N^2$ for a small e ($\ll \min\{P', Q'\}$), and concludes that $x = 1$ since e is necessarily co-prime with $\phi(N^2)/4 = NP'Q'$. This conclusion cannot be made when P' and Q' are allowed to be composite numbers.

These three benefits of safe primes are also drawbacks of general primes. The work [DK01] overcomes these drawbacks and applies the same zero knowledge proof as in [FPS01], but does not assume that the primes are safe and therefore relies on the following non-standard assumptions:

1. It is computationally hard to compute an element $a \in \mathbb{Z}_N^*$ such that $a \neq \pm 1 \mod N$ and the order of a is not divisible by the largest factor of $\phi(N)$.
2. A random element in QR_N is indistinguishable from those elements in QR_N with maximal order.

While these assumptions have not been proven insecure to date, relying on non-standard cryptographic assumptions that have received very little attention can potentially pose a risk in practice.

In [FS01] mentioned earlier, which aims at avoiding extra assumptions without requiring the primes to be safe, the efficiency is degraded for two reasons. First, the protocol has to be repeated with many bases $g_1, g_2, \ldots$, which *together* generate QR_{N^2} with overwhelming probability. Second, the protocol requires $\frac{P-1}{2}$ and $\frac{Q-1}{2}$ to be B-rough (i.e., to have all prime factors larger than B), and the soundness depends on B. Since all known practical key generation protocols result in quite a small B, soundness must be amplified via parallel repetitions.

Our approach. In this work we take the same approach as in [DK01] (as it applies to the threshold Paillier cryptosystem presented in [DJN10] with $s = 1$), but *remove their extra assumptions*. Specifically, we manage to obtain an efficient zero knowledge proof of equality of discrete logs over QR_{N^2}, even when N is a product of general primes.[9] By setting minimal and practically achievable properties to the primes, we overcome the above three drawbacks by using the following observations.

1. Even though P, Q are not safe-primes, with high (but not overwhelming) probability $\frac{P-1}{2}$ and $\frac{Q-1}{2}$ are co-prime, and therefore QR_{N^2} is a cyclic group of order $\frac{N\phi(N)}{4}$. During the key generation protocol the parties will reject prime candidates that do not meet the requirement that $\frac{P-1}{2}$ and $\frac{Q-1}{2}$ are co-prime. The rejection of prime candidates that do not satisfy that condition incurs only a small constant factor overhead (about $1.2\times$ as calculated in Appendix F) to the key generation protocol.
2. Even when P, Q are not safe, the order of a random element $g \in \mathsf{QR}_{N^2}$ is 'close enough' to the order of QR_{N^2}, and so for our purpose it can be used as if it was a generator. Specifically, we prove that with overwhelming probability, the order of a randomly sampled $g \in \mathsf{QR}_{N^2}$ is at least $|\mathsf{QR}_{N^2}|$ divided by a sufficiently smooth number (whose prime factors are all small).
3. For similar reasons, in the proof of soundness, instead of obtaining the equation $x^e = 1 \mod N^2$ we obtain $x^e \cdot \eta = 1 \mod N^2$, where $\eta \in \mathsf{QR}_{N^2}$ has a smooth order δ. This means that $x^{e \cdot \delta} = 1 \mod N^2$. Then, we divide the proof into two cases: If $x^e = 1$ (as in the case where g is a generator), then we can find the factorization of e since e is small, from which we can find the factorization of N using classical techniques, as long as $x \neq 1$. Otherwise, $x^e \neq 1 \mod N^2$ is an element of small smooth order and so we can employ Pollard's $p - 1$ method in order to factor N.

Notice that the above ideas would result in an efficient reduction, albeit non-polynomial, and thus would only break the sub-exponential factoring problem. Our simulation carefully chooses challenges such that the size of e has a bound dependent on the chance of success of the PPT adversary rather directly on the security parameter, resolving this issue. There is also a dependence on the smoothness of the order of x^e in the statistical security parameters which may be reduced to a constant by using multiple bases.

As we are concerned with concrete security there are further issues we consider throughout the paper. For example we provide two reductions one to the uniform factoring problem and another for the non-uniform factoring problem. The reason is that although we do not require non-uniformity to prove that our scheme is secure asymptotically, when fixing concrete parameters, the reduction

[9] We do require $N = PQ$ to satisfy $\gcd(P - 1, Q - 1) = 2$, which is a much weaker condition than the common requirement that $(P - 1)/2$ and $(Q - 1)/2$ are prime. This property has a small impact on the efficiency of the key generation protocol, does not affect the efficiency of the threshold decryption, and does not introduce additional cryptographic assumptions.

against a non-uniform PPT adversary is more tight. Specifically, an adversary $\mathcal{P}^*$ that breaks our scheme in time $T(\kappa)$ with probability $\varepsilon(\kappa)$ is reduced to a non-uniform adversary $\mathcal{A}$ that breaks the factoring game above in time $T'(\kappa)$ with probability $\varepsilon'(\kappa)$ such that $\frac{T'(\kappa)}{\varepsilon'(\kappa)} = \frac{T(\kappa)}{\varepsilon(\kappa)} \cdot \mathsf{poly}(\kappa)$. That is, the reduction is linear in T/ε. On the other hand, the reduction against uniform PPTs has $\frac{T'(\kappa)}{\varepsilon'(\kappa)} = \frac{T(\kappa)}{\varepsilon^2(\kappa)} \cdot \mathsf{poly}(\kappa)$. While this reduction is still polynomial, when fixing concrete parameters, the computational security parameter has to be doubled. Due to the sub-exponential hardness of factoring, this in turn requires increasing the modulus size $\log_2(N)$ by $8\times$, and so exponentiation will cost at-least $64\times$ more. Another subtle issue arises when applying the Fiat-Shamir transform to the batched proof, which is a 5 rounds protocol. Generically this may cause significant loss of soundness which would degrade the parameters of the scheme. We show this is not the case for the batched protocol by applying concepts from [AFK22]. The reduction is analyzed such that one may estimate concrete statistical and computational security based on different parameters of the scheme (see table 2).

1.4 Organization

In Section 2 we present our notation and the mathematical background that is necessary in order to understand the rest of the paper. A brief reminder of basic number theoretic facts can be found in Appendix A. In Appendix B we recall Shamir Secret Sharing (SSS) over a field along with proving improved bounds on the size of SSS shares over the integers. In Section 3 we present the adjusted distributed key generation and threshold decryption with a trustless setup. Background on the Paillier encryption scheme in can be found in Appendix C, and a simplified version of threshold scheme using a trusted setup can be found in Appendix D. For the ideal functionality of threshold decryption and further details on the distributed key generation protocol we refer to Appendices E and F respectively. In Section 4 we present the zero knowledge protocol for equality of discrete logs over QR_{N^2}, and provide our new proof of security. Omitted proofs are in Appendix G. Subsections 4.4 and 4.6 present batch and optimization techniques, and in Appendix H we further discuss batch verification optimization and multi-exponentiation. Finally, we report our experiments in Section 5.

2 Preliminaries

General notation. We let $\mathbb{N}, \mathbb{Z}, \mathbb{Z}_m$ denote the set of natural numbers excluding 0, integers, and integers modulo m, respectively. In addition, we denote by $\mathbb{Z}_m^*$ the multiplicative group modulo m. We denote by primes and primes_m the set of all prime numbers and the set of prime numbers smaller than m, respectively. For $a, b \in \mathbb{Z}$ we denote by $[a], [a, b], [a, b)$ and (a, b) the sets $\{1, \ldots, a\}$, $\{a, \ldots, b\}$, $\{a, \ldots, b-1\}$ and $\{a+1, \ldots, b-1\}$, respectively. The bit representation of an integer x is denoted $(x_{\ell-1}, \ldots, x_0)$ with $x_{\ell-1}$ being the most significant bit. We denote by $X \leftarrow \Omega$ a uniform sampling from a set Ω. We use κ and σ to denote computational and statistical security parameters, respectively. We denote by σ_0 a preliminary statistical security parameter which is upgraded to σ. We denote

by $\mathsf{time}(A(x_1, x_2, \ldots))$ the run time of an algorithm A on inputs $(x_1, x_2, \ldots)$. We may denote a vector of group elements by $(g_1, \ldots, g_c) := \mathbf{g} \in \mathcal{G}^c$ where $c := \frac{\sigma}{\sigma_0}$ and $\mathbf{g}^r := (g_1^r, \ldots, g_c^r)$.

Mathematical Background. We refer Appendix A for preliminary mathematical background. Below we list 3 useful definitions.

Definition 2.1 (β-Smooth Number) *For $\beta \in \mathbb{N}$, an integer k is called* β-smooth *if all the prime factors of k are smaller than β.*

Definition 2.2 (Safe Prime) *A prime number p is called* safe *if $\frac{p-1}{2}$ is prime.*

Definition 2.3 (Conforming Bi-Prime) *An integer N is a* conforming bi-prime *if there exist two primes P, Q such that $N = PQ$, $P = Q = 3 \mod 4$, $\gcd(N, \phi(N)) = 1$, and $\gcd(P-1, Q-1) = 2$.*

Groups. We use multiplicative notation for groups. Let $\mathcal{G}$ be a finite abelian group. For $g_1, \ldots, g_k \in \mathcal{G}$ we denote by $\langle g_1, \ldots, g_k \rangle$ the subgroup $\mathcal{H} \subseteq \mathcal{G}$ generated by $g_1, \ldots, g_k$. That is, $\mathcal{H} = \left\{ \prod_{i=1}^{k} g_i^{\alpha_i} \right\}_{\alpha_i \in \mathbb{Z}, i \in [k]}$. We denote by $|\mathcal{G}|$ and $\mathsf{ord}(g)$ (or $|\langle g \rangle|$) the order of (number of elements in) $\mathcal{G}$ and $\langle g \rangle$, respectively. An element $y \in \mathcal{G}$ is a *quadratic residue* if there exists an $x \in \mathcal{G}$ with $x^2 = y$. For abelian groups, the set of quadratic residues forms a subgroup. For an integer $m > 2$, we denote by QR_m the subgroup of quadratic residues in $\mathbb{Z}_m^*$. By Lagrange's theorem, if $\mathcal{G}$ is a group and $\mathcal{H} \subseteq \mathcal{G}$ then $|\mathcal{H}|$ divides $|\mathcal{G}|$.

Definition 2.4 (Statistical Distance) *Let $X, Y : \Omega \to [M]$ be two random variables. The* statistical distance *between X, Y, denoted $\mathsf{SD}(X, Y)$, is defined as $\mathsf{SD}(X, Y) := \sum_{w \in \Omega} |\Pr[X = w] - \Pr[Y = w]|$.*

Hardness assumptions. Let $\mathsf{GenModulus}$ be a polynomial time algorithm that, on input 1^κ, outputs (N, P, Q) where $N = PQ$, and N is a conforming bi-prime except with probability negligible in κ.

Definition 2.5 (DCR) *We say that the decisional composite residuosity (DCR) problem is hard relative to* $\mathsf{GenModulus}$ *if for all probabilistic polynomial time algorithms $\mathcal{A}$ there exists a negligible function neg such that*

$$\left| \Pr\left[\mathcal{A}\left(N, [r^N \mod N^2]\right) = 1 \right] - \Pr\left[\mathcal{A}(N, r) = 1 \right] \right| \leq \mathsf{neg}(\kappa)$$

where the probabilities are taken over $N \leftarrow \mathsf{GenModulus}(1^\kappa)$ and $r \leftarrow \mathbb{Z}_{N^2}^*$.

Definition 2.6 (Factoring) *We say that the factoring problem is hard relative to* $\mathsf{GenModulus}$ *if for all (uniform / non-uniform) probabilistic polynomial time algorithms $\mathcal{A}$ there exists a negligible function neg such that*

$$\Pr_{N \leftarrow \mathsf{GenModulus}(1^\kappa), (P', Q') \leftarrow \mathcal{A}(N)} [P' \cdot Q' = N \wedge P', Q' \notin \{1, N\}] \leq \mathsf{neg}(\kappa).$$

Remark 2.1. In the classic factoring problem $\mathsf{GenModulus}$ is defined by choosing P, Q as independently, uniformly random ℓ-bits random primes. In the case of distributed key generation we get a different $\mathsf{GenModulus}$. In a classical paper [BF97] provide a reduction between the two cases.

2.1 Shamir Secret Sharing over the Integers

We present Shamir's threshold secret sharing over a field [Sha79] in Appendix B.1 and present its extension to over the integers [NS10, Rab98, VAS19] below.

Let $s \in \mathbb{Z} \cap [-b, +b]$ be a secret. Define $\Delta_n = n!$ and define some bound $I(\sigma, n, b)$ on the (absolute value of the) coefficients of the polynomial. Until recently (see, e.g., [VAS19]), the bound $I(\sigma, n, b) = 2^\sigma \cdot \Delta_n^2 \cdot b$ was used. However, following [BDO22], we provide a tighter bound for $I(\sigma, n, b)$ in Appendix B.2.

The algorithm $\mathsf{Share}_{t,n}(s)$ picks $a_1, \ldots, a_t \leftarrow [-I(\sigma, n, b), +I(\sigma, n, b)]$ and outputs $([s]_1, \ldots, [s]_n)$ where $[s]_j = p(j)$ and $p(x) = \Delta_n \cdot s + \sum_{i=1}^{t} a_i \cdot x^i$.

Reconstruction works as follows. Given a set $T \subset [n]$ of $t+1$ distinct elements and given points $\{(j, [s]_j)\}_{j \in T}$, we have $p(0) = \sum_{j \in T} \lambda_{T,j}^0 \cdot [s]_j = \Delta_n s$, where $\lambda_{T,j}^v$ is the Lagrange coefficient corresponding to point $j \in T$, to restore the polynomial p evaluation at v (see Appendix B). However, since the Lagrange coefficients might not be in $\mathbb{Z}$, we multiply them first by Δ_n, and get

$$\left(\sum_{j \in T} \Delta_n \lambda_{T,j}^0 \cdot [s]_j \right) / \Delta_n^2 = s. \tag{1}$$

For an (n, t)-threshold Shamir sharing over the integers of secret $s \in [-b, +b]$, we denote the upper bound on the absolute value of the shares on s by $D(\sigma, n, t, b)$, which is: $D(\sigma, n, t, b) = \Delta_n \cdot b + \sum_{i=1}^{t} I(\sigma, n, b) \cdot n^i \leq \Delta_n \cdot b + 2n^t I(\sigma, n, b)$

2.2 Zero Knowledge

A *zero knowledge protocol* between a prover $\mathcal{P}$ and a Verifier $\mathcal{V}$ is a protocol in which the prover convinces the verifier of some fact regarding a public instance without revealing any information to the verifier beyond the correctness of this fact. A formulation of such property is typically done via three definitions, namely *completeness*, *soundness* and *zero-knowledge*. Completeness means that if both the prover and the verifier acts according to the protocol then the verifier will accept. Soundness means that it's computationally infeasible for a cheating prover to convince an honest verifier of a false fact. Lastly zero-knowledge means that the verifier does not learn any new information about the instance except the fact. In our context, we are specifically interested in *Special honest verifier zero-knowledge (SHVZK)* defined below.

Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a binary relation, where $(x, w) \in R$ implies $|w| \in \mathsf{poly}(|x|)$. We call w a witness for instance x. Define L_R to be the set of inputs x for which there exists a w such that $(x, w) \in R$.

Definition 2.7 *A protocol π between a Prover $\mathcal{P}$ and Verifier $\mathcal{V}$ is a zero-knowledge argument of relation R if it satisfies the following:*

- **Completeness.** *If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on input x and private input w to $\mathcal{P}$, where $(x, w) \in R$, then $\mathcal{V}$ always accepts.*
- **Soundness.** *For any stateful polynomial time prover $\mathcal{P}^* = (\mathcal{P}_1^*, \mathcal{P}_2^*)$ (i.e., $\mathcal{P}_1^*$ and $\mathcal{P}_2^*$ have an access to the same state)*

$$\Pr_{x \leftarrow \mathcal{P}_1^*(1^\kappa, 1^\sigma)} [\forall w : (x, w) \notin R \wedge (\mathcal{P}_2^*(1^\kappa, 1^\sigma) \leftrightarrow \mathcal{V}(x)) = 1] \leq \mathsf{neg}(\kappa).$$

While the standard definition of soundness requires the above to hold for every x*, here we require that it holds only for instances* x *that are output by the dishonest prover* $\mathcal{P}^*$ *itself, hinting to the difficulty of finding an instance* x *for which the equation does not hold.*

– **Special honest verifier zero-knowledge.** *There exists a PPT simulator* $\mathcal{S}$ *such that for every* $z \in \{0,1\}^*$ *the view of the verifier* $\mathsf{View}_{\mathcal{V}}[\mathcal{P}(x;w) \leftrightarrow \mathcal{V}(x;z)]$ *is statistically indistinguishable from* $\mathcal{S}(x,z)$*.*

It is common to turn a zero-knowledge protocol into a non-interactive zero-knowledge protocol using the Fiat-Shamir (FS) transform [FS87, CCH+18, AFK22], for which it is sufficient to consider only an honest verifier. We refer the reader to [GO94] for further discussion on ZK protocols definitions.

3 Secure Key Generation for Threshold Pallier

As a warm up, we refer the reader to Appendix C for the definition of the Paillier additively homomorphic encryption scheme, and Appendix D for a threshold Paillier scheme based on a trusted setup. In this section we present a construction for a threshold Pallier cryptosystem with a trustless setup.

Next, we describe a protocol to realize the ideal threshold Paillier Functionality E.1 ([HMR+19]). The threshold decryption part remains unchanged from the case of a trusted dealer. The key generation part of the protocol is a composition of 2 steps: (i) an RSA UC-secure key-generation from Diogenes [CHI+21]; (ii) a UC-secure protocol by [HMR+19] to generate the verification keys that are used in our Paillier decryption protocol. [HMR+19] first generates a none-standard verification key, namely an El-Gamal encryption of the key share. Nevertheless, they provide a zk proof in that ties this verification key with the traditional one that we use. Importantly, the zk proof in [HMR+19] is also a proof of knowledge. This is crucial as our security proofs require extraction of the adversary's secret key shares. Also, this proof does not rely on N being composed of safe-primes. Hence, proving validity of decryption shares with respect to the traditional verification keys remains the core issue addressed in this work, in Section 4.

We henceforth give a short summary of the ideals used for key generation which are presented in further detail in appendix F. Many protocols for distributed generation of bi-prime (or RSA) modulus were proposed (e.g. [BF97, DdSGMRT21, BDF+23]). In Diogenes the parties obtain a bi-prime $N = PQ$ for *some* large primes P and Q, but in our context (threshold Paillier) we need N to be a *conforming* bi-prime (Definition 2.3). Namely, $N = PQ$ should satisfy: (1) $P = Q = 3 \mod 4$, (2) $\gcd(P-1, Q-1) = 2$, and (3) $\gcd(N, \phi(N)) = 1$. Generating a conforming bi-prime requires only a minor modification to Diogenes that will not affect security. The first condition is already satisfied by Diogenes, resulting $P = Q = 3 \mod 4$. We ensure that the other two conditions are satisfied by invoking the GCD test sub-protocol, which receives one secret and one public value as input and outputs their GCD. The third condition is easily achieved using that GCD test, by running it on inputs $\phi(N)$ (which is secret) and N (which is public) and verifying the result is 1. To verify that the second condition is met we need to manipulate the inputs, since both $P-1$

and $Q-1$ are secrets. First note that $\gcd(P-1,Q-1)=2$ is equivalent to $\gcd\left(\frac{P_i-1}{2},\frac{Q_i-1}{2}\right)=1$. Now, applying the same trick as in in [FS01]), we note that $\gcd(P_i-1,Q_i-1)=\gcd(P-1+(Q-1)P,Q-1)=\gcd(N-1,Q-1)$. And so we run the GCD test on the secret value $Q-1$ and the public value $N-1$. Notably, this additional check is efficient, and fails with probability $\approx\frac{1}{5}$. Therefore, it has almost no effect on the time complexity of the key generation.

Then, using the obtained values above, the parties generate the secret key d and the verification keys. Recall that the secret key should satisfy $d=0 \mod \phi(N)$ and $d=1 \mod N$. As $\phi(N)$ is already shared by the parties, such secret key can be obtained by computing $d=\phi(N)[\phi(N)^{-1} \mod N]\in\mathbb{Z}$, which satisfies both constraints: $d=0 \mod \phi(N)$ as it is a multiple of $\phi(N)$, and $d=1 \mod N$ as $\phi(N)\in\mathbb{Z}_N^*$ (guaranteed by the fact that N is a conforming bi-prime) and so $\phi(N)\cdot\phi(N)^{-1}=1 \mod N$.

The parties obtain a sharing of d using standard techniques, see Hazay et al. [HMR+19, Appendix C.2]. In the same work ([HMR+19]) it is shown how the parties obtain the verification keys v_j as well; below we briefly describe how it works. During the key generation phase the parties obtain $c_j=\mathsf{Enc}_{pk}(d_j)$ for every j, where Enc is the El-Gamal encryption scheme and pk is a joint encryption public key that was previously generated by the parties. Then, the parties sample random bases $\mathbf{g}\in\mathsf{QR}_{N^2}^c$ and finally each party publishes $\mathbf{v_j}=\mathbf{g}^{d_j}$ and proves that c_j is an encryption of $\log_{\mathbf{g}}(\mathbf{v_j})$. The language L_{EQ} is formally described in [HMR+19, Section 3] and the zero knowledge protocol π_{EQ} is presented in [CKY09]. Note that, we could have used the exact same technique of [HMR+19] for our threshold Paillier protocol, namely, whenever a party sends a decryption share $\mathsf{ct}_j=\mathsf{ct}^{d_j}$, it also provides a proof that c_j is an encryption of $\log_{\mathsf{ct}}(\mathsf{ct}_j)$. This, however, would result with an inefficient threshold decryption protocol, as such proof is at least $\times 64$ more expensive than the proof of the language L_{EDL} that we use (see Section 4).

4 Zero-Knowledge Proof of Equality of Discrete Logs

In this section we present the proof of equality of discrete logs, which is utilized to prove the validity of threshold decryptions by the parties. Notably, defining the appropriate language is somewhat subtle.

4.1 Formalizing the Language

Say that d, the secret decryption key, is upper bounded by $\hat{d}$, thus, we define $D=D(\sigma,n,t,\hat{d})$ as the upper bound on the shares $|d_j|$, where $\hat{d}<N^3\cdot n$.

The naïve approach:

$$L_{\mathsf{EDL}'}^{naive}[N,\mathbf{g}',\mathbf{a}]=\{(h',b';x')\mid h',b'\in\mathbb{Z}_{N^2}^*\wedge\mathbf{a}=\mathbf{g}'^{x'}\wedge b'=h'^{x'}\} \tag{2}$$

where (h',b') is the (public) prover's statement, $N,\mathbf{g}',\mathbf{a}$ are the language's parameters, and x' is a witness. The meaning of these values follows:

1. $\mathbf{g}'$ are the base elements chosen in the distributed key generation (DKG) phase.
2. $\mathbf{a}=\mathbf{v}_j=\mathbf{g}'^{2\Delta_n d_j}$ is the verification key associated with the prover P_j.

3. $x' = 2\Delta_n d_j$ is a witness known by the prover P_j (note that we prove soundness and not knowledge soundness).
4. $h' = \mathsf{ct}$ is the ciphertext to be decrypted.
5. $b' = \mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$ is the claimed partial decryption of the prover P_j.

The above formalization raises two issues:

1. The Paillier ciphertext $h' = \mathsf{ct}$ is in $\mathbb{Z}^*_{N^2}$ and so it would be most natural to prove equality of discrete logs over this group. However, since $\mathbb{Z}^*_{N^2}$ is not a cyclic group, we work over the subgroup QR_{N^2}, which raises another issue: deciding membership to QR_{N^2} is assumed to be a computationally hard problem, known as quadratic residuosity problem (QRP).
2. The goal of the zero knowledge proof of the language is to make sure the prover provides (h', b') such that $b' = h'^{2\Delta_n d_j}$. Since $\mathbf{g}'$ might not contain a generator of QR_{N^2}, we may have that for every $g' \in \mathbf{g}'$ $\mathsf{ord}(g') < |\mathsf{QR}_{N^2}|$. In such case an adversary may find $x'' \neq 2\Delta_n d_j \mod |\mathsf{QR}_{N^2}|$ yet $\mathbf{g}'^{x''} = \mathbf{g}'^{2\Delta_n d_j} = \mathbf{a}$. This means that for some h's, picking $b' = h'^{x''}$ gives $(h', b'; x'')$ in the language although $b' \neq h'^{2\Delta_n d_j}$.

To solve the first issue, in the DKG and threshold decryption phases the parties will publish the roots of the elements, and prove statements on their squares. This ensures that the whole proof holds over QR_{N^2}. We stress that we do not need to roll back and prove any statement in $\mathbb{Z}^*_{N^2}$. Instead, after combining the (squared) decryption shares, the reconstructed plaintext will end up being multiplied by 2, and so a final multiplication by $2^{-1} \mod N$ is needed to restore the plaintext. To be more specific, random elements $\mathbf{g}' \leftarrow (\mathbb{Z}^*_{N^2})^c$ is sampled and published in the DKG phase, and we set $\tilde{\mathbf{g}} = \mathbf{g}'^{\Delta_n}$, and $\mathbf{g} = \tilde{\mathbf{g}}^2 = \mathbf{g}'^{2\Delta_n} \in \mathsf{QR}^c_{N^2}$. Similarly, we set $\tilde{h} = \mathsf{ct}^{2\Delta_n}$ and $h = \tilde{h}^2 = \mathsf{ct}^{4\Delta_n} \in \mathsf{QR}_{N^2}$. Finally, we define $\tilde{b} = \mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j} = \tilde{h}^{d_j}$ and $b = \tilde{b}^2 = h^{d_j}$. Under the new syntax, we have:

1. $\mathbf{g}'$ is *some* vector of elements *chosen from* $(\mathbb{Z}^*_{N^2})^c$ in the DKG phase, and $\mathbf{g} = \mathbf{g}'^{2\Delta_n}$.
2. $\mathbf{a} = g^{d_j}$ is the verification key $\mathbf{v}_j = \mathbf{g}'^{2\Delta_n d_j}$ associated with the prover P_j, which is generated in the DKG phase.
3. $x = d_j$ is a witness known by the prover P_j.
4. $\tilde{h} = \mathsf{ct}^{2\Delta_n}$.
5. $\tilde{b} = \mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$ is the claimed partial decryption of the prover P_j.

We get that $\log_h b = \log_{\mathbf{g}} \mathbf{a} = d_j$, and to make sure that $\mathbf{g} \in \mathsf{QR}^c_{N^2}, h, b \in \mathsf{QR}_{N^2}$ we define the language with $\tilde{\mathbf{g}}, \tilde{h}, \tilde{b}$ (instead of $\mathbf{g}, h, b$):

$$L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}] = \{(\tilde{h}, \tilde{b}; x) \mid \tilde{h}, \tilde{b} \in \mathbb{Z}^*_{N^2} \wedge \mathbf{a} = \tilde{\mathbf{g}}^{2x} \wedge \tilde{b}^2 = \tilde{h}^{2x}\}. \tag{3}$$

As a side effect of that formulation we enjoy a shorter witness, that is, in Eq. (2) the witness was $x' = 2\Delta_n d_j$ whereas here (Eq. (3)) the witness is $x = d_j$.

To address the second issue, our soundness argument in Theorem 4.8 implies that $(\tilde{h}, \tilde{b}; d_j) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ for every $(\tilde{h}, \tilde{b})$ provided by the (potentially malicious) prover. That is, we essentially prove that the prover must use d_j as its witness in order for the verifier to accept.

In Section 4.2 we describe the setup phase; we present and prove a zero-knowledge protocol for L_{EDL^2} in Section 4.3; in Sections 4.4-4.6 we show how to batch the protocol over multiple ciphertexts; and finally we present a non-interactive version of the protocol via the Fiat-Shamir transform in Section 4.7.

4.2 Setup Phase

Let us define the Setup algorithm for generating the parameters of the language. We break the setup into two separate phases $\mathsf{Setup} = (\mathsf{Setup}_1, \mathsf{Setup}_2)$. The first one is responsible for generating the conforming bi-prime $N \leftarrow \mathsf{Setup}_1(1^\kappa, 1^\sigma)$. The second one is responsible for generating all the rest, given N: $(\tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}_2(1^\kappa, 1^\sigma, N)$. This algorithm takes 1^κ and 1^σ as inputs and outputs:

- A conforming bi-prime $N = P \cdot Q$.
- $\mathbf{g}'$ is drawn uniformly at random from $(\mathbb{Z}^*_{N^2})^c$ and computes $\tilde{\mathbf{g}} = \mathbf{g}'^{\Delta_n}$ and $\mathbf{g} = \tilde{\mathbf{g}}^2$. We stress that $\mathbf{g}$ does not necessarily contain a generator of QR_{N^2}.
- x is drawn from $[-D, +D]$.
- $\mathbf{a} := \mathbf{g}^x$.

We note that $\mathbf{g}'^2$ is a random vector of elements in $\mathsf{QR}^c_{N^2}$ due to the absence of any known algorithm for computing a generator within this group. This observation underscores the challenge previously mentioned, as conventional protocols for the equality of discrete logarithms with a low soundness error typically assume $\mathbf{g}$ contains a generator (see, for example, [FPS01, DJN10]). We address this challenge by demonstrating that a randomly selected vector of elements contains an "almost generator" of the group with overwhelming probability, as defined below:

Definition 4.1 *Let $\beta \in \mathbb{N}$, let N be a conforming bi-prime, let $g \in \mathcal{G}$ be an element. We say that g is a* β-almost generator *if $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is a β-smooth number.*

Setting $\mathbf{g} = (\mathbf{g}'^2)^{\Delta_n}$, Corollary 4.3 below implies that $\mathbf{g}$ contains a β_{σ_0}-almost generator with probability $> 1 - 2^{-(\sigma+1)}$ where $\beta_{\sigma_0} \leq 2^{\sigma_0+3} \log N$. It turns out that assuming $\mathbf{g}$ to contain an almost generator may suffice to obtain a low soundness error.

Lemma 4.2 *Let $\mathcal{G}$ be a cyclic group and let $g \in \mathcal{G}$ be a uniformly random element, then $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is β-smooth with probability at least $1 - \frac{\log \mathsf{ord}(\mathcal{G})}{\beta \log \beta}$.*

By Lemma 4.2, if we let $\beta_{\sigma_0} := 2^{\sigma_0+2} \log(|\mathcal{G}|)$, then $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is β_{σ_0} smooth with probability greater than $1 - 2^{-(\sigma_0+1)}$.

Corollary 4.3 *Let $\mathcal{G}$ be a cyclic group, $g \in \mathcal{G}$ be a uniformly random element, and $\beta > n$, then $\frac{|\mathcal{G}|}{\mathsf{ord}(g^{\Delta_n})}$ is β-smooth with probability at least $1 - \frac{\log \mathsf{ord}(\mathcal{G})}{\beta \log \beta}$.*

Omitted proofs are in Appendix G. To conclude, the algorithm may fail with probability $2^{-\sigma}$. Specifically, failure means that either N is not a bi-prime or $\mathbf{g}$ does not contain a β_{σ_0}-almost generator. The modulus generation phase can be taken to provide a failure probability of $2^{-\sigma+1}$ which by the union bound gives the desired failure probability.

4.3 The Protocol

The formal description is given in Protocol 4.1.

PROTOCOL 4.1 (Π_{EDL^2}: *ZKP of Equality of Discrete Logs over* QR_{N^2})

Inputs. $\mathcal{P}$ has $(N, \tilde{\mathbf{g}}, \mathbf{a}, \tilde{h}, \tilde{b}; x)$ and $\mathcal{V}$ has $(N, \tilde{\mathbf{g}}, \mathbf{a}, \tilde{h}, \tilde{b})$ where $(N, \tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$ and arbitrary $\tilde{h}, \tilde{b}$. Denote $\mathbf{g} = [\tilde{\mathbf{g}}^2 \mod N^2], h = [\tilde{h}^2 \mod N^2]$ and $b = [\tilde{b}^2 \mod N^2]$.

Protocol.

1. $\mathcal{P}$ samples $r \leftarrow [-2^{2\kappa}D, +2^{2\kappa}D)$ and sends $\mathbf{u} = [\mathbf{g}^r \mod N^2], v = [h^r \mod N^2]$ to $\mathcal{V}$.
2. $\mathcal{V}$ samples $e \leftarrow [0, 2^\kappa)$ and sends e to $\mathcal{P}$.
3. $\mathcal{P}$ sends $z = r - e \cdot x \in \mathbb{Z}$ to $\mathcal{V}$.
4. $\mathcal{V}$ verifies that:
   - $h, b, v \in \mathbb{Z}^*_{N^2}, \mathbf{u} \in (\mathbb{Z}^*_{N^2})^c$ (it suffices to check that $h, b, v, u_1, \ldots, u_c \neq 0 \mod N$ as otherwise we can use these values to factor N),
   - $z \in (-D(2^{2\kappa} + 2^\kappa), +D(2^{2\kappa} + 2^\kappa))$,
   - $\mathbf{u} = \mathbf{g}^z \cdot \mathbf{a}^e \mod N^2$ and $v = h^z \cdot b^e \mod N^2$.

We prove the following.

Theorem 4.4 *The protocol* Π_{EDL^2} *(Protocol 4.1) is a zero-knowledge argument for relation* EDL^2 *(Definition 2.7) under the factoring assumption.*

Proof. In the following we show that all the properties of a zero-knolwdge proof of relation are satisfied.

Completeness.

Let $(N, \tilde{\mathbf{g}}, \mathbf{a}; x)$ be the output of $\mathsf{Setup}(1^\kappa, 1^\sigma)$, then, for every $\tilde{h} \in \mathbb{Z}^*_{N^2}$ and $\tilde{b} = [\tilde{h}^x \mod N^2]$ the protocol's transcript is accepting. The range check of $\tilde{h}, \tilde{b}, \mathbf{u}$ and v obviously goes through, as well as the range check of z, which follows immediately from the ranges of r, e and x. Then, we have

$$\begin{aligned}[\mathbf{g}^z \cdot \mathbf{a}^e \mod N^2] &= [\mathbf{g}^{r-ex} \cdot \mathbf{g}^{ex} \mod N^2] \\ &= [\mathbf{g}^r \mod N^2] = \mathbf{u}\end{aligned}$$

and

$$\begin{aligned}[h^z \cdot b^e \mod N^2] &= [\tilde{h}^{2(r-ex)} \cdot \tilde{h}^{2ex} \mod N^2] \\ &= [\tilde{h}^{2r} \mod N^2] = [h^r \mod N^2] = v.\end{aligned}$$

Special Honest-verifier zero-knowledge (SHVZK). We show that there exists a PPT simulator $\mathcal{S}$, such that for every $(\tilde{h}, \tilde{b}) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ and $e' \in \{0,1\}^\kappa$ it holds that

$$\mathcal{S}_{(N, \tilde{\mathbf{g}}, \mathbf{a})}(\tilde{h}, \tilde{b}, e') \stackrel{c}{\equiv} \left\{ \mathsf{View}(\mathcal{P}(\tilde{h}, \tilde{b}; x) \leftrightarrow \mathcal{V}(\tilde{h}, \tilde{b}, e')) \right\}.$$

The simulator $\mathcal{S}$ samples $z' \leftarrow (-D(2^{2\kappa}+2^{\kappa}), +D(2^{2\kappa}+2^{\kappa}))$, and computes $\mathbf{u}' = [\mathbf{g}^{z'} \cdot \mathbf{a}^{e'} \mod N^2]$ and $v' = [h^{z'} \cdot b^{e'} \mod N^2]$. We argue that the statistical distance between $(\mathbf{u}, v, e', z)$ (of the real execution) and $(\mathbf{u}', v', e', z')$ (of the simulation) is negligible in κ. Note that $(\mathbf{u}, v)$ and $(\mathbf{u}', v')$ are fully determined by $(N, \tilde{\mathbf{g}}, \mathbf{a}, \tilde{h}, \tilde{b}, e', z)$ and $(N, \tilde{\mathbf{g}}, \mathbf{a}, \tilde{h}, \tilde{b}, e', z')$, respectively. We have that z' (in the simulation) is uniformly distributed from $(-D(2^{2\kappa}+2^{\kappa}), +D(2^{2\kappa}+2^{\kappa}))$ independent of e', whereas z (in the real execution) is computed by $z = r - e'x$, where $x \in [-D, +D]$ and r is drawn uniformly from $[-2^{2\kappa}D, +2^{2\kappa}D)$. In the following we show that z and z' are statistically close for every x and e. The distribution of z is as follows:

- $x \geq 0$: z is uniformly distributed over $(-D(2^{2\kappa}+2^{\kappa})-e|x|, D(2^{2\kappa}+2^{\kappa})-e|x|)$.
- $x < 0$: z is uniformly distributed over $(-D(2^{2\kappa}+2^{\kappa})+e|x|, D(2^{2\kappa}+2^{\kappa})+e|x|)$.

In both cases there are $2D(2^{2\kappa}+2^{\kappa}) - e|x|$ values ζ in the range $(-D(2^{2\kappa}+2^{\kappa}), +D(2^{2\kappa}+2^{\kappa}))$ for which $\Pr[z=\zeta] = \Pr[z'=\zeta] = 1/\left(2D(2^{2\kappa}+2^{\kappa})\right)$. For the rest $e|x| \leq 2^{\kappa}D$ values ζ we have $\Pr[z=\zeta] = 0$. Therefore, the distance is

$$\frac{1}{2}\sum_{\zeta} |\Pr[z=\zeta] - \Pr[z'=\zeta]| \;\leq\; \frac{2^{\kappa}D}{2D(2^{2\kappa}+2^{\kappa})} \;<\; 2^{-\kappa},$$

where ζ iterates over $(-D(2^{2\kappa}+2^{\kappa}) - e|x|, +D(2^{2\kappa}+2^{\kappa}) + e|x|)$. □

Soundness. Before moving to our main theorem we present some lemmas used during it's proof. The proofs of the Lemmas are in Appendix G.

Lemma 4.5 *Let $\mathcal{G}$ be cyclic group and let $|\mathcal{G}| = \prod_{i=1}^{k} p_i^{r_i}$ where $p_i \in \mathsf{primes}$ are distinct and $r_i \in \mathbb{N}$ for all $i \in [k]$. Let $a \in \mathcal{G}$ be an element, and denote $\mathsf{ord}(a) = \prod_{i=1}^{k} p_i^{\alpha_i}$ where $\alpha_i \leq r_i$ for all i. Then there exists $b \in \mathcal{G}$ such that $\langle a, b\rangle = \mathcal{G}$ and $\mathsf{ord}(b) = \prod_{i:\alpha_i \neq r_i} p_i^{r_i}$.*

Corollary 4.6 *Let* Factor *be a factoring algorithm. There exists an algorithm* Factor′ *that, given inputs N, x, e such that*

- *$N = PQ$ is a conforming bi-prime; and*
- *$x \in \mathsf{QR}_{N^2}$ satisfies $x \neq 1 \mod N^2$, $e \neq 0$, but $x^e = 1 \mod N^2$,*

outputs P, Q and has a time complexity $\mathsf{time}(\mathsf{Factor}'(N,x,e)) \leq \mathsf{time}(\mathsf{Factor}(e)) + \mathit{polylog}(N)$.

Lemma 4.7 *There exists an algorithm* Factor′*, that given N, x, β as input, where $N = PQ$ is a conforming bi-prime, $1 \neq x \in \mathsf{QR}_{N^2}$, and $\mathsf{ord}(x)$ is β-smooth for some $\beta < N$, outputs P, Q in* $\mathsf{time}(\mathsf{Factor}'(N,x,\beta)) \leq \beta \log^3(N)$.

We prove the following theorem, which constitutes the main contribution of this work:

Theorem 4.8 *Let $\mathcal{P}^* = (\mathcal{P}_1^*, \mathcal{P}_2^*)$ be a stateful* probabilistic *prover such that*

$$\Pr_{\substack{(N,\tilde{\mathbf{g}},\mathbf{a};x)\leftarrow \mathsf{Setup}(1^\kappa,1^\sigma),\\ (\tilde{h},\tilde{b})\leftarrow\mathcal{P}_1^*(x)}}\left[\begin{array}{c|l}(\tilde{h},\tilde{b};x)\notin L_{\mathsf{EDL}^2}[N,\tilde{\mathbf{g}},\mathbf{a}] & N \text{ is a conforming}\\ (\mathcal{P}_2^*(x)\leftrightarrow\mathcal{V}(\tilde{h},\tilde{b}))=1 & \text{bi-prime and}\\ & \tilde{\mathbf{g}}^2 \text{ contains a } \beta_{\sigma_0}\\ & \text{almost generator}\end{array}\right]\geq\varepsilon=\varepsilon(\kappa),$$

where $\mathcal{P}_1^*, \mathcal{P}_2^*$ and $\mathcal{V}$ receive $(1^\kappa, 1^\sigma, N, \tilde{\mathbf{g}}, \mathbf{a})$ implicitly as inputs, and such that $\mathsf{time}(\mathcal{P}_1^*) + \mathsf{time}(\mathcal{P}_2^*) \leq T = T(\kappa)$. Assume further that $\varepsilon - 2^{-\sigma} > 2^{-\kappa+3}$. Then there exists a non-uniform adversary $\mathcal{A}$ that solves the factorization problem (Definition 2.6) with respect to $\mathsf{Setup}_1(1^\kappa, 1^\sigma)$. The adversary $\mathcal{A}$ first applies $(\tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}_2(1^\kappa, 1^\sigma, N)$. Algorithm $\mathcal{A}$ solves this problem with probability $\geq \varepsilon' = \varepsilon'(\kappa)$ and $\mathsf{time}(\mathcal{A}) \leq T' = T'(\kappa)$ such that $\frac{T'}{\varepsilon'} \leq \tilde{\mathcal{O}}(\frac{32\kappa c(T+\beta_{\sigma_0})}{\varepsilon - 2^{-\sigma}})$.

Moreover, there exists a uniform adversary $\mathcal{A}_2$ that solves the factorization problem with probabaility $\geq \varepsilon_2' = \varepsilon_2'(k)$ and $\mathsf{time}(\mathcal{A}_2) \leq T_2' = T'(k)$ such that $\frac{T_2'}{\varepsilon_2'} \leq \tilde{\mathcal{O}}(\frac{16\kappa c(T+\beta_{\sigma_0})}{\varepsilon(\varepsilon - 2^{-\sigma})})$.

As noted earlier and demonstrated in Lemma 4.2, Setup generates a non-conforming bi-prime N or $\tilde{\mathbf{g}}^2$ which does not contain a β_{σ_0}-almost generator with a probability of at most $2^{-\sigma}$. However, the setup phase occurs only once, and the prover cannot affect its outcome. Assuming the setup phase succeeds and that the factoring problem is hard, the theorem states that the probability of a malicious prover cheating the verifier is $\mathsf{neg}(\kappa)$. The distinction between the probability of the setup failing and the probability of the prover cheating plays a crucial role in Section 4.7, where we apply the Fiat-Shamir transform to make the protocol non-interactive.

In addition, the theorem states that except a negligible probability, if the verifier accepts the proof then the prover not only provided $(\tilde{h}, \tilde{b}) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ but also it used the specific witness x that is given from $\mathsf{Setup}(1^\kappa, 1^\sigma)$. Notably, there may exist statements $(\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ for which an adversary is able to convince the verifier. However, this does not contradict the above theorem since such statements are computationally hard to find, as it reduces to factoring N.

Before proving Theorem 4.8, we prove a similar statement, where N is fixed and $\mathcal{P}^*$ is deterministic:

Lemma 4.9 *There exists an oracle machine $\mathcal{A}^{(\cdot)}$ with the following property: Let $\kappa > 0$, and let $\varepsilon \geq \frac{4}{2^\kappa}$. Let $\mathcal{P}^* = (\mathcal{P}_1^*, \mathcal{P}_2^*)$ be a stateful* deterministic *prover, and let us fix a tuple $(N, \tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$ produced by Setup. Suppose that N is a conforming bi-prime, $\tilde{\mathbf{g}}^2$ contains a β_{σ_0}-almost generator and*

$$\Pr_{(\tilde{h},\tilde{b})\leftarrow\mathcal{P}_1^*(x)}\left[\begin{array}{c}(\tilde{h},\tilde{b};x)\notin L_{\mathsf{EDL}^2}[N,\tilde{\mathbf{g}},\mathbf{a}]\\ (\mathcal{P}_2^*(x)\leftrightarrow\mathcal{V}(\tilde{h},\tilde{b}))=1\end{array}\right]\geq\varepsilon,$$

where $\mathcal{P}_1^*, \mathcal{P}_2^*$ and $\mathcal{V}$ receive $(1^\kappa, 1^\sigma, N, \tilde{\mathbf{g}}, \mathbf{a})$ implicitly as inputs. Then $\mathcal{A}^{(\mathcal{P}^*)}$, given the input $(N, \tilde{\mathbf{g}}, \mathbf{a}, \varepsilon)$, outputs a non-trivial factorization of N with probability at least $\frac{1}{8c}$ in time $\tilde{\mathcal{O}}(\frac{T}{\varepsilon} + \beta_{\sigma_0})$, where $T = \mathsf{time}(\mathcal{P}^*(1^\kappa, 1^\sigma, N, \tilde{\mathbf{g}}, \mathbf{a}; x))$.

Proof. First, we introduce an algorithm called $\mathcal{A}_1^{(\cdot)}$, such that for every deterministic prover $\mathcal{P}^*$, the machine $\mathcal{A}_1^{(\mathcal{P}^*)}$ has a time complexity of $\tilde{\mathcal{O}}\left(\frac{T}{\varepsilon}\right)$ (following the notations of the lemma). The purpose of algorithm $\mathcal{A}_1^{(\mathcal{P}^*)}$ is to find an element $d \in \mathsf{QR}_{N^2}$ and an exponent $0 < e' < \frac{2}{\varepsilon}$, such that $d \neq 1$ and $\mathsf{ord}(d^{e'})$ is a β_{σ_0}-smooth number. We show that $\mathcal{A}_1$ does so with probability at least $\frac{1}{8c}$. Then we describe algorithm $\mathcal{A}^{(\cdot)}$ that applies $\mathcal{A}_1^{(\cdot)}$ to factor N. Algorithm $\mathcal{A}_1^{(\mathcal{P}^*)}$ works as follows:

1. **Receiving the claim**: $\mathcal{A}_1^{(\mathcal{P}^*)}$ calls $\mathcal{P}_1^*(1^\kappa, 1^\sigma, N, \tilde{\mathbf{g}}, \mathbf{a}; x)$ and receives $\tilde{h}$ and $\tilde{b}$. Since $\mathcal{P}_1^*$ is deterministic and $(N, \tilde{\mathbf{g}}, \mathbf{a}; x)$ is fixed, the values $\tilde{h}$ and $\tilde{b}$ are fixed as well. Since we have $(\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ with a positive probability, we may deduce that $\tilde{b}^2 \neq \tilde{h}^{2x}$. We denote $b := \tilde{b}^2, h := \tilde{h}^2$, and $d = \frac{b}{h^x}$.
2. **Receiving proofs**: $\mathcal{A}_1^{(\mathcal{P}^*)}$ calls $\mathcal{P}_2^*$ and receives a (deterministic) claim $\mathbf{u}, v$. $\mathcal{A}_1^{(\mathcal{P}^*)}$ then forks the protocol and sends at most ε^{-1} random challenges, until receiving the first valid response. If none of the responses is valid, then $\mathcal{A}_1^{(\mathcal{P}^*)}$ outputs the failure symbol $\perp$. Otherwise, for one of the challenges e_1 it received a valid response. In this case, $\mathcal{A}_1^{(\mathcal{P}^*)}$ forks the protocol and sends the challenges $e_1 + 1, e_1 + 2, \ldots, \min(e_1 + \frac{2}{\varepsilon} - 1, 2^\kappa - 1)$. If none of the responses to these challenges is valid, then $\mathcal{A}_1^{(\mathcal{P}^*)}$ outputs $\perp$. Otherwise, $\mathcal{A}_1^{(\mathcal{P}^*)}$ has two accepting transcripts (e_1, z_1) and (e_2, z_2), such that $e_1 < e_2 < e_1 + \frac{2}{\varepsilon}$. Output $(d, e_2 - e_1)$.

The time complexity of $\mathcal{A}_1^{(\mathcal{P}^*)}$ is dominated by that of $\mathcal{P}^*$ and the exponentiation by x modulo N^2, and is hence bounded by $\tilde{\mathcal{O}}\left(\frac{T}{\varepsilon}\right)$. Let us prove that whenever $\mathcal{A}_1$ does not return $\perp$, its output is valid with high probability. Clearly, $d \in \mathsf{QR}_{N^2}, e' \in (0, \frac{2}{\varepsilon})$, and $d \neq 1$, so it remains to show that $\mathsf{ord}(d^{e'})$ is a β_{σ_0}-smooth number.

Since the transcripts (e_1, z_1) and (e_2, z_2) are both accepting, the following equations hold (modulo N^2):

$$\mathbf{u} = \mathbf{g}^{z_1} \cdot \mathbf{a}^{e_1}, \quad v = h^{z_1} \cdot b^{e_1}, \quad u = \mathbf{g}^{z_2} \cdot \mathbf{a}^{e_2}, \quad \text{and} \quad v = h^{z_2} \cdot b^{e_2}.$$

Denoting $e' = e_2 - e_1, z' = z_2 - z_1$, we obtain

$$1 = \mathbf{g}^{z'} \cdot \mathbf{a}^{e'} \mod N^2 \quad \text{and} \quad 1 = h^{z'} \cdot b^{e'} \mod N^2.$$

Since $\mathbf{g}$ contains a β_{σ_0}-almost generator, we may pick $i \leftarrow [c]$ and set $g = \mathbf{g}_i, a = \mathbf{a}_i$, and deduce that g is a β_{σ_0}-almost generator with probability at least $\frac{1}{c}$. For which by Lemma 4.5, there exists an element $\eta \in \mathsf{QR}_{N^2}$ such that $\mathsf{ord}(\eta)$ is β_{σ_0}-smooth and $\langle g, \eta \rangle = \mathsf{QR}_{N^2}$. Therefore, there exist α, δ such that $h = g^\alpha \eta^\delta$ and consequently

$$1 = g^{z'} \cdot a^{e'} \mod N^2 \quad \text{and} \quad 1 = g^{\alpha \cdot z'} \eta^{\delta \cdot z'} \cdot b^{e'} \mod N^2.$$

Recall that α, δ and η are unknown to $\mathcal{A}_1$. Dividing the second equation by the first equation raised to the power of α we get

$$1 = \frac{g^{\alpha \cdot z'} \eta^{\delta \cdot z'} \cdot b^{e'}}{g^{\alpha \cdot z'} \cdot a^{\alpha \cdot e'}} = \eta^{\delta z'} \left(\frac{b}{a^\alpha}\right)^{e'} \mod N^2. \tag{4}$$

Recall that $a = g^x$. We may deduce that

$$\frac{b}{a^\alpha} = \frac{b}{h^x} \cdot \frac{h^x}{a^\alpha} = d \cdot \frac{g^{\alpha x}\eta^{\delta x}}{g^{\alpha x}} = d\eta^{\delta x} \mod N^2.$$

Then, substituting $\frac{b}{a^\alpha} = d\eta^{\delta x}$ in Eq. (4), we obtain

$$1 = \eta^{\delta z'} \left(d\eta^{\delta x}\right)^{e'} = \eta^{\delta(z'+xe')} d^{e'} \mod N^2.$$

Finally, raising to the power of $\mathsf{ord}(\eta)$, we conclude that

$$1 = \left(\eta^{\delta(z'+xe')} d^{e'} \mod N^2\right)^{\mathsf{ord}(\eta)} = d^{\mathsf{ord}(\eta)e'} \mod N^2.$$

Since $\mathsf{ord}(d^{e'}) | \mathsf{ord}(\eta)$ We may conclude that $\mathsf{ord}(d^{e'})$ is β_{σ_0}-smooth.

It remains to bound the probability that $\mathcal{A}_1^{(\mathcal{P}^*)}$ succeeds. With probability at least $\frac{1}{c}$ $\mathcal{A}_1^{(\mathcal{P}^*)}$ picks $g = \mathbf{g}_i$ which is a β_{σ_0}-almost generator, otherwise we will assume that it fails. In the first part, $\mathcal{A}_1^{(\mathcal{P}^*)}$ sends ε^{-1} challenges. Since $\mathcal{P}^*$ is deterministic, we may define an *accepting challenge* to be a challenge e for which $\mathcal{P}^*$ returns a valid response. Each of the ε^{-1} challenges is accepting with probability at least ε, and the challenges are independent, so the probability that none of them is accepting is at most $(1-\varepsilon)^{\varepsilon^{-1}} \leq e^{-1} \approx 0.37$. Otherwise, $\mathcal{A}_1^{(\mathcal{P}^*)}$ receives e_1 which is sampled uniformly from the set of the accepting challenges.

In the next step, $\mathcal{A}_1^{(\mathcal{P}^*)}$ sends the challenges $e_1+1, e_1+2, \ldots, \min(e_1+\frac{2}{\varepsilon}-1, 2^\kappa-1)$, and fails if none of them is accepting. Let us denote by ℓ the *number* of accepting challenges e_1 for which all the challenges $e_1+1, e_1+2, \ldots, \min(e_1+\frac{2}{\varepsilon}-1, 2^\kappa-1)$ are not accepting, and bound ℓ: The distance between any such two accepting challenges is at least $\frac{2}{\varepsilon}$, and they all lie in $[0, 2^\kappa)$, so $\frac{2}{\varepsilon}(\ell-1) \leq 2^\kappa$, implying that $\ell \leq \frac{\varepsilon 2^\kappa}{2} + 1$. There are at least $\varepsilon 2^\kappa$ accepting challenges, so the probability that a uniform accepting transcript fails in this step is at most

$$\frac{\ell}{\varepsilon 2^\kappa} \leq \frac{\frac{\varepsilon 2^\kappa}{2}+1}{\varepsilon 2^\kappa} = 0.5 + \frac{1}{\varepsilon 2^\kappa} \leq \frac{3}{4}.$$

To conclude, $\mathcal{A}_1^{(\mathcal{P}^*)}$ succeeds with probability at least $c^{-1}\frac{1}{4}(1-e^{-1}) \geq \frac{1}{8c}$.

Now we describe the required algorithm $\mathcal{A}^{(\mathcal{P}^*)}$: It calls $\mathcal{A}_1^{(\mathcal{P}^*)}$ and obtains d and e' as above with a probability of at least $\frac{1}{8c}$. If $\mathcal{A}_1^{(\mathcal{P}^*)}$ outputs $\perp$ then $\mathcal{A}^{(\mathcal{P}^*)}$ outputs $\perp$. Otherwise, $\mathcal{A}^{(\mathcal{P}^*)}$ obtains elements d and $0 < e' < \frac{2}{\varepsilon}$ such that $\mathsf{ord}(d^{e'})$ is β_σ-smooth. If $d^{e'} = 1 \mod N^2$ then, by Corollary 4.6, $\mathcal{A}^{(\mathcal{P}^*)}$ may factor N in time complexity $\mathsf{time}(\mathsf{Factor}(e'))+\mathsf{polylog}(N) = \tilde{\mathcal{O}}(\varepsilon^{-0.5})$. Otherwise, $\mathcal{A}^{(\mathcal{P}^*)}$ obtains that the element $1 \neq d^{e'} \in \mathsf{QR}_{N^2}$ has a β_{σ_0}-smooth order. Thus, by Lemma 4.7, $\mathcal{A}^{(\mathcal{P}^*)}$ may factor N with time complexity $\beta_{\sigma_0}\log^3(N)$. ■

Now we are ready to prove Theorem 4.8:

Proof (Proof of Theorem 4.8). Our proof is based on ideas that are derived from the proof of Theorem 3 in [BG11].

Let $\mathcal{P}^*$ be such a prover. We denote by $\mathcal{P}^*_\omega = (\mathcal{P}^*_{1,\omega}, \mathcal{P}^*_{2,\omega})$ the deterministic prover obtained by fixing the random tape of $\mathcal{P}^*$ to ω. We denote by $\omega_{\mathcal{V}}$ the random tape of the verifier $\mathcal{V}$. We denote by $\tilde{\mathcal{A}}^{(\cdot)}$ the oracle machine described in Lemma 4.9. We further denote

$$p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) = \Pr_{\omega_{\mathcal{V}}}\left[\begin{array}{c}(\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}] \\ (\mathcal{P}^*_2(x) \leftrightarrow \mathcal{V}(\tilde{h}, \tilde{b})) = 1\end{array}\right]$$

if N is a conforming bi-prime and $\tilde{\mathbf{g}}^2$ contains a β_{σ_0}-almost generator, and $p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) = 0$ otherwise.

First, for $i \in \mathbb{N}$, denote $I_i = (2^{-i}, 2^{-i+1}]$, and describe an algorithm $\mathcal{A}_i$ that receives N, samples ω and $\tilde{\mathbf{g}}, \mathbf{a}, x$ (according to Setup_2), and tries to factor N assuming $p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) \in I_i$. Algorithm $\mathcal{A}_i(N)$ works as follows:

1. Sample ω and $(\tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}_2(1^\kappa, 1^\sigma, N)$.
2. Call $\tilde{\mathcal{A}}^{(\mathcal{P}^*_\omega)}(N, \tilde{\mathbf{g}}, \mathbf{a}, 2^{-i})$ and return its output.

By Lemma 4.9, $\mathsf{time}(\mathcal{A}_i) = \tilde{\mathcal{O}}(T2^i + \beta_{\sigma_0})$. Moreover, if $p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) \in I_i$ and $i \leq \kappa - 2$ then $\mathcal{A}_i$ outputs a non-trivial factorization of N with probability at least $\frac{1}{8c}$. Next, we claim that for every κ, there exists $i \leq \kappa - 2$ such that

$$\Pr_{\omega \xleftarrow{\$} \{0,1\}^*, (N, \tilde{\mathbf{g}}, \mathbf{a}, x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)}[p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) \in I_i] \geq \frac{2^i(\varepsilon - 2^\sigma)}{4\kappa}.$$

Let us assume by contradiction that no i satisfies this inequality. Since Setup fails with probability at most $2^{-\sigma}$, we obtain that $\mathbb{E}(p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x)) \geq \varepsilon - 2^{-\sigma}$. However,

$$\begin{aligned}\mathbb{E}(p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x)) &\leq \sum_{i=1}^{\kappa-2} \Pr[p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) \in I_i] \cdot 2^{-i+1} + \\ &\qquad \Pr[p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) \leq 2^{-\kappa+2}] \cdot 2^{-\kappa+2} \\ &\leq 0.5(\varepsilon - 2^{-\sigma}) + 2^{-\kappa+2} < \varepsilon - 2^{-\sigma},\end{aligned}$$

since $\varepsilon - 2^{-\sigma} > 2^{-\kappa+3}$, in contradiction. We may conclude that for every κ, there exists a value $i \leq \kappa - 2$ such that $\Pr[p(\omega, N, \tilde{\mathbf{g}}, \mathbf{a}, x) \in I_i] \geq \frac{2^i(\varepsilon - 2^{-\sigma})}{4\kappa}$. For this value of i, algorithm $\mathcal{A}_i$ succeeds with probability at least $\frac{2^i(\varepsilon - 2^{-\sigma})}{32\kappa c} =: \varepsilon_i$ and has $\mathsf{time}(\mathcal{A}_i) = \tilde{\mathcal{O}}(T2^i + \beta_{\sigma_0})$, and so $\frac{\mathsf{time}(\mathcal{A}_i(N))}{\varepsilon_i} \leq \tilde{\mathcal{O}}(\frac{32\kappa c(T + \beta_{\sigma_0})}{\varepsilon - 2^{-\sigma}})$. We define an advice function $\mathsf{advice}(\kappa)$ that returns this value of i. We also define a non-uniform algorithm $\mathcal{A}$ as follows:

1. $\mathcal{A}$ receives N, κ and $i := \mathsf{advice}(\kappa)$.
2. $\mathcal{A}$ calls $\mathcal{A}_i(N)$.

We obtain that $\mathcal{A}$ succeeds with probability $\varepsilon'(\kappa)$ such that $\frac{\mathsf{time}(\mathcal{A}(N))}{\varepsilon'} \leq \tilde{\mathcal{O}}(\frac{32\kappa c(T+\beta_{\sigma_0})}{\varepsilon-2^{-\sigma}})$.

Finally, we observe that $\mathsf{advice}(\kappa)$ can be computed in $\frac{T}{\varepsilon^2}$. This can be done by running $\mathcal{A}_i$ for each $i \leq \kappa - 2$, for $\mathcal{O}(2^i)$ sampled random tapes (for $\mathsf{Setup}_1, \mathsf{Setup}_2, \mathcal{P}^*$), and observing which i succeeds in factorization. Hence, we can define a uniform adversary $\mathcal{A}_2$ that computes $\mathsf{advice}(\kappa)$ and then calls $\mathcal{A}$, yielding $\mathsf{time}(\mathcal{A}_2)/\varepsilon_2' \leq \tilde{\mathcal{O}}(\frac{32\kappa c(T+\beta_{\sigma_0})}{\varepsilon(\varepsilon-2^{-\sigma})})$, where ε_2' is the success probability of $\mathcal{A}_2'$ rather than $\mathcal{A}$. ■

4.4 Batching

Batching techniques allow a prover to convince a verifier of the correctness of many statements in an efficient way, i.e., much faster then it would take to prove (and verify) each statement alone. In the context of threshold decryption, a good batching technique may shift the bottleneck from the verification of the validity of a partial decryption to the combination of the parties' verified partial decryption into the plaintext.

Recall (from Protocol 4.1) that proving $(\tilde{h}, \tilde{b}; x) \in \mathsf{EDL}^2$ requires raising $\mathbf{g}$ and h to the power of r, which is a large exponents. Then, without batching, proving B statements $(\tilde{h}_i, \tilde{b}_i; x)$ requires performing $2B$ exponentiations with large exponents: namely, raising $\mathbf{g}$ and each h_i to the power of the corresponding large exponent r_i. To improve efficiency, we use the 'small exponent' (SE) technique, introduced in [BGR98] and followed by [APB+04]. The idea of the technique is to combine the $(\tilde{h}_i, \tilde{b}_i)$ statements into a single statement $(\tilde{h}, \tilde{b})$ using a random linear combination, such that $\tilde{h} = \prod \tilde{h}_i^{t_i}$ and $\tilde{b} = \prod \tilde{b}_i^{t_i}$, and then use Protocol 4.1 only once, on the combined $(\tilde{h}, \tilde{b})$. Hence, raising to the power of a large exponent r happens only twice, just like in a proof of a single statement. The efficiency gain by that combination depend on the size of the coefficients t_i, which we show can be much smaller than the size of r without increasing the soundness error of the proof. The resulting proof of B statements requires the prover to raise $2B$ times to the power of a small exponent and then only twice to the power of a large exponent (instead of raising $2B$ times to the power of a large exponent). The same efficiency gain affects the verifier's computational cost as well.

While our batching protocol is similar to existing protocols, our security analysis is novel. Similarly to Theorem 4.8, we use the notion of conforming bi-primes and present a reduction to the factoring problem, without assuming N is a safe bi-prime. Intuitively, the soundness of the batched protocol relies on the fact that it is not possible for the prover to pick statements $(\tilde{h}_i, \tilde{b}_i; x)$, of which at least one is incorrect, such that their random combination $(\tilde{h}, \tilde{b}; x)$ is a correct statement (except for a negligible probability).

We note that using the small exponents technique requires the verifier to pick the coefficients t_i *only after* the prover committed to its statements, which incurs two additional rounds over Protocol 4.1. We show, however, that even this protocol (with five rounds) can be turned non-interactive using the Fiat-Shamir transform without significantly increasing soundness error (see Section 4.7).

The batched proof of equality of discrete logs is formally described in Protocol 4.2.

PROTOCOL 4.2 ($\Pi^B_{\mathsf{EDL}^2}$: *Batched Proof for* EDL^2.)

Inputs. $\mathcal{P}$ has $(N, \tilde{\mathbf{g}}, \mathbf{a}; x)$ and $(\tilde{h}_i, \tilde{b}_i)_{i\in[B]}$, and $\mathcal{V}$ has $(N, \tilde{\mathbf{g}}, \mathbf{a})$ and $(\tilde{h}_i, \tilde{b}_i)_{i\in[B]}$, where $(N, \tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$ and arbitrary $(\tilde{h}_i, \tilde{b}_i)_{i\in[B]}$.

Protocol.

1. $\mathcal{V}$ checks that $\tilde{h}_i, \tilde{b}_i \in \mathbb{Z}^*_{N^2}$, and sends $t_i \leftarrow [0, 2^\kappa)$ to $\mathcal{P}$ for every $i \in [B]$. Then $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{h} = \prod_{i\in[B]} \tilde{h}_i^{t_i}$, $\tilde{b} = \prod_{i\in[B]} \tilde{b}_i^{t_i}$.
2. $\mathcal{P}$ and $\mathcal{V}$ run Π_{EDL^2} (Protocol 4.1) to prove $(\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$.

Completeness follows by the fact that if $\tilde{b}_i = \tilde{h}_i^x$ for all $i \in [B]$ then $\tilde{b} = \left(\prod_{i\in[B]} \tilde{b}_i^t\right) = \left(\prod_{i\in[B]} \tilde{h}_i^{tx}\right) = \left(\prod_{i\in[B]} \tilde{h}_i^t\right)^x = \tilde{h}^x$, and so $(\tilde{h}, \tilde{b}; x) \in \mathsf{EDL}^2$.

As for HVZK, we show that for every $(\tilde{h}_i, \tilde{b}_i)_{i\in[B]}$ such that $(\tilde{h}_i, \tilde{b}_i; x) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ for all i, there exists a PPT simulator $\mathcal{S}$, such that

$$\mathcal{S}_{(N,\tilde{\mathbf{g}},\mathbf{a})}(\{\tilde{h}_i, \tilde{b}_i\}_{i\in[B]}) \stackrel{c}{\equiv} \left\{\mathsf{View}(\mathcal{P}(\{\tilde{h}_i, \tilde{b}_i\}_{i\in[B]}; x) \leftrightarrow \mathcal{V}(\{\tilde{h}_i, \tilde{b}_i\}_{i\in[B]}))\right\}.$$

The simulator $\mathcal{S}$ simply computes $\tilde{h}$ and $\tilde{b}$ as in the protocol, and runs the simulator associated with Π_{EDL^2} (Protocol 4.1) on $(\tilde{h}, \tilde{b})$ and outputs $(\mathbf{u}', v', e', z')$ as output by that simulator. The transcript produced by $\mathcal{S}$ and the one under the real execution are statistically close with the exact same analysis as in the proof of Theorem 4.4.

Next, we argue soundness:

Theorem 4.10 *Let $\mathcal{P}^* = (\mathcal{P}^*_1, \mathcal{P}^*_2)$ be a stateful prover such that:*

$$\Pr_{\substack{(N,\tilde{\mathbf{g}},\mathbf{a};x)\leftarrow \mathsf{Setup}(1^\kappa,1^\sigma),\\ (\tilde{h}_i,\tilde{b}_i)_{i\in[B]}\leftarrow\mathcal{P}^*_1(x)}} \left[\begin{array}{l|l} \exists i : (\tilde{h}_i, \tilde{b}_i; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}] & N \text{ is a conforming} \\ \left(\mathcal{P}^*_2(x) \leftrightarrow \mathcal{V}((\tilde{h}_i, \tilde{b}_i)_{i\in[B]})\right) = 1 & \text{bi-prime and} \\ & \tilde{\mathbf{g}}^2 \text{ contains a } \beta_{\sigma_0} \\ & \text{almost generator} \end{array}\right] \geq \varepsilon,$$

where $\mathcal{P}^*_1, \mathcal{P}^*_2$ and $\mathcal{V}$ receive $(1^\kappa, 1^\sigma, N, \tilde{\mathbf{g}}, \mathbf{a})$ implicitly as inputs. Then assuming factorization is hard, $\varepsilon = \mathsf{neg}(\kappa)$.

Notably, if $\tilde{b}_i = -\tilde{h}_i^x$ for some i, then $(\tilde{h}_i, \tilde{b}_i; x) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ (recall the definition of L_{EDL^2} from Section 4.1) and Protocol 4.2 may succeed. However, as mentioned earlier, decryption shares are squared before being used, so multiplying a decryption share by (-1) does not affect the decryption.

Proof. For brevity, denote the event that the prover attempts to cheat the verifier (assuming a successful setup) by

$$\mathsf{Cheat} = \exists i \in [B] : (\tilde{h}_i, \tilde{b}_i; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}],$$

and the event that $\mathcal{P}^*$ breaks soundness by

$$\mathsf{Break} = [(\mathcal{P}_2^*() \leftrightarrow \mathcal{V}((\tilde{h}_i, \tilde{b}_i)_{i\in[B]})) = 1] \big| \mathsf{Cheat}.$$

Then, the theorem states that $\varepsilon = \Pr[\mathsf{Break}]$ is negligible, because:

$$\begin{aligned}\Pr[\mathsf{Break}] &= \Pr\left[\mathsf{Break} | (\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]\right] \cdot \Pr[(\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2} | \mathsf{Cheat}] \\ &+ \Pr\left[\mathsf{Break} | (\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]\right] \cdot \Pr[(\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2} | \mathsf{Cheat}] \\ &\leq \Pr\left[\mathsf{Break} | (\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]\right] + \Pr[(\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2} | \mathsf{Cheat}].\end{aligned}$$

Hence, we bound $\Pr[\mathsf{Break}]$ by the sum of $\varepsilon_1 := \Pr\left[\mathsf{Break} | (\tilde{h}, \tilde{b}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]\right]$ and $\varepsilon_2 := \Pr[(\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2} | \mathsf{Cheat}]$. We have $\varepsilon_1 = \mathsf{neg}(\kappa)$ by Theorem 4.8 (otherwise we can construct an adversary $\mathcal{P}^*$ who breaks the soundness of Π_{EDL^2} (Protocol 4.1)). In addition, $\varepsilon_2 \leq 2^{-\kappa}$ by Lemma 4.11 below, assuming factorization is hard, which concludes the proof. ■

Lemma 4.11 *Let $\mathcal{P}_1^*$ be a PPT for which*

$$\Pr_{\substack{(N,\tilde{\mathbf{g}},\mathbf{a};x)\leftarrow \mathsf{Setup}(1^\kappa,1^\sigma),\\ (\tilde{h}_i,\tilde{b}_i)_{i\in[B]}\leftarrow\mathcal{P}_1^*(1^\kappa,1^\sigma,N,\tilde{\mathbf{g}},\mathbf{a};x),\\ \{t_i\}\leftarrow[0,M)}} \left[\begin{array}{c|c} \exists i : (\tilde{h}_i, \tilde{b}_i; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}] & N \text{ is a conforming bi-prime} \\ \wedge \quad (\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}] & \end{array}\right] = \varepsilon,$$

where $M \in \mathbb{N}$ is the coefficients domain, $\tilde{h} = \prod_{i\in[B]} \tilde{h}_i^{t_i}$ and $\tilde{b} = \prod_{i\in[B]} \tilde{b}_i^{t_i}$. If $\varepsilon \geq \frac{2}{M}$, then there exist an algorithm that factors N with a time complexity $\tilde{\mathcal{O}}(\varepsilon^{-0.5})$.

Proof. Let $(N, \tilde{\mathbf{g}}, \mathbf{a}; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$, and let $(\tilde{h}_i, \tilde{b}_i)\}_{i\in[B]}$ be the statements produced by $\mathcal{P}_1^*$. Let us assume that $(\tilde{h}_{i_0}, \tilde{b}_{i_0}; x) \notin L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ for some index i_0 and that the probability of $(\tilde{h}, \tilde{b}; x) \in L_{\mathsf{EDL}^2}[N, \tilde{\mathbf{g}}, \mathbf{a}]$ is at least $\varepsilon \geq \frac{2}{M}$. Following the definition of L_{EDL^2}, we may denote $h_i = \tilde{h}_i^2, b_i = \tilde{b}_i^2, h = \tilde{h}^2, b = \tilde{b}^2$, and obtain that $h_{i_0}^x \neq b_{i_0}$ and that the probability of $h^x = b$ is at least $\varepsilon \geq \frac{2}{M}$. WLOG, we may assume that $i_0 = 1$. By definition, $h^x = b$ if and only if $(\Pi_{i\in[B]} h_i^{t_i})^x = \Pi_{i\in[B]} b_i^{t_i}$, which is equivalent to $\Pi_{i\in[B]} \left(\frac{h_i^x}{b_i}\right)^{t_i} = 1$. By isolating the $i_0 = 1$ term we get:

$$\left(\frac{h_1^x}{b_1}\right)^{t_1} = \prod_{i\in[B]\setminus\{1\}} \left(\frac{h_i^x}{b_i}\right)^{-t_i}. \tag{5}$$

There exists a choice of $t_2, \ldots, t_B$, for which Equation (5) holds with a probability of at least ε where t_1 is uniformly distributed. Therefore, there exist at least εM choices of $t_1 \in [0, M)$ such that $\left(\frac{h_1^x}{b_1}\right)^{t_1}$ yields the same result. Assuming $\varepsilon \geq \frac{2}{M}$, we may deduce that $\mathsf{ord}\left(\frac{h_1^x}{b_1}\right) \leq \left(\varepsilon - \frac{1}{M}\right)^{-1} \leq \frac{2}{\varepsilon}$.

To conclude, $\mathsf{ord}\left(\frac{h_1^x}{b_1}\right) = \mathcal{O}(\varepsilon^{-1})$. While $\frac{h_1^x}{b_1}$ is known to $\mathcal{A}$, its order may not be known. Nevertheless, $\mathcal{A}$ can find the order using Pollard's rho algorithm with a time complexity of $\tilde{\mathcal{O}}(\varepsilon^{-0.5})$. Next, $\mathcal{A}$ has an element $\frac{h_1^x}{b_1} \in \mathsf{QR}_{N^2}$ with $\frac{h_1^x}{b_1} \neq 1 \mod N^2$, and knows its order, which is $\mathcal{O}(\varepsilon^{-1})$. Therefore, $\mathcal{A}$ can factor $\mathsf{ord}\left(\frac{h_1^x}{b_1}\right)$ (naively) in time $\tilde{\mathcal{O}}(\varepsilon^{-0.5})$. Therefore, by Corollary 4.6, $\mathcal{A}$ can factor N in time $\tilde{\mathcal{O}}(\varepsilon^{-0.5})$. ■

4.5 Concrete Parameters

In Table 2 we refer to possible instantiations of the Tiresias scheme and compute the concrete statistical and computational security ensured by following the reductions.

Modulus Size	Number of g_i's	Computational Security	Statistical DKG Security
$\log_2(N)$	c	$\kappa - \log_2(32c\kappa)$	$\sigma = \sigma_0 c$
2048	1	112 – 12 = 100	40
3072	2	128 – 13 = 115	80

Table 2: Possible parameter instantiations for Tiresias with $\sigma_0 := 40$. The computational security parameter is controlled by the complexity of factoring (by NIST) as a function of the modulus size $\kappa(\log_2(N))$. The non-uniform reduction in Theorem 4.8 suggests a multiplicative overhead of $32c\kappa$. The statistical security of the DKG is controlled by c, the number of β_{σ_0}-almost generator candidates.

4.6 Batch Verification

Batch verification is a technique that allows a verifier to simultaneously verify proofs from multiple non-interactive provers, thereby reducing computational load. It is somewhat analogous to the batching procedure done using the 'small exponent' method, however since different provers have different verification keys a_j, instead of digesting multiple exponentiation operations into a single one, we get a multi-exponentiation, see [Pip80]. Essentially, instead of validating two (or more) equations $\mathbf{g}^x = 1$ and $h^y = 1$ separately, we may sample randomizers $r_1, r_2 \in [0, 2^\kappa]$ and verify $\mathbf{g}^{r_1 x} \cdot h^{r_2 y} = 1$, and with $1 - \mathsf{neg}(\kappa)$ probability this implies the validity of both (or all) equations. An algorithm for computing $\prod_{i=1}^{C} g_i^{t_i}$ is called a C-multi-exponentiation and can be computed more efficiently than performing C individual exponentiations (namely, $g_i^{t_i}$) and then multiplying the results.

It is important to note that if batch verification fails, the verifier does not know the identity of the cheaters, since all claims were merged into one. In that case, the verifier 'falls back' to verifying each proof individually.

Since the technique is known and security proof is analogous to that of the batching technique, we refer to Appendix H for more details.

4.7 Fiat-Shamir Transform for Batched Proofs

The Fiat-Shamir Transform ([FS87]) is a general tool which enables to transform any public-coin interactive proof into a non-interactive proof in the random oracle model (ROM). A proof with soundness error ε becomes a proof with soundness error at most $Q^{\mu}\varepsilon$ after the transform, where $2\mu+1$ represents the number of rounds, and Q is the number of oracle queries performed by the adversary. In Protocol 4.1 we have $\mu = 1$, which yields the bound $Q\varepsilon$ to the soundness error. However, in the batched ZKP protocol (Protocol 4.2) we have $\mu = 2$, and so we get a soundness error of $2^{-\kappa/2}$ (that is, for an adversary who queries the random oracle $Q = 2^{\kappa/2}$ times, the soundness error is at most $2^{\kappa}\epsilon$). To get a soundness error of $2^{-\kappa}$ *after* the transform, one may aim to get soundness error of $\epsilon = 2^{-2\kappa}$ in the interactive version, and then after the transform it gets soundness error of $2^{-\kappa}$. Therefore, a tighter bound is desired.

The increase of soundness error is inevitable for some protocols (e.g., the non-parallel k-fold repetition of a Σ protocol), but this is not the case for all multi-round protocols. In particular, as shown by Attema et al. [AFK22], if a protocol has '$(k_1, \ldots k_\mu)$-out-of-$(N_1, \ldots, N_\mu)$-special soundness' then applying the Fiat-Shamir transform does not admit a large increase in soundness error. In the following we give the definition from [AFK22] and explain why Protocol 4.2 fits that definition.

Definition 4.12 (Def. 7 in [AFK22]) *A $2\mu+1$ rounds public-coin interactive proof Π for relation R, where $\mathcal{V}$ samples the i^{th} challenge from a set of cardinality $N_i > k_i$, is said to have '$(k_1, \ldots, k_\mu)$-out-of-$(N_1, \ldots, N_\mu)$-special soundness' if there exist a polynomial time algorithm that, on input a statement (x, w) and a $(k_1, \ldots k_\mu)$ tree of accepting transcripts outputs a witness w such that $(x, w) \in R$ with overwhelming probability.*

Note that in [AFK22] they aim to achieve knowledge-soundness (a.k.a extraction of the witness) whereas in our case we do not achieve extraction, but rather a break of the factoring problem.

By the following theorem, applying the Fiat-Shamir transform on a protocol with $(k_1, \ldots, k_\mu)$-out-of-$(N_1, \ldots, N_\mu)$-special soundness only increases soundness error by a factor of Q:

Theorem 4.13 (Thm. 2 in [AFK22]) *The Fiat-Shamir transformation of a $(k_1, \ldots k_\mu)$-out-of-$(N_1, \ldots N_\mu)$ special sound interactive proof Π with soundness $2^{-\kappa}$, has soundness of $(Q+1)2^{-\kappa}$.*

By examining the proof of soundness of Protocol 4.2 (and particularly the proof of Theorem 4.10) one can observe that the protocol has $(2, 2)$-out-of-$(2^{\kappa}, 2^{\kappa})$-special soundness and so the Fiat-Shamir does not affect it significantly.

It is important to note that the Fiat-Shamir transform is not applied on the setup phase that generates $(N, \tilde{g}, a; x)$. Consequently, any potential attacker cannot influence the failure probability of this phase, which remains at most $2^{-\sigma}$. This illustrates why one might opt for σ to be significantly smaller than κ, as it serves as a statistical bound rather than a computational one.

5 Performance

We implemented our threshold Paillier scheme in Rust; the implementation is released as open source at https://github.com/odsy-network/tiresias. In our implementation we use crypto-bigint[10] for constant-time computations over sensitive data to avoid leakage. In addition, we use rayon[11] for parallelism, and our evaluation demonstrates that the scheme can greatly leverage that.

We evaluate the performance of our scheme with number of parties, n, varying from 10 to 1000, and batch sizes, B, varying from 1 (without batching) to 1000. In cases where it applies, we use $t = (2/3)n$ as the threshold. All experiments are conducted over two machine types: (1) AWS EC2 instance of type c6i.24xlarge[12] with 96 3rd generation Intel Xeon Scalable vCPUs @ 3.50GHz, and (2) MacBook Pro Apple M1 Max with 10 Cores @ 3.22GHz.

Our experiments use a 2048-bit bi-prime modulus N (equivalent to 4096-bit Paillier modulus N^2) where the secret Paillier decryption key $d = \phi(N)[\phi(N)^{-1} \bmod N] \in \mathbb{Z}$ is (t, n)-Shamir shared over the integers using the tighter bound on the coefficients as analysed in Appendix B.2 above. All presented runtimes are the average over 10 runs.

As presented in Table 1, existing protocols, notably [FS01] and [HMR+19], which are based on standard hardness assumptions and do not assume a trusted dealer, closely resemble our protocol but require extensive repetitions. Consequently, a performance comparison with these protocols is unnecessary, as our approach is clearly more efficient. Instead, we compare our performance with the "ideal" one, of a semi-honest model with no proofs.

Figures & tables. In Figure 1 and the supporting Table 3 we report the run time for a single party to produce B decryption shares (for B different ciphertexts) when there are n parties.

Then, in Figure 2 and the supporting Table 5 we report the run time for combining the decryption shares from the parties. In the malicious security model the parties also provide a proof of equality of discrete logs to prove the correctness of the decryption shares, in which case we use B-batched proofs, and the 'Mal' columns include the time it takes to verify these.

Finally, in Table 4 we isolate the time it takes to verify a B-batched proof.

Apart for run time, we report on the secret decryption key size and the proof size in bits, which both depend on the number of parties. For $n = \{10, 100, 1000\}$ parties, the size in bits of the key d (which is shared over the integers) is $\{4295, 5324, 19937\}$ and the proof size in bits is $\{12743, 13772, 28385\}$.

[10] https://github.com/RustCrypto/crypto-bigint
[11] https://github.com/rayon-rs/rayon
[12] https://aws.amazon.com/ec2/instance-types/c6i/

Explaining the results. Note that in all figures and tables the number of parties, n, affects the run time. This is due to the fact that n affects the party's share size of the Paillier decryption key when shared over the integers (see Section 2.1 and Appendix B.2), which in turn affects both prover's and verifier's exponent size.

The attentive reader may observe an abrupt jump in run time when increasing from a batch of $B_1 = 100$ ciphertexts to a batch of $B_2 = 1000$ in the C6i machine, and when increasing from $B_1 = 10$ ciphertexts to $B_2 = 100$ in the M1 machine. This jump is due to the parallelism of our implementation, which utilizes up to 96 cores of the C6i machine and up to 10 cores of the M1 machine. Up to B_1 ciphertexts, the workload is quite concurrent and runs simultaneously for all ciphertexts in the batch, whereas above B_1 ciphertexts the work becomes sequential.

Importantly, the figures show that adding protection against a malicious adversary does not incur high overhead. For all n and B this overhead is a small constant and as n and B grow (toward $n = 1000$ or $B = 1000$) this constant reaches 1.5. We present the precise factors in the tables under the '$\times$' column.

The batching technique (along the multi exponentiation described in Section H.1) proves itself necessary if a truly scalable solution is needed. For decryption share computation (Table 3) with $n = \{10, 100, 1000\}$, the prover's time for generating a decryption share for a *single* ciphertext is $\{134, 355, 1434\}$ milliseconds on M1 machine, respectively. When generating decryption shares for $B = 1000$ ciphertexts, the cost is only $\{12.5, 32.5, 144\}$ milliseconds for each one, respectively, which is about $10\times$ improvement. Similar results are obtained for the combination of decryption shares (Table 2) and for proof verification (Table 4).

6 Conclusion

This paper introduces a novel security reduction technique, from the soundness of the proof of equality of discrete logs to the factoring problem. Combining our zero-knowledge proof (and its batching capabilities) with a large scale modulus generation (e.g., Diogenes [CHI+21]), we show for the first time, that threshold Paillier encryption scheme is practical under standard assumptions. In fact, we demonstrate that threshold Paillier is not only practical, but also ready for a large scale deployment with thousands of parties.

		Compute Decryption Share (ms)					
		AWS C6i			MBP M1		
n	B	S.H.	Mal	×	S.H.	Mal	×
10	1	47.7	151.93	3.19	41.8	134.3	3.21
10	10	66.8	233.02	3.49	56.9	210.76	3.70
10	100	160.8	873.98	5.44	526.8	1218.2	2.31
10	1000	1198.4	7424.6	6.20	6336.7	12531.0	1.98
100	1	69.1	193.28	2.80	246.3	355.2	1.44
100	10	85.9	272.0	3.17	336.4	506.59	1.51
100	100	221.6	959.41	4.33	2153.4	2861.2	1.33
100	1000	1666.8	7909.8	4.75	26455.9	32585.0	1.23
1000	1	383.7	790.67	2.06	1081.2	1434.1	1.33
1000	10	415.0	884.33	2.13	2060.3	2475.0	1.20
1000	100	1122.1	2138.8	1.91	13100.6	14058.0	1.07
1000	1000	8034.3	14558.0	1.81	137575.7	143970.0	1.05

Table 3: Computation time, in milliseconds, of B decryption shares when there are n parties (and up to $t = 2n/3$ are corrupted). Times are measured on two types of machines: AWS C6i, with 128 vCPU's and MacBook Pro M1 with 10 cores. S.H and Mal stand for semi-honest and malicious, where the 'malicious' column includes the time it takes to generate the proof of equality of discrete logs, and the × column is the overhead factor Mal/S.H.

		Proof Verification (ms)	
n	B	AWS C6i	MBP M1
10	1	63.8	57.2
10	10	126.1	118.3
10	100	674.0	655.4
10	1000	6188.4	6069.6
100	1	73.7	65.6
100	10	136.0	126.5
100	100	683.6	664.1
100	1000	6215.7	6079.2
1000	1	215.3	187.4
1000	10	277.6	248.5
1000	100	824.9	789.4
1000	1000	6344.5	6224.4

Table 4: Proof verification time in milliseconds, when there are n parties (of which t provides their proofs), each providing a batch proof of B decryption shares. Times are measured on two types of machines: AWS C6i, with 128 vCPU's and MacBook Pro M1 with 10 cores.

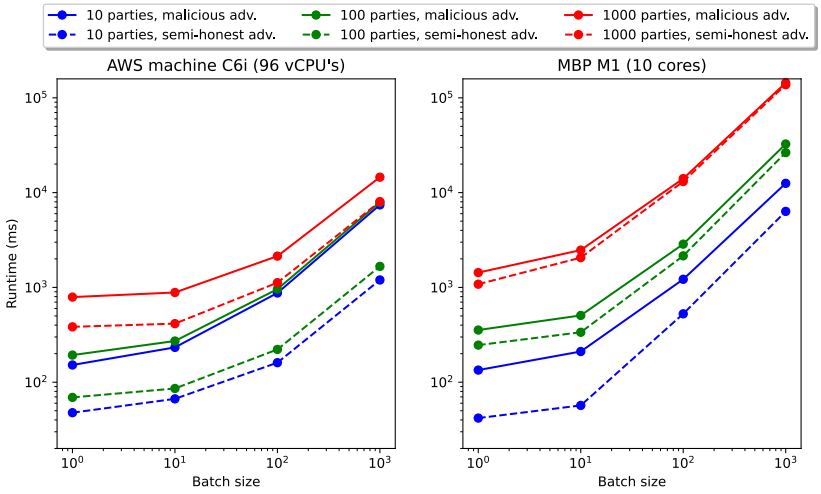

Fig. 1: Time in milliseconds to generate a decryption share with and without proof of equality of discrete logs (e.g., in the presence of a malicious and semi-honest adversaries, resp.).

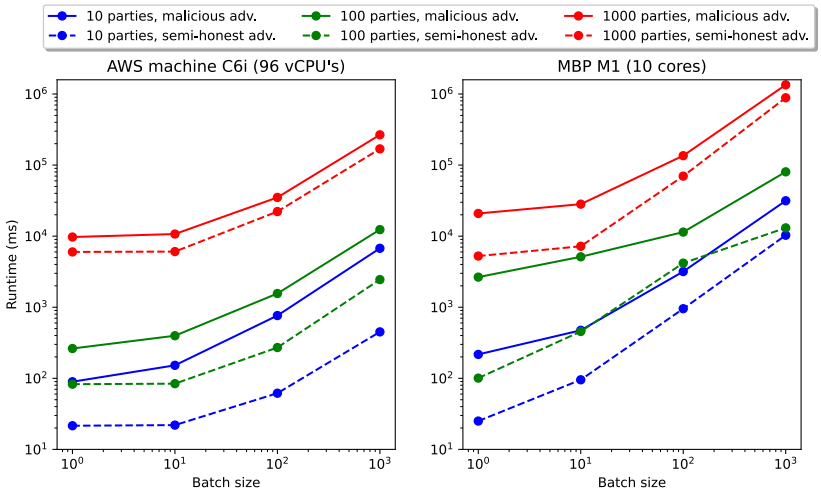

Fig. 2: Time in milliseconds to combine decryption shares from $t = 2n/3$ parties of B ciphertexts, with and without proof of equality of discrete logs (e.g., in the presence of a malicious and semi-honest adversaries, resp.).

		Combining Decryption Shares (ms)					
		AWS C6i			MBP M1		
n	B	S.H.	Mal	×	S.H.	Mal	×
10	1	21.517	89.517	4.16	25.027	216.82	8.66
10	10	21.958	152.28	6.94	95.336	473.6	4.97
10	100	61.703	763.5	12.37	953.03	3178.2	3.33
10	1000	450.11	6746.1	14.99	10303.0	31480.0	3.06
100	1	82.635	262.53	3.18	100.69	2653.4	26.35
100	10	84.031	397.24	4.73	455.83	5124.2	11.24
100	100	270.96	1559.8	5.76	4173.4	11409.0	2.73
100	1000	2449.3	12362.0	5.05	13083.0	80335.0	6.14
1000	1	5997.6	9722.5	1.62	5249.1	20851.0	3.97
1000	10	6064.8	10709.0	1.77	7201.5	28201.0	3.92
1000	100	22178.0	35058.0	1.58	69940.0	135580.0	1.94
1000	1000	169050.0	266640.0	1.58	884770.0	1345200.0	1.52

Table 5: Time it takes, in milliseconds, to combine a batch of B decryption shares (of t-out-of-n-Shamir sharing, with $t = 2n/3$). S.H and Mal stand for semi-honest and malicious, where the 'malicious' column includes the time it takes to verify the proof, and the × column is the overhead factor Mal/S.H. Times are measured on two types of machines: AWS C6i, with 128 vCPU's and MacBook Pro M1 with 10 cores.

References

ACS02. Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Advances in Cryptology-CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18-22, 2002 Proceedings 22*, pages 417–432. Springer, 2002.

AFK22. Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, pages 113–142. Springer, 2022.

APB+04. Riza Aditya, Kun Peng, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Batch verification for equality of discrete logarithms and threshold decryptions. In *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings 2*, pages 494–508. Springer, 2004.

BBBF18. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I*, pages 757–788. Springer, 2018.

BDF+23. Jakob Burkhardt, Ivan Damgård, Tore Kasper Frederiksen, Satrajit Ghosh, and Claudio Orlandi. Improved Distributed RSA Key Generation Using the Miller-Rabin Test. *Cryptology ePrint Archive*, 2023.

BDO22. Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure Multiparty Computation from Threshold Encryption based on Class Groups. *Cryptology ePrint Archive*, 2022.

BDTZ16. Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Zakarias. Better preprocessing for secure multiparty computation. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, pages 327–345. Springer, 2016.

BF97. Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology-CRYPTO 97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17-21, 1997 Proceedings 17*, pages 425–439. Springer, 1997.

BG11. Mihir Bellare and Oded Goldreich. On probabilistic versus deterministic provers in the definition of proofs of knowledge. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 114–123, 2011.

BGR98. Mihir Bellare, Juan A Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology-EUROCRYPT98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31-June 4, 1998 Proceedings 17*, pages 236–250. Springer, 1998.

BS21. Omar Rafik Merad Boudia and Sidi Mohammed Senouci. An Efficient and Secure Multidimensional Data Aggregation for Fog-Computing-Based Smart Grid. *IEEE Internet Things J.*, 2021.

CCH+18. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N Rothblum, and Ron D Rothblum. Fiat-Shamir from simpler assumptions. *Cryptology ePrint Archive*, 2018.

CHI+21. Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 590–607. IEEE, 2021.

CKY09. Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the Portability of Generalized Schnorr Proofs. In *EUROCRYPT*, volume 5479, pages 425–442. Springer, 2009.

DdSGMRT21. Cyprien Delpech de Saint Guilhem, Eleftheria Makri, Dragos Rotaru, and Titouan Tanguy. The return of eratosthenes: Secure generation of rsa moduli using distributed sieving. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 594–609, 2021.

DJN10. Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9:371–385, 2010.

DK01. Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In *Advances in Cryptology-EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6-10, 2001 Proceedings 20*, pages 152–165. Springer, 2001.

DN03. Ivan Damgård and Jesper Buus Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, volume 7417, pages 643–662. Springer, 2012.

FPS01. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography: 4th International Conference, FC 2000 Anguilla, British West Indies, February 20-24, 2000 Proceedings 4*, pages 90–104. Springer, 2001.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology-CRYPTO86: Proceedings 6*, pages 186–194. Springer, 1987.

FS01. Pierre-Alain Fouque and Jacques Stern. Fully distributed threshold RSA under standard assumptions. In *Advances in Cryptology-ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9-13, 2001 Proceedings 7*, pages 310–330. Springer, 2001.

GGN16. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, pages 156–174. Springer, 2016.

GO94. Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

HMR+19. Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32:265–323, 2019.

KL14. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2nd edition, 2014.

KLM+20. Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *EuroS&P*, 2020.

MT21. Dimitris Mouris and Nektarios Georgios Tsoutsos. Masquerade: Verifiable Multi-Party Aggregation with Secure Multiplicative Commitments. 2021.

MV06. Hugh L. Montgomery and Robert C. Vaughan. *Multiplicative Number Theory I: Classical Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2006.

NS10. Takashi Nishide and Kouichi Sakurai. Distributed Paillier Cryptosystem without Trusted Dealer. In *Information Security Applications - 11th International Workshop, WISA 2010, Jeju Island, Korea, August 24-26, 2010, Revised Selected Papers*, volume 6513 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2010.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology-EUROCRYPT 99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2-6, 1999 Proceedings 18*, pages 223–238. Springer, 1999.

Pip80. Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980.

Pol74. John M Pollard. Theorems on factorization and primality testing. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 76, pages 521–528. Cambridge University Press, 1974.

Rab98. Tal Rabin. A Simplified Approach to Threshold and Proactive RSA. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 1998.

RSA78. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.

SB21. István András Seres and Péter Burcsi. A note on low order assumptions in RSA groups. *Rad Hrvatske akademije znanosti i umjetnosti. Matematičke znanosti*, (546= 25):15–31, 2021.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Sho00. Victor Shoup. Practical threshold signatures. In *Advances in Cryptology-EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*, pages 207–220. Springer, 2000.

VAS19. Thijs Veugen, Thomas Attema, and Gabriele Spini. An implementation of the Paillier crypto system with threshold decryption without a trusted dealer. *ePrint*, 2019.

A Mathematical background.

Let $a, b \in \mathbb{Z}$. If $a = b \cdot q + r$ for some $q \in \mathbb{Z}$ and $r \in [0, b)$ then $r = [a \mod b]$ represents the reduction of a modulo b. If $r = 0$ we say that b *divides* a and denote it by $b|a$. If, in addition, $b \notin \{1, a\}$, then b is a *non-trivial factor* of a. We write $a_1 = a_2 = \ldots = a_k \mod b$ if $[a_i \mod b] = [a_j \mod b]$ for all i, j. The values $\gcd(a, b)$ and $\mathsf{lcm}(a, b)$ are the greatest common divisor and least common multiple of a and b, respectively. If $\gcd(a, b) = 1$ we say that a and b are *co-prime*. In the following we present some basic number theoretic properties that can be found in introductory books (e.g., [KL14, Section 7]).

Proposition A.1 *Let $a, b \in \mathbb{Z}$. if $c = [ab \mod b^2]$ then $a = c/b \mod b$.*

Proposition A.2 *Let $m \in \mathbb{Z}$ and $x, r \in [0, m-1]$. Then:*

- *Euclidean algorithm:* $\gcd(m, x) = \gcd(m, [x \mod m])$.
- $\gcd(m, xr) = 1$ *if and only if* $\gcd(m, x) = 1$ *and* $\gcd(m, r) = 1$.

Proposition A.3 *Let $a, m_1, m_2, \ldots \in \mathbb{Z}$ and $\gcd(m_i, m_j) = 1$ for all $i \neq j$. Then, $\gcd(a, \prod_i m_i) = \prod_i \gcd(a, m_i)$.*

Corollary A.4 *Let $a \in \mathbb{Z}$ and $p_1, p_2, \ldots \in$ primes: $\gcd(a, p_i) = 1$ for all i if and only if $\gcd(a, \prod_i p_i) = 1$.*

Number classification and properties. We use some classical number theoretic notions. The reader is referred to [MV06, Section 7] for more information.

B Shamir Secret Sharing

B.1 Shamir Secret Sharing over a Prime Field

Shamir t-out-of-n secret sharing over the field $\mathbb{F}$ (where $t < n \in \mathbb{N}$) is defined by a tuple of algorithms $\mathsf{SS}_{\mathbb{F}} = (\mathsf{Share}, \mathsf{Reconstruct})$, where $[s] = ([s]_1, \ldots, [s]_n) = \mathsf{Share}_{t,n}(s; r)$ denotes a sharing of s using randomness r, and $s = \mathsf{Reconstruct}([s]_{j_1}, \ldots, [s]_{j_{t+1}})$ denotes the reconstruction using $t+1$ shares, which may result with $\perp$ if the shares are inconsistent. With more details:

- $[s] = \mathsf{Share}_{t,n}(s; r)$. Given a secret $s \in \mathbb{F}$ and a random tape $r = (a_1, \ldots, a_t) \in \mathbb{F}^t$, output $[s] = ([s]_1, \ldots, [s]_n)$, where $[s]_j = p(j)$ and $p(x) = s + a_1 x + a_2 x^2 + \ldots + a_t x^t$.
- $s = \mathsf{Reconstruct}(\{(j, [s]_j)\}_{j \in T})$. Let $T \subset [n]$ be a set of $t+1$ distinct elements from $[n]$. Let p be the unique interpolation polynomial such that $p(j) = [s]_j$ for all $j \in T$. Output $s = p(0)$.

Lagrange interpolation is used in order to get $p(j)$ directly. For a set $T \subset [n]$ of $t+1$ distinct elements, given points $\{(j, [s]_j)\}_{j \in T}$, the polynomial that passes through them is $p(x) = \sum_{j \in T} [s]_j \cdot \ell_{T,j}(x)$, where

$$\ell_{T,j}(x) = \prod_{\substack{i \in T \\ i \neq j}} \frac{x - i}{j - i}.$$

Now, for any subset T of size $t+1$, every $j \in T$ and every evaluation point $v \in \mathbb{F}$, the Lagrange coefficient is defined as $\lambda^v_{T,j} = \ell_{T,j}(v)$. Consequently, we can write $p(v) = \sum_{j \in T} \lambda^v_{T,j} \cdot [s]_j$.

Local operations over sharings. Given two polynomials $p_1(x)$ and $p_2(x)$ with secrets $s_1 = p_1(0)$ and $s_2 = p_2(0)$, the secret sharing of $s = s_1 + s_2$ can be locally computed by having each party P_j compute $[s]_j = [s_1]_j + [s_2]_j$ so that the secret s is shared using the polynomial $p(x) = (p_1 + p_2)(x)$. In addition, given a polynomial $p(x)$ and a (public) constant $\alpha \in \mathbb{F}$, the secret sharing of $\alpha \cdot s$ can be locally computed by having each party P_j compute $[\alpha s]_j = \alpha [s]_j$. Similarly, the above is extended to any affine operation over the secrets.

B.2 A Tighter Bound of Shamir Shares Over the Integers

Recall that $[0, b]$, $b \in \mathbb{N}$, is the range of the secret, n is the number of parties and σ is the statistical security parameter. We analyze the required value of $I(\sigma, n, b)$ mentioned in Section 2.1 for the security of the Shamir sharing over the integers.

Our analysis is based on [BDO22]. First we cite the relevant definitions:

Definition B.1 ([BDO22, Definition 8]) *A secret sharing scheme is statistically private if for any set of corrupted parties $C \subseteq \{P_1, \ldots, P_n\}$ with $|C| \leq t$, any two secrets $\alpha, \alpha' \in [0, b]$ and independent random coins r, r', we have that the statistical distance between $\{Share_i(\alpha, r) \mid i \in C\}$ and $\{Share_i(\alpha', r') \mid i \in C\}$ is negligible in σ.*

Definition B.2 ([BDO22, Definition 9]) *Let $C \subseteq [n]$ such that $|C| = t$. Then we define the sweeping polynomial $h_C(X) = \sum_{i=0}^{t} h_{C,i} X^i$ as the unique polynomial of degree at most t such that $h(0) = \Delta_n$ and $h(i) = 0$ for all $i \in C$. Moreover, let h_{max} be an upper bound on the coefficients of the sweeping polynomials, i.e., $h_{max} \geq \max\{|h_{C,i}| \mid i \in [0, t], C \subseteq [n], |C| = t\}$.*

In our analysis we use the following result (restated):

Theorem B.3 ([BDO22, Theorem 16]) *The protocol described in Section 2.1 is statistically private when the coefficients are sampled from $[0, I(\sigma, n, b)]$ for $I(\sigma, n, b) \geq 2^{\sigma+2} b h_{max} t$.*

Therefore, in order to derive concrete bound for $I(\sigma, n, b)$, it suffices to bound $h_{\max}$. For a set $C \subseteq [n]$, denote $\Delta_C = \Delta_n \left(\prod_{j \in C} j \right)^{-1}$. We bound $h_{\max}$ using the following lemma:

Lemma B.4 (implied in [BDO22, Theorem 16]) *For every set $C \subseteq [n]$ with $|C| = t$, we have*

$$h_C(X) = \Delta_C \cdot \prod_{j \in C} (j - X).$$

which is trivially correct by evaluating on 0 and C. Then:

Lemma B.5 $h_{max} < (t+1)\Delta_n$.

Proof. Let $C \subseteq [n]$ with $|C| = t$. By Lemma B.4, we may write

$$h_C(X) = \sum_{i=0}^{t} h_{C,i} X^i = \Delta_C \cdot \prod_{j \in C} (j - X).$$

In order to prove that $|h_{C,i}| \leq (t+1)\Delta_n$, it suffices to prove $\sum_i |h_{C,i}| \leq (t+1)\Delta_n$. Notice that $h_C(-X) = \Delta_C \cdot \prod_{j \in C}(j + X)$ and therefore all h_C's coefficients are positive and we can write $h_C(-X) = \sum_{i=0}^{t} |h_{C,i}| X^i$. Therefore,

$$\sum_{i=0}^{t} |h_{C,i}| = h_C(-1) = \Delta_C \cdot \prod_{j \in C} (j+1) = \Delta_n \cdot \prod_{j \in C} \left(\frac{j+1}{j} \right) \leq (t+1)\Delta_n.$$

where the first two equalities hold by applying $h_C(-1) = \sum_{i=0}^{t} |h_{C,i}| \cdot 1 = \Delta_C \cdot \prod_{j \in C}(j+1)$, the third equality holds by substituting $\Delta_C = \Delta_n \left(\prod_{j \in C} j \right)^{-1}$ and the inequality holds since $\prod_{j \in C} \left(\frac{j+1}{j} \right) \leq \left(\frac{2}{1} \right) \cdot \left(\frac{3}{2} \right) \cdot \ldots \cdot \left(\frac{t+1}{t} \right) = (t+1)$.

We conclude that the coefficients for Shamir secret sharing over the integers, when the secret is from $[0, b]$, are uniformly sampled from $[0, I(\sigma, n, b)]$ where $I(\sigma, n, b) \geq 2^{\sigma+2} \cdot b \cdot t \cdot (t+1) \cdot \Delta_n$. From this we conclude that the shares to the parties are bound by

$$\begin{aligned} \sum_{i=0}^{t} a_i x^i &\leq \sum_{i=0}^{t} I(\sigma, n, b) n^i = I(\sigma, n, b) \sum_{i=1}^{t} n^i = I(\sigma, n, b) \frac{n^{t+1} - 1}{n - 1} \\ &\leq I(\sigma, n, b) \frac{n^{t+1}}{n/2} = 2I(\sigma, n, b) n^t \end{aligned}$$

where the first inequality is since the coefficients a_i's are bounded by $I(\sigma, n, b)$ and the party's evaluation point is at most n; the first equality is trivial; the second equality is the result of the sum of a geometric series $n^0, n^1, n^2, \ldots, n^t$; then the second inequality is by reducing 1 from both nominator and denominator (which increases the value) and writing $n/2$ instead of n in the denominator (which again increases the value as long as $n > 2$); and the last equality is trivial.

C The Paillier Encryption Scheme

In this section we present the Paillier encryption scheme [Pai99], which is characterized by the following tuple of algorithms $\mathsf{Paillier} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

- $\mathsf{Gen}(1^\kappa)$. Given a security parameter 1^κ, sample $\ell = \mathsf{poly}(\kappa)$-bit primes P, Q such that $\gcd(N, \phi(N)) = 1$. The algorithm outputs the public encryption key $N = P \cdot Q$ and the secret decryption key $(N; d)$ where $d = 0 \mod \phi(N)$ and $d = 1 \mod N$.
- $\mathsf{Enc}(N, \mathsf{pt}; r)$. Given the public key N, a plaintext $\mathsf{pt} \in \mathbb{Z}_N$ and randomness $r \in \mathbb{Z}_N^*$, output ciphertext

$$\mathsf{ct} = \left[(1+N)^{\mathsf{pt}} \cdot r^N \mod N^2\right].$$

- $\mathsf{Dec}((N, d), \mathsf{ct})$. Given the secret key d and a ciphertext ct, output

$$\mathsf{pt} = \left[\frac{[\mathsf{ct}^d \mod N^2] - 1}{N} \mod N\right].$$

Correctness. Decryption is always correct because:

$$\begin{aligned}\left[\frac{[\mathsf{ct}^d \mod N^2] - 1}{N} \mod N\right] &= \left[\frac{\left[(1+N)^{d\cdot\mathsf{pt}} \cdot r^{d\cdot N} - 1 \mod N^2\right]}{N} \mod N\right] \\ &= \left[\frac{\left[d \cdot \mathsf{pt} \cdot N \mod N^2\right]}{N} \mod N\right] \\ &= [d \cdot \mathsf{pt} \mod N] = \mathsf{pt}\end{aligned}$$

where equalities (top to bottom) hold by replacing ct with the definition of encryption; the binomial expansion $(1+N)^k = 1 + Nk \mod N^2$ and the fact that $r^{d\cdot N} = 1 \mod N^2$ (since $\phi(N) | d$ and $\mathsf{ord}(\mathbb{Z}_{N^2}) = N\phi(N)$); Proposition A.1; and the fact that $d = 1 \mod N$ and that $\mathsf{pt} < N$.

D Warm-Up: Threshold Paillier with a Trusted Setup

Here we present the standard approach for threshold decryption of Paillier ciphertexts by [FPS01], and for simplicity assume the key generation is done by a trusted dealer. Namely, consider a trusted dealer, who runs $(N; d) \leftarrow \mathsf{Paillier.Gen}(1^\kappa)$. Then the dealer publishes N and shares d *over the integers* to n parties, $P_1, \ldots, P_n$, using a (t, n)-sharing scheme (cf. 2.1), resulting with $[d]$ such that P_j receives d_j. The sharing of d is over $\mathbb{Z}$ rather than over $\mathbb{Z}_{|\mathsf{QR}_{N^2}|}$ as the parties cannot compute the Lagrange coefficients over $\mathbb{Z}_{|\mathsf{QR}_{N^2}|}$ because the order $|\mathsf{QR}_{N^2}|$ is unknown.

In the following we assume that the parties behave honestly and later we describe how to handle malicious behaviour. Given a ciphertext $\mathsf{ct} \in \mathbb{Z}_{N^2}^*$, party

P_j broadcasts its decryption share $\mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$ (recall that $\Delta_n = n!$). Given ct_j for all $j \in T$, where $T \subset [n]$ and $|T| = t+1$, decrypt by computing:

$$\begin{aligned}
\mathsf{ct}' := \left[\prod_{j\in T} \mathsf{ct}_j^{2\Delta_n \lambda^0_{T,j}} \mod N^2\right] &= \left[\prod_{j\in T} (\mathsf{ct}^{2\Delta_n d_j})^{2\Delta_n \lambda^0_{T,j}} \mod N^2\right] \\
&= \left[\prod_{j\in T} \mathsf{ct}^{4\Delta_n^2 d_j \lambda^0_{T,j}} \mod N^2\right] \\
&= \left[\mathsf{ct}^{4\Delta_n \sum_{j\in T} \Delta_n d_j \lambda^0_{T,j}} \mod N^2\right] \\
&= \left[\mathsf{ct}^{4\Delta_n^3 d} \mod N^2\right] \\
&= \left[\left(\mathsf{ct}^{4\Delta_n^3}\right)^d \mod N^2\right] \\
&= \mathsf{Enc}\left(N, 4\Delta_n^3 \mathsf{pt}; r^{4\Delta_n^3}\right)^d,
\end{aligned}$$

where the first equality holds since $\mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$, the third follows by Lagrange interpolation in the exponent, since we have $\left(\sum_{j\in T} \Delta_n d_j \lambda^0_{T,j}\right) = \Delta_n^2 d$) by Eq. (1), and the last one follows by the encryption definition.

Then, obtain the plaintext by computing

$$\left[\left(\frac{\mathsf{ct}'-1}{N}\right) \cdot (4\Delta_n^3)^{-1} \mod N\right] = \left[\mathsf{pt} \cdot 4\Delta_n^3 \cdot (4\Delta_n^3)^{-1} \mod (N)\right] = \mathsf{pt},$$

as the first equality is derived from the correctness of the Paillier scheme.

Handling corrupted parties. To detect a malicious party P_j that sends an incorrect decryption share ct_j, we require P_j to send a zk-proof that ct_j is computed correctly using the public base ct and the secret share d_j. As discussed in Section 1.1, several approaches were proposed in the literature, and here we follow the one taken by Damgård et al. [DJN10], which uses a single verification key. In more detail, in addition to the sharing $[d]$, the trusted dealer computes and publishes g and $v_j = g^{\Delta_n d_j}$ for every P_j, where g is a random element in QR_{N^2}. Then, whenever P_j wishes to send its decryption share $\mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$, it also sends a proof that the discrete log of ct_j in the basis ct^2 equals the discrete log of v_j in the basis g. In that sense, v_j is a commitment to P_j's secret share d_j.

We remark that the discrete logarithm equality described above does not assure that P_j behaves honestly, and a slightly stronger claim needs to be proved. See Section 4 for the exact formulation of the language and the proof protocol.

E Threshold Decryption Ideal Functionality

We describe an ideal functionality for threshold decryption based on [HMR+19] and modified to the threshold regime.

FUNCTIONALITY E.1 (*Threshold Decryption Functionality* $\mathcal{F}_{\mathsf{thresh}}$)

The functionality interacts with a set of n parties $\{\mathcal{P}_i\}_{i\in[n]}$ and an adversary $\mathcal{A}$ which corrupts a subset U of the parties.

1. **Key Generation.** Upon receiving $(\mathsf{generate}, \mathsf{sid}, \mathcal{P}_\mathsf{i})$ from each party send to $\mathcal{A}$ $(\mathsf{randinput})$ an awaits for the adversary to reply with $(\mathsf{geninput}, \{\mathsf{r_i}\}\mathsf{i} \in \mathsf{U})$. The functionality then runs the key generation algorithm using the adversary randomness for parties in U. It then saves the result (pk, sk, sid) to memory and sends pk to $\mathcal{A}$. Upon receiving the public key the adversary may reply with **continue** in which case it sends pk to all parties. Else it sends **abort** to all parties.
2. **Decryption.** Upon receiving $(\mathsf{decrypt}, \mathsf{c}, \mathsf{sid}, \mathcal{P}_\mathsf{i})$ check if there exist a secret key for sid, if so save $(c, sid, \mathcal{P}_i)$ to memory. If there $t+1$ such messages saved to memory calculate $\mathsf{pt} = \mathsf{Dec}(c)$ to the adversary and await to receive $(\mathbf{abort}, U')$. If $U' \cap U = \emptyset$ broadcast (pt, sid) to all parties. Else broadcast $(\mathbf{abort}, sid, U' \cap U)$ to all parties.

F Full Description of the DKG protocol

In this section we describe the DKG protocol in detail. Since the DKG protocol is composed of sub-protocols from existing literature, we first review those and then describe the overall protocol.

F.1 Bi-Prime Generation Protocol

A protocol $(N; P_i, Q_i) \leftarrow \mathsf{BiPrimeGen}(n, 1^\kappa, 1^\sigma, \mathsf{sid})$ is an interactive protocol between n parties which returns $N = PQ$ such that:

1. Denoting $P := \sum_{i\in[n]} P_i$, $Q := \sum_{i\in[n]} Q_i$, it holds that $N = PQ$.
2. P, Q are prime numbers of size ℓ corresponding to a computational security parameter κ of the factoring problem.
3. $P \equiv Q \equiv 3 \mod 4$.
4. N is a bi-prime with probability $\geq 1 - 2^{-\sigma}$.

The protocol should be UC-secure and tolerate t corrupted parties. In particular, there should be a reduction from an adversary that factors challenges N generated from $\mathsf{BiPrimeGen}$ to an adversary that factors challenges N generated by $\mathsf{GenModulus}$, the challenger in the factoring game. We refer the reader to [BF97] and [CHI+21] for full details.

In addition, we require the protocol to have identifiable abort. Since the parties in the protocol have no secret input, this can be trivially achieved by letting each party P_i reveal the random tape $\mathcal{H}(r_i, \mathsf{sid})$ in case a DKG protocol with session id sid fails.

F.2 GCD Test Protocol

A protocol $b \leftarrow \mathsf{GCD}(n, 1^\kappa, 1^\sigma, t, x; y_i)$ is an interactive protocol between n parties which returns a boolean value $b \in \{0, 1\}$ such that:

1. If $b = 1$ then $\gcd\left(x, \sum_{i\in[n]} y_i\right) | t$.
2. If $\gcd\left(x, \sum_{i\in[n]} y_i\right) = t$ then $b = 1$ with probability $\alpha(x) > 0$.

Notice that one may reduce the false negative probability $1 - \alpha(x)$ as close to zero as is optimal for a specific use-case. In our context we will take $\alpha(x) > \frac{\pi^2}{10}$. Such protocol is often used as a part of a bi-prime generation protocol. We refer to [CHI+21] for the protocol details.

F.3 Factorization Sharing to Paillier Sharing Protocol

A protocol $(\{c_i\}_{i\in[n]}; x_i) \leftarrow \mathsf{F2P}(N, \mathcal{G}, 1^\kappa, 1^\sigma; P_i, Q_i)$ is an algorithm which receives a bi-prime and shares on it's non-trivial factors and returns $\{c_i\}_{i\in[n]} \in \mathcal{G}$ and $x_i \in \mathbb{Z}$ such that:

1. $\mathcal{G}$ is a DDH group.
2. for every $i \in [n]$ and every $c_i \in \mathbf{c}_i$ we have that $c_i = \mathsf{Enc}_{pk}(x_i)$. Where pk is a public key for El-Gamal in the exponent encryption with a corresponding private key which is unkown to the participants.
3. $\{x_i\}_{i\in[n]}$ is a Shamir Secret Sharing over the integers for the value x.
4. $[x \equiv 1 \mod N], [x \equiv 0 \mod \phi(N)]$.

Such a protocol is presented in [HMR+19].

F.4 ZK Proof for DL-Encryption Relation

Lastly we need a ZK proof for the following language

$$L_{EQ}(\mathbf{g}, \mathbf{a}, c) = \{(x, r) : c = \mathsf{Enc}(x, r) \wedge \mathbf{a}^x = \mathbf{g}\}$$

Description of such proof can be found in [CKY09].

F.5 The Protocol

We split the protocol into two steps Setup_1 and Setup_2. The first setup is responsible to the generation of a conforming bi-prime and an additive sharing over its factors. The second one is responsible for the generation of the Shamir secret sharing over the integers of the Paillier secret key, and the corresponding verification keys.

Let us analyze the expected running time of Setup_1 algorithm. We assume that $\frac{P-1}{2}, \frac{Q-1}{2}$ behave as random odd integers. As such the chance of them being co-prime is $\Pi_{p\neq 2}(1 - \frac{1}{p^2}) = \frac{8}{\pi^2}$, and so the success probability in each iteration is $\frac{8}{\pi^2} \cdot \alpha > \frac{4}{5}$. This gives that the expected running time is $\frac{5}{4}(\mathsf{time}(\mathsf{BiPrimeGen}) + \mathsf{time}(\mathsf{GCD}))$. Typically we have that $\mathsf{time}(\mathsf{BiPrimeGen}) \gg \mathsf{time}(\mathsf{GCD})$. The security of the above protocol follows from UC-security of $\mathsf{BiPrimeGen}$ and GCD. The only non-trivial part is that the distribution of N is different, as we add a condition. However, since this condition is fulfilled with non-negligible probability, the

ALGORITHM F.1 (*Conforming bi-prime generation*)

Input: $(n, 1^\kappa, 1^\sigma)$.

Algorithm: Each party does the following:

1. Init $\mathsf{sid} \leftarrow 0$
2. Run $(N; P_i, Q_i) \leftarrow \mathsf{BiPrimeGen}(n, 1^\kappa, 1^\sigma, \mathsf{sid})$
3. Run $b \leftarrow \mathsf{GCD}(n, 1^\kappa, 1^\sigma, 2, N-1; Q_i - \delta_{i,1})$
4. If $b = 1$ output $(N; P_i, Q_i)$ else increment sid and return to (1).

ALGORITHM F.2 (*Paillier Key Generation Given a Bi-Prime*)

Input: $(n, N, 1^\kappa, 1^\sigma; P_i, Q_i)$.

Algorithm: Each party does the following:

1. Run $(\{c_i\}_{i\in[n]}; x_i) \leftarrow \mathsf{F2P}(n, N, 1^\kappa, 1^\sigma; P_i, Q_i)$
2. Sample $\mathbf{g}_i \leftarrow (\mathbb{Z}_N^*)^c$ and broadcast $\mathsf{com}(\mathbf{g}_i)$.
3. Broadcast $\mathbf{g}_i$ and verify that the commitment is valid. If so set $\mathbf{g} = \Pi_{i\in[n]}\mathbf{g}_i$. Then set $\mathbf{a}_i = \mathbf{g}^{x_i}$. In addition calculate a zk-proof $\pi_{EQ}(\mathbf{a}_i, \mathbf{g}, c_i; x_i, r_i)$
4. Verify the proofs of all other parties.

identity reduction suffices. That is, given an adversary for the factoring problem with respect to Setup_1, we may use it on $\mathsf{BiPrimeGen}$ as well. Conditioned on whether $\gcd(P-1, Q-1) = 2$, which happens with probability ≈ 0.8, the bi-prime challenges of Setup_1 and $\mathsf{BiPrimeGen}$ have the same distribution. We can bound the success probability when the condition doesn't hold by zero.

Notably, [CHI+21] provide a UC secure protocol $\mathsf{BiPrimeGen}$, which includes a UC-simulation for the GCD sub-protocol, but they do not prove that the GCD subroutine is UC-secure by itself. However, similar proof techniques applies. As for Setup_2 the running time is dominated by $\mathsf{time}(\mathsf{F2P})$. The security proof is straight forward given a secure protocol $\mathsf{F2P}$. We note that in particular, there exist a simulator for the protocol, that given N can generate all public keys as well as the secret key-shares for the adversary. This is referred as Setup_2 in 4.2.

G Omitted Proofs

G.1 Proof of Lemma 4.2

Proof. Let $h \in \mathcal{G}$ be a generator, and fix a prime factor $p > \beta$ of $\mathsf{ord}(\mathcal{G})$. Given an element $g \in \mathcal{G}$, we can write $g = h^i$ for $0 \le i < \mathsf{ord}(\mathcal{G})$. Clearly, p divides $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ if and only if $p|i$. Therefore, the probability that p divides $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is $\frac{1}{p} < \frac{1}{\beta}$.

Next, we denote the prime factors of $\mathsf{ord}(\mathcal{G})$ that are larger than β by $p_1, \ldots, p_k$. Since $\gcd(p_i, p_j) = 1$ for all $i \neq j$ and since $p_i|\mathsf{ord}(\mathcal{G})$ for all i, we may deduce that $(\prod_i p_i)|\mathsf{ord}(\mathcal{G})$, so $\prod_i p_i \le \mathsf{ord}(\mathcal{G})$. By the assumption, $p_i > \beta$ for all i, so $\beta^k \le \mathsf{ord}(\mathcal{G})$ and $k \le \frac{\log \mathsf{ord}(\mathcal{G})}{\log \beta}$.

Now we are ready to conclude the proof: For an element $g \in \mathcal{G}$, we have that $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is not β-smooth if and only if there exists a prime factor $p > \beta$ of

$\mathsf{ord}(\mathcal{G})$ that divides $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$. The probability that a fixed p divides $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is at most $\frac{1}{\beta}$, and there are at most $\frac{\log \mathsf{ord}(\mathcal{G})}{\log \beta}$ such primes. Thus, by the union bound, the probability that $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is not β-smooth is at most $\frac{\log \mathsf{ord}(\mathcal{G})}{\beta \log \beta}$. ■

G.2 Proof of Corollary 4.3

Proof. By Lemma 4.2, it follows that $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is β-smooth with probability $1 - \frac{\log \mathsf{ord}(\mathcal{G})}{\beta \log \beta}$. We note that $\frac{|\mathcal{G}|}{\mathsf{ord}(g^{\Delta_n})} = \frac{|\mathcal{G}|}{\mathsf{ord}(g)/\gcd(\Delta_n, \mathsf{ord}(g))} = \frac{|\mathcal{G}| \cdot \gcd(\Delta_n, \mathsf{ord}(g))}{\mathsf{ord}(g)}$. In particular, since Δ_n is a β-smooth number (as all factors of Δ_n are smaller than n), so is $\gcd(\Delta_n, \mathsf{ord}(g))$ and the product of two β-smooth numbers is a β-smooth number. ■

G.3 Proof of Lemma 4.5

Proof. Since $\mathcal{G}$ is a cyclic group and $\prod_{i:\alpha_i \neq r_i} p_i^{r_i}$ divide $|\mathcal{G}|$, there exists b such that $\mathsf{ord}(b) = \prod_{i:\alpha_i \neq r_i} p_i^{r_i}$. The groups generated by g and b, namely, $\langle g \rangle$ and $\langle b \rangle$, are subgroups of $\langle g, b \rangle$, which implies that $|\langle g, b \rangle|$ is divisible by both $\mathsf{ord}(g)$ and $\mathsf{ord}(b)$. Thus, $|\langle g, b \rangle|$ is also divisible by $\mathsf{lcm}(\mathsf{ord}(g), \mathsf{ord}(b)) = |\mathcal{G}|$. ■

G.4 Proof of Corollary 4.6

In order to prove Corollary 4.6, we first present and prove a similar lemma over $\mathbb{Z}_N^*$:

Lemma G.1 *Let* Factor *be a factoring algorithm. There exists an algorithm* Factor' *that, given inputs* N, x, e*, such that*

- $N = PQ$ *is a conforming bi-prime;*
- $|e| < N$*;*
- $x \neq \pm 1 \mod N$*,* $e \neq 0$*, but* $x^e = 1 \mod N$*,*

outputs P, Q *and has a time complexity* $\mathsf{time}(\mathsf{Factor}'(N, x, e)) \leq \mathsf{time}(\mathsf{Factor}(e)) + \mathit{polylog}(N)$.

Note that the conditions on x *between this Lemma and Corrolary 4.6 are different.*

Proof. We can assume w.l.o.g that $e \geq 0$ by taking the inverse $(x^{-1})^e = (x^e)^{-1} = 1^{-1} = 1$ if necessary. Also, $e \neq 1$ since $x^e = 1$ and $x \neq 1$. If $e = 2$ then x is a non-trivial square root of 1, and we can compute $\gcd(N, x \pm 1)$ to get the factors of N in time $\mathsf{polylog}(N)$. Therefore, in the following we assume $e > 2$ and $[x^2 \neq 1 \mod N]$. Since $|\mathsf{QR}_N| = \frac{(P-1)(Q-1)}{4}$ is odd (and so 2 is co-prime to $|\mathsf{QR}_N|$), the function $f : \mathsf{QR}_N \to \mathsf{QR}_N$ defined by $f(a) = a^2$ is a bijection. Now, if $4|e$ then $x^e = (x^{e/2})^2$ and $x^{e/2} \in \mathsf{QR}_N$ (with $x^{e/4}$ being a square root), so $x^{e/2} = f^{-1}(1) = 1$. Thus, given (N, x, e), define $e_0 = e$ and $e_i = e_{i-1}/2$; we

can solve for (N, x, e') where $e' = e_i$ and i is the minimal index for which $4 \nmid e_i$. Define $s, t \in \mathbb{Z}$ such that $e' = s \cdot 2^t$, where $t \geq 0$ and s is odd (there must exist such s and t). The above analysis implies that $t \leq 1$, so either $e' = s$ (in case $t = 0$) or $e' = 2s$ (in case $t = 1$). Consider algorithm $\mathsf{Factor}'(N, x, e')$ below.

1. Call $\mathsf{Factor}(e')$ to obtain $p_1, \cdots, p_k$ such that $e' = \prod_{i=1}^{k} p_i$ and $p_1 \leq \cdots \leq p_k$.
2. For $i = 1$ to k:
   - If $x^{p_i} = 1$, return $P = \gcd(N, x - 1)$ and $Q = N/P$.
   - Otherwise, update $x \leftarrow [x^{p_i} \mod N]$.

The call to $\mathsf{Factor}(e')$ results with $p_1 \leq \cdots \leq p_k$ where $p_i > 2$ for all $i > 1$ (since $4 \nmid e_i$). Define $e'_0 = e'$ and $e'_{i+1} = \frac{e'_i}{p_{i+1}}$ (alternatively, $e'_i = \frac{e'}{\prod_{j=1}^{i} p_i}$) and similarly $x_0 = x$ and $x_{i+1} = (x_i)^{p_{i+1}}$ (alternatively, $x_i = x^{(\prod_{j=1}^{i} p_i)}$); note that $(x_i)^{e_i} = x_k = 1 \mod N$. Let i be the minimal index such that $x_i = 1$, and set $y = x_{i-1}$. We get that $y \neq 1 \mod N$ and $y^{p_i} = 1 \mod N$; since p_i is prime we have $p_i = \mathsf{ord}(y)$. If $p_i = 2$ then $i = 1$ and we have a non-trivial square root of 1, so we can factor N as above. Therefore, continue by assuming $p_i > 2$. Since N is a conforming bi-prime, we have $\gcd(P - 1, Q - 1) = 2$. In addition, $\mathsf{ord}(y) = p_i | (P - 1)(Q - 1)$ and $p_i > 2$; therefore, either $p_i | (P - 1)$ or $p_i | (Q - 1)$. Wlog assume $p_i | (Q - 1)$ (implying $p_i \nmid (P - 1)$ as otherwise $p_i > 2$ is a common divisor for $(P - 1)$ and $(Q - 1)$), so $\gcd(p_i, P - 1) = 1$ and the function $g : \mathbb{Z}_P \to \mathbb{Z}_P$ defined by $g(a) \to a^{p_i}$ is a bijection. Since $y^{p_i} = 1 \mod N$ we get $y^{p_i} = 1 \mod P$ and so $g(y) = g(1) = 1 \mod P$, implying $y = 1 \mod P$. On the other hand, $y \neq 1 \mod Q$ as otherwise (combining $y = 1 \mod P$ and $y = 1 \mod Q$) we have $y = 1 \mod N$, in contradiction to the fact that i is the minimal index for which $x_i = y^{p_i} = x_{i-1}^{p_i} = 1 \mod N$ (i.e., if $y = 1 \mod N$ then the condition already holds at $i - 1$). We summarize that $\gcd(y - 1, N) = P$.

To conclude, the aforementioned algorithm factors N. Since it begins with factoring e and the rest of the algorithm has a time complexity of $\mathsf{polylog}(N)$, the total time complexity of Factor' is at most $\mathsf{time}(\mathsf{Factor}(e)) + \mathsf{polylog}(N)$. ■

Now we are ready to prove Corollary 4.6:

Proof. Again note that $e < N$. $x^e = 1 \mod N^2$ implies $x^e = 1 \mod N$, therefore, if $x \neq \pm 1 \mod N$ then we apply algorithm Factor' from the proof of Lemma G.1.

We argue that $x \neq -1 \mod N$: we have the Legendre symbol $\left(\frac{-1}{P}\right) = [(-1)^{(P-1)/2} \mod P] = -1$ since $(P - 1)/2$ is odd, so $(-1) \neq \mathsf{QR}_P$. An element $g \in \mathbb{Z}_{N^2}$ belongs to QR_{N^2} if and only if $g \in \mathsf{QR}_P$ and $g \in \mathsf{QR}_Q$. Therefore, we have $(-1) \notin \mathsf{QR}_{N^2}$ and $x \neq -1 \mod N^2$.

It remains to address the case where $x \neq 1 \mod N^2$ and $x = 1 \mod N$. This implies that $x = 1 + kN$ for some $0 < k < N$, so $x^e = (1 + kN)^e = 1 + kNe = 1 \mod N^2$. It means that $(1 + kNe) - 1 = kNe = k'N^2$ for some $k' \in \mathbb{Z}$, so $ke = k'N$ and thus $PQ | ke$. Since $0 < k, e < N$, we may deduce that $N \nmid k$ and $N \nmid e$, so WLOG we may assume that $P | e$ and $Q \nmid e$. In that case, a factoring algorithm could find $P = \gcd(N, e)$. ■

G.5 Proof of Lemma 4.7

Proof. The algorithm Factor' is based on the Pollard's $p-1$ factorization method [Pol74]. It receives N, x, β as inputs and runs as follows:

1. For all $p_i \in \mathsf{primes} \cap [0, \beta]$:
   (a) Repeat $\lceil \log_{p_i}(N^2) \rceil$ times:
      i. If $x^{p_i} = 1 \mod N^2$ and $x = 1 \mod N$, return $P = p_i$ and $Q = N/P$.
      ii. If $x^{p_i} = 1 \mod N^2$ and $x \neq 1 \mod N$, return $P = \gcd(x-1, N)$ and $Q = N/P$.
      iii. Otherwise, assign $x \leftarrow [x^{p_i} \mod (N^2)]$ and continue.
2. Return $\perp$.

The algorithm certainly reaches x for which $x \neq 1 \mod N^2$ and $x^{p_i} = 1 \mod N^2$. Since $p_i < \beta < N$, Corollary 4.6 implies that, wlog, either $x = 1 \mod N$ and $\gcd(p_i, N) = P$, or $x \neq 1 \mod N$ and $\gcd(x-1, N) = P$. ■

H Batch Verification in Detail

In Algorithm H.1 below we present the batched verification algorithm, which takes as input a B-batched proof from C provers.

ALGORITHM H.1 (*C-Batched Verification of B-Batched Proofs*)

Input: The transcripts $\left(\tilde{\mathbf{g}}_j, \mathbf{a}_j, (\tilde{h}_{i,j}, \tilde{b}_{i,j})_{i\in[B]}, \mathbf{u}_j, v_j, z_j\right)_{j\in[C]}$. [a]

Algorithm:

1. $\mathcal{V}$ proceeds as follows:
   (a) Computes $\tilde{h}_j := \prod_{i\in[B]} \tilde{h}_{i,j}^{t_{i,j}}$, $\tilde{b}_j = \prod_{i\in[B]} \tilde{b}_{i,j}^{t_{i,j}}$, as in Protocol 4.2, and $\{t_{i,j}\}_{j\in[B]}, e_{j\,j\in[B]}$ as in Protocol 4.2 after FS transform.
   (b) Samples $s_j \leftarrow [0, 2^\kappa)$ for each $j \in [C]$.
   (c) Computes using multi-exponentiation (all computation is modulo N^2):
   $\hat{\tilde{\mathbf{g}}} = \prod_{j\in[C]} \tilde{\mathbf{g}}_j^{z_j \cdot s_j}$, $\hat{\tilde{h}} = \prod_{j\in[C]} \tilde{h}_j^{z_j \cdot s_j}$, $\hat{\mathbf{u}} = \prod_{j\in[C]} \mathbf{u}_j^{s_j}$, $\hat{v} = \prod_{j\in[C]} v_j^{s_j}$, $\hat{a} = \prod_{j\in[C]} a_j^{e_j \cdot s_j}$, $\hat{\tilde{b}} = \prod_{j\in[C]} \tilde{b}_j^{e_j \cdot s_j}$, and $\hat{\mathbf{g}} = \hat{\tilde{\mathbf{g}}}^2$, $\hat{h} = \hat{\tilde{h}}^2$, $\hat{b} = \tilde{b}^2$.
   (d) Accepts if
   – $\tilde{h}_{i,j}, \tilde{b}_{i,j}, u_j, v_j \in (0, N^2) \setminus \{N\}$ for all i, j,
   – $z_j \in (-D(2^{2\kappa} + 2^\kappa), +D(2^{2\kappa} + 2^\kappa))$ for all j,
   – $\hat{\mathbf{u}} = \hat{\mathbf{g}} \cdot \hat{\mathbf{a}} \mod N^2$ and $\hat{v} = \hat{h} \cdot \hat{b} \mod N^2$.

[a] In our case, $\tilde{\mathbf{g}}_j \equiv \tilde{\mathbf{g}}, \tilde{h}_{i,j} \equiv \tilde{h}_i$ across all provers.

The complexity of the batch verification in Algorithm H.1 consists of two separate C-multi-exponentiations with large exponents for $\hat{\tilde{\mathbf{g}}}$ and $\hat{\tilde{h}}$, other four separate C-multi-exponentiations with small exponents for $\hat{\mathbf{u}}, \hat{v}, \hat{a}$ and $\hat{b}$, and

the additional cost of computing $\tilde{h}_j$ and $\tilde{b}_j$ per batch proof (which incurs B-multi-exponentiation with small exponents C times). We note that in our case we assume $\tilde{\mathbf{g}}_j = \tilde{\mathbf{g}}$, implying that we can compute $\hat{\tilde{\mathbf{g}}} = \tilde{\mathbf{g}}^{\sum_j z_j s_j}$ and save one multi-exponentiation. We also note that we can combine the six separate multi-exponentiations in Step (c) into a single one.

Following [Pip80], a C-multi-exponentiation with E-bits exponents can be done in time $\mathcal{O}(E \cdot \frac{C}{\log(C)})$. Therefore, the total overhead in our case is $\mathcal{O}(\frac{C}{\log(C)} \cdot \log D + \frac{BC}{\log(B)} \cdot \kappa)$. In the following we show that the batch verification algorithm accepts only if each individual transcript is accepting (except for a negligible probability in κ).

Assume there exist i, j such that $\log_{\tilde{\mathbf{g}}}(\mathbf{a}_j) \neq \log_{\tilde{h}_i}(\tilde{b}_{i,j})$. Then by Theorem 4.10, assuming factorization is hard, $\log_{\tilde{\mathbf{g}}}(\mathbf{a}_j) \neq \log_{\tilde{h}_j} \tilde{b}_j$. Therefore, $\tilde{h}_j^{x_j \cdot e_j} \neq \tilde{b}_j^{e_j}$. Analogously to the proof of Lemma 4.11,

$$\Pr_{(s_{j'})_{j' \in [C]}} \left[\prod_{j' \in [C]} (\hat{h}_{j'}^{x_{j'} \cdot e_{j'}})^{s_{j'}} = \prod_{j' \in [C]} (\hat{b}_{j'}^{e_{j'}})^{s_{j'}} \right] \leq 2^{-\kappa}$$

Otherwise, there exist $s'_j \neq s_j$ such that the above equation holds, and by rearranging we get $\left(\hat{h}_j^{x_j} / \hat{b}_j\right)^{e_j \Delta s_j} = 1$. Again, if $\left(\hat{h}_j^{x_j} / \hat{b}_j\right)^{e_j} = 1$ we use Corollary 4.6. Otherwise, we use Pollard's rho algorithm on $\left(\hat{h}_j^{x_j} / \hat{b}_j\right)^{e_j}$, which has a low order that divides $\Delta s_j < 2^{\kappa}$. In both cases we get a factorization of N. Therefore by union bound, all BC statements about EDL relations hold with probability $\geq 1 - BC \cdot 2^{-\kappa} = 1 - \mathsf{neg}(\kappa)$ for $B, C = \mathsf{poly}(\kappa)$, therefore the probability to cheat is negligible.

Remark H.1 (Possible Trade-off) *In case many provers want to prove multiple statements at the same time, we can further improve the verifier's computational overhead, so it performs only two large exponentiations instead of two C-multi-exponentiations, at the cost of an additional communication round. Namely, each prover $\mathcal{P}_j$ first broadcasts its statement $((\tilde{h}_{i,j})_{i \in [B]}, (\tilde{b}_{i,j})_{i \in [B]})$. Then, all provers derive the same randomizers t_i by hashing all statements, and compute the same combinations $\tilde{h}_j = \prod_{i \in [B]} \tilde{h}_{i,j}^{t_i}$ and $\tilde{b}_j = \prod_{i \in [B]} \tilde{b}_{i,j}^{t_i}$ for every j. Each party then completes its proof on its statement $(\tilde{h}_j, \tilde{b}_j)$, which results with $(\mathbf{u}_j, v_j, e_j, z_j)$. $\mathcal{V}$ may then compute $\tilde{h}_j, \tilde{b}_j$, sample $s_j \leftarrow [0, 2^{\kappa})$, set $z := \sum_{j \in [C]} z_j \cdot s_j$, and compute $\hat{\mathbf{u}} = \prod_{j \in [C]} u_j^{s_j}$, $\hat{v} = \prod_{j \in [C]} v_j^{s_j}$, $\hat{a} = \prod_{j \in [C]} a_j^{e_j \cdot s_j}$, $\hat{b} = \prod_{j \in [C]} b_j^{e_j \cdot s_j}$, and verify that $\hat{\mathbf{u}} = \mathbf{g}^z \cdot \hat{\mathbf{a}}$ and $\hat{v} = \hat{h}^z \cdot \hat{b}$ (as well as the other range checks). Note that here there are only two large exponentiations (with z as the large exponent). HVZK and soundness arguments are similar to those in the Fiat-Shamir transform and the batch verification.*

H.1 Multi-Exponentiation

Suppose we want to compute:

$$\mathsf{pt} = \prod_{j=0}^{t} \mathsf{ct}_j^{\sum_{i=0}^{m-1} e_{i,j} \cdot B^i}$$

This is the type of computation required to preform decryption from partial Paillier decryptions, which is the bottleneck of our scheme, where we write each Lagrange coefficient (multiplied by $\Delta_n = n!$) in base $B = 2^b$, and so the bit-length of each exponent is $m \cdot b$. A naive computation for pt would take $\mathcal{O}(t \cdot m \cdot b)$ multiplications. Consider the following algorithm:

1. For each $0 \leq j \leq t$, and each $0 \leq k \leq B$, compute $y_{j,k} = \mathsf{ct}_j^k$. This takes Bt multiplications.
2. For each $0 \leq i < m$, compute $z_i = \prod_{j=0}^{t} \mathsf{ct}_j^{e_{i,j}} = \prod_{j=0}^{t} y_{j,e_{i,j}}$. This takes $m \cdot t$ multiplications.
3. Compute:

$$\mathsf{pt} = \left(\left(\left(\left(\left(z_{m-1}\right)^B \cdot z_{m-2}\right)^B \cdot z_{m-3}\right)^B \dots\right)^B \cdot z_1\right)^B \cdot z_0$$

   This takes m multiplications and $m \cdot b$ squares.

Complexity Analysis. If for example, $B = 2^8$ and $m = 2^9$ so that the bit-length of each Lagrange coefficient is $m \cdot b = 2^{12} = 4096$. A naive computation with t decryption shares would therefore take $4096t$ multiplications and squares. The above approach would take $256t$ multiplications for the first step, $512t$ for the second, and 4096 squares for the last part plus 512 multiplications. Overall, an improvement of about $\approx 80\%$ for $t \gg 1$ parties. In general, we want $m \approx B$, so $2^b \cdot b$ is the bit-length of the Lagrange coefficients. This results in about $\times \frac{1}{b}$ speed-up.