Tiresias: Large Scale, Maliciously Secure Threshold Paillier

Offir Friedman, Avichai Marmor, Dolev Mutzari, Yehonatan C. Scaly, Yuval Spiizer, and Avishay Yanai

dWallet Labs,
research@dwalletlabs.com

Abstract. In the threshold version of Paillier's encryption scheme a set of parties hold a share of the secret decryption key. Whenever a ciphertext is to be decrypted, the parties sends 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 handful proposals for a maliciously secure scheme, one must choose between an efficient implementation that relies on non-standard assumptions or an infeasible one that relies on widely acceptable assumptions.

In this work, we present a new protocol that combines the benefits of both worlds. We depart from the efficient scheme, which was proven secure relying on non-standard assumptions, and for the first time, prove that it is secure under standard assumptions only. 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, our simple and efficient proof supports *batching*, and hence enables batched threshold Paillier decryption for the first time.

Until now, verifying that a decryption share is correct was the bottleneck of threshold Paillier schemes, and prevented its implementation in practice (unless one is willing to rely on a trusted dealer). Our new proof and batching techniques shift that 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 implemented our 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, introduced by Pascal Paillier [Pai99] in 1999, 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, black box in the underlying ring.

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 group 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_i possesses a secret decryption key share sk_i. 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_i broadcasts a "decryption share" $\mathsf{ct}_j = Dec_{sk_i}(\mathsf{ct})$. 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 actualy 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]. However, the utilization of the threshold Paillier scheme necessitates the implementation of a protocol for both public key generation and distribution of secret key shares. Previous works, such as [FPS01,GGN16], which made use of the threshold Paillier decryption feature, relied on a trusted dealer for key generation. However, this approach inadvertently reintroduced a security risk similar to the one originally intended to be avoided. Other works such as [GG20] compromised on non-threshold Paillier for threshold ECDSA signatures. Nevertheless, this requires each party to have its own Paillier public key, and therefore in the pre-sign phase each pair of parties exchanges unique messages, rather than having each party only send broadcast encrypted messages under a single public key. 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_i's signature share on message m is computed by $\sigma_i = H(m)^{d_i} \mod N$ where $d = \sum d_i$ is the secret signing key shared among the parties, and the final signature is $\sigma = \prod \sigma_i$. If one of the parties computes an incorrect partial signature $\sigma'_i \neq \sigma_i$ 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, all proposals for threshold Paillier devise such a verifiability mechanism, in the form of a zero knowledge 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 that relies on non-standard assumptions, and a protocol whose proof is inefficient, but relies on standard widely accepted assumptions. 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*. 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,DJN10], 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 that lowers the bar by generating public key N that is a product of 'general' primes[1] P, Q (i.e., they are not necessarily safe), and using the same efficient zero knowledge proof as [FPS01]. 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*.

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 safe primes – distributed generation of almost safe primes can be done, but is impractical for most use-cases (our estimations were that generation of almost safe primes would be 1000x slower than general ones), and unlike general primes – the B-roughness property facilitates a proof of soundness for the zero knowledge protocol without relying on new assumptions. This, however, incurs a degradation of the efficiency of the zero knowledge proof. As reported by the authors, the proof efficiency is about $30\times$ worse than that used in [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].

Lastly, 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[2].

[1] 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].

[2] 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

1.2 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[3] 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[4], such that party P_i obtains a share d_i. In addition to the public modulus N, the parties generate a public verification key v_i for every P_i, such that $v_i = g^{d_i}$ 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_i sends the decryption share $\mathsf{ct}_i = \mathsf{ct}^{d_i}$ along with a zero-knowledge proof that ct_i is computed correctly using the public ct and the secret d_i.[5] This proof is a proof of *equality of discrete logs* of the values ct_i and v_i over QR_{N^2}, with respect to the bases ct and g. That is, if P_i computes its decryption share correctly then $\log_{\mathsf{ct}}(\mathsf{ct}_i) = \log_g(v_i) = d_i$.

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, it is guaranteed to have a generator g, meaning that every element in QR_{N^2} equals g^i for some i. This fact typically helps when arguing security for zero knowledge protocols. Finding a generator of QR_{N^2} is easy when N is a product of safe primes, since a random element of QR_{N^2} is highly probable to generate the group. However, in the general case (where the primes are not safe), even when QR_{N^2} is cyclic, no efficient algorithm for finding a generator of QR_{N^2} is known.

---

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.

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

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

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

3. The standard proof of soundness (see [FPS01]) obtains an equation of the form $x^e = 1 \mod N^2$ for some small e (i.e., $e \ll \min\{P', Q'\}$), 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 rely 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.

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], 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.[6] 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-primes, and therefore QR_{N^2} is cyclic. 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-primes. The rejection of prime candidates that do not satisfy that condition incurs only a small constant factor overhead (about $1.2\times$) to the key generation protocol.
2. Even when P, Q are not safe primes, 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 the order of QR_{N^2} divided by a sufficiently small smooth number (whose prime factors are all small).

[6] We do require N to meet one extra constraint, which only slightly affects the efficiency of the key generation protocol, but not the efficiency of the threshold decryption, and doesn't require additional cryptographic assumptions.

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.

Interestingly, our proof of soundness leverages the fact that there is a large gap between a computational security parameter κ and the hardness of the factorization problem. In particular, it leverages the fact that factoring a number with κ bits is actually *easy* for popular choices of κ like 128 or 256.

1.3 Our Contribution

- We present the first practical, efficient, large-scale threshold Paillier encryption protocol, supporting efficient distributed key generation and threshold decryption, relying only on standard assumptions (the decisional composite residuosity assumption). 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 of soundness of the equality of discrete logs over QR_{N^2}, *even when N is not a product of safe primes*. Such proof may find 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 a future work.[7] 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, which is required to continuously process many ciphertexts, the parties need to verify decryption shares received from other parties for many ciphertexts. Verification of such many proofs (number of parties times number of ciphertexts) can easily dominate the overall cost of decryption. To solve this, we adapt a novel batching technique, which is applicable when verifying proofs of multiple decryption shares from multiple provers. Specifically, computation and verification of B proofs incurs single 'large' exponentiation and $\mathcal{O}(B)$ 'small' exponentiations instead of $\mathcal{O}(B)$ large exponentiations (where 'large' refers to the size of the shamir share over the integers of the decryption key, e.g., 4096 bits, and 'small' refers to the size of the computational security parameter, e.g., 128 bits).

[7] 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.

For example, in a setting of 1000 parties, computing decryption shares for 1000 ciphertexts over a standard Mac laptop takes only 144 milliseconds per share, while it takes 137 milliseconds per share if no proof is required (i.e., in the semi-honest setting). This means that protecting against a malicious security in this case incurs only $\times 1.05$ overhead.

– We stress that this is the first time that batch proof and verification for threshold Paillier decryption is shown possible, when N is not a product of safe-primes. As argued above, such batching technique is crucial for the efficiency of threshold Paillier decryption and is the difference between a practical and impractical solution.

Although our contributions are concentrated around threshold decryption, we briefly discuss the protocol for distributed key generation as well (in Section 3.3), for completeness of the exposition.

1.4 Organization

In Section 2 we present our notation and the mathematical background that is necessary in order to understand our protocols; then we present in detail the background on the Paillier encryption scheme, threshold decryption and distributed key generation in Section 3. In Section 4 we present our new zero knowledge protocol for equality of discrete logs over QR_{N^2}, and provide a full proof of security. Finally, we present an optimization to the plaintext reconstruction, as well as a tighter analysis Shamir sharing over the integers in Section 5, where we also report on our experiments.

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 $\mathsf{time}(A(x_1, x_2, \ldots))$ the run time of an algorithm A on inputs $(x_1, x_2, \ldots)$.

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 $r = 0$ then we say that b *divides* a and denote it by $b|a$. If, in addition, $b \notin \{1, a\}$, then we say that 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 2.1 *Let $a, b \in \mathbb{Z}$. if $c = [ab \mod b^2]$ then $a = c/b \mod b$.*

Proposition 2.2 *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 2.3 *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 2.4 *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.

Definition 2.5 (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 2.6 (Safe Prime) *A prime number p is called* safe *if $\frac{p-1}{2}$ is prime.*

Definition 2.7 (Conforming Bi-Prime) *An integer N is called a* conforming bi-prime *if there exist two primes P, Q such that $N = P \cdot Q$, $\gcd(P-1, Q-1) = 2$, $P = Q = 3 \mod 4$ and $\gcd(N, \phi(N)) = 1$.*

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 $\mathcal{H} \subseteq G$ then $|\mathcal{H}|$ divides $|\mathcal{G}|$.

Indistinguishability.

Definition 2.8 (Statistical Distance) *Let $X, Y : \Omega \to [M]$ be two random variables. The* statistical distance *between X, Y, denoted $\mathsf{SD}(\mathsf{X}, \mathsf{Y})$, is*

$$\mathsf{SD}(\mathsf{X}, \mathsf{Y}) := \sum_{\mathsf{w} \in \Omega} |\Pr[\mathsf{X} = \mathsf{w}] - \Pr[\mathsf{Y} = \mathsf{w}]|$$

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 2.9 (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*

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

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

The decisional composite residuosity (DCR) assumption is the assumption that there exists a modulus generator algorithm $\mathsf{GenModulus}$ relative to which the decisional composite residuosity problem is hard.

Definition 2.10 (Factoring) *We say that factoring is hard relative to* $\mathsf{GenModulus}$ *if for all probabilistic polynomial time algorithms* $\mathcal{A}$ *there exists a negligible function* neg *such that*

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

The factoring assumption is the assumption that there exists a $\mathsf{GenModulus}$ relative to which factoring is hard.

2.1 Shamir Secret Sharing

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

Shamir 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 $s = \mathsf{Reconstruct}([s]_{i_1}, \ldots, [s]_{i_{t+1}})$ 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 $[s]_i = p(i)$ and $p(x) = s + a_1 x + a_2 x^2 + \ldots + a_t x^t$.
- $s = \mathsf{Reconstruct}(\{(t, [s]_t)\}_{j \in T})$. Let $T \subset [n]$ be a set of $t+1$ distinct elements from $[n]$. Let p be the unique interpolation polynomial such that $p(t) = [s]_t$ for all $j \in T$. Output $s = p(0)$.

Lagrange interpolation is used in order to get $p(i)$ directly. For $T = \{i_1, \ldots, i_{t+1}\}$, given points $([s]_{i_1}, \ldots, [s]_{i_{t+1}})$, the polynomial that passes through them is $p(x) = \sum_{j=1}^{t+1} [s]_{i_j} \cdot \ell_j(x)$, where

$$\ell_j(x) = \prod_{\substack{1 \le k \le t+1 \\ k \ne j}} \frac{x - i_k}{i_j - i_k}.$$

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 $\lambda_{T,j}^v = \ell_j(v)$; then we can write $p(v) = \sum_{j=1}^{t+1} \lambda_{T,j}^v \cdot [s]_{i_j}$.

Shamir sharing over the integers. Let $s \in \mathbb{Z}$ be a secret such that $s \in [-b, +b]$. Define $\Delta_n = n!$ and define some bound $I(\sigma, n, b)$ on the (absolute value of the) coefficients of the polynomial. Until recently (e.g., see [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 Section 5.2.

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 $[s]_i = p(i)$ and $p(x) = \Delta_n \cdot s + \sum_{j=1}^{t} a_j \cdot x^j$.

Reconstruction works as follows. Given a set $T = \{i_1, \ldots, i_{t+1}\}$ and shares $\{[s]_{i_j}\}_{i_j \in T}$, we have

$$p(0) = \sum_{j=1}^{t+1} \lambda_{T,j}^0 \cdot [s]_{i_j} = \Delta_n s,$$

However, since the Lagrange coefficients might not be in $\mathbb{Z}$, we multiply them first by Δ_n, and get

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

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:

$$D(\sigma, n, t, b) = \Delta_n \cdot b + \sum_{i=1}^{t} I(\sigma, n, b) \cdot n^i \le \Delta_n \cdot b + 2n^t I(\sigma, n, b)$$

for all $n \ge 2$.

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_i compute $[s]_i = [s_1]_i + [s_2]_i$ 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_i compute $[\alpha s]_i = \alpha [s]_i$. Similarly, the above is extended to any affine operation over the secrets.

2.2 Zero Knowledge

We use the formalization from [HL10, Chapter 6] with a slight deviation in the soundness definition. 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 $(x,w) \in R$. A Σ-protocol template for relation R is given in Protocol 2.1 below. We say that the transcript (a,e,z) is an *accepting transcript* for x if the protocol instructs $\mathcal{V}$ to accept based on the values (x,a,e,z).

PROTOCOL 2.1 (*Σ-protocol template for relation R*)

- **Common input:** The prover $\mathcal{P}$ and verifier $\mathcal{V}$ both have x.
- **Private input:** $\mathcal{P}$ has a value w such that $(x,w) \in R$.
- **Protocol template:**
  1. $\mathcal{P}$ sends $\mathcal{V}$ a message a.
  2. $\mathcal{V}$ sends $\mathcal{P}$ a random challenge $e \leftarrow \{0,1\}^\kappa$.
  3. $\mathcal{P}$ sends a reply z, and $\mathcal{V}$ decides to accept or reject based solely on (x,a,e,z).

Definition 2.11 *A protocol π is a Σ-protocol for relation R if it is an instance of the protocol template above (Protocol 2.1), namely, it is a three-round public-coin protocol, and the following requirements hold:*

- **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 statefull probabilistic polynomial 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)*

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

  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 $\mathcal{S}$, which on input x outputs a transcript of the form (a,e,z) with the same probability distribution as transcripts between $\mathcal{P}$ and $\mathcal{V}$ on common input x. Formally, for every x and w such that $(x,w) \in R$, it holds that $\mathcal{S}(x)$ and $(\mathcal{P}(x;w) \leftrightarrow \mathcal{V}(x))$ are statistically indistinguishable.*

Proposition 2.12 (Proposition 6.2.3 [HL10]) *Let π be a Σ-protocol for a relation R. Then, π is an interactive zero-knowledge proof of membership for L_R.*

It is common to turn a Σ-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 honest verifier.

3 Background on Threshold Paillier

3.1 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 2.1; and the fact that $d = 1 \mod N$ and that $\mathsf{pt} < N$.

3.2 Threshold Decryption with a Trusted Setup

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 the integers 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 $j \in T$ where $T \subset \{P_1, \ldots, P_n\}$ of size $t+1$, it is possible to 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, by Eq. (1) we have $\left(\sum_{j\in T} \Delta_n d_j \lambda^0_{T,j}\right) = \Delta_n^2 d$), and the last one follows by the encryption definition.

Then, obtain the plaintext by computing

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

where the first equality is derived from the correctness of the standard 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 zero knowledge 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., which uses a single verification key. In more detail, in addition to the sharing $[d]$, the trusted dealer computes and publishes $v_j = g^{\Delta_n d_j}$ 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, we view v_j as a commitment to P_j's share of the secret key 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.

3.3 Distributed Key Generation (Without a Trusted Dealer)

Many protocols for distributed generation of bi-prime (or RSA) modulus were proposed (e.g. [BF97,DdSGMRT21,BDF+23]). In this work we make use of

the Diogenes [CHI+21] protocol as it is highly scalable. The functionality realized by the Diogenes protocol is given in Functionality A.1. 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 2.7). 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 $\gcd(P_i - 1, Q_i - 1) = \gcd(P_i - 1 + (Q_i - 1)P_i, Q_i - 1) = \gcd(N_i - 1, Q_i - 1)$. And so we run the GCD test on the secret value $Q_i - 1$ and the public value $N_i - 1$.

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 a random basis $g \in \mathsf{QR}_{N^2}$ and finally each party publishes $v_j = g^{d_j}$ and proves that c_j is an encryption of $\log_g(v_j)$. 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).

The security of the key generation is proven in [CHI+21,HMR+19] to be UC-secure [Can01] under the decisional composite residuosity, RLWE and the decisional Diffie-Hellman assumptions.

4 Zero-Knowledge of Equality of Discrete Logs

4.1 Formalizing the Language

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. 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 would be:

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

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

1. g is a base element chosen in the DKG phase.
2. a is the verification key v_j associated with the prover P_j. namely, $v_j = g^{2\Delta_n d_j}$.
3. x is a witness known by the prover P_j, namely, $x = 2\Delta_n d_j$.
4. h is the ciphertext to be decrypted ct.
5. b is the claimed partial decryption of the prover P_j, namely, $b = \mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$.

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 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. Since g might not be a generator of QR_{N^2}, we have $\mathsf{ord}(g) < |\mathsf{QR}_{N^2}|$ and so there exists $x' \in \mathbb{Z}$ such that $x \neq x' \mod |\mathsf{QR}_{N^2}|$ and $g^{x_1} = g^{x_2} = a$. Indeed, we may take any $x' = x \mod \mathsf{ord}(g)$. Therefore, a cheating prover may send (h, b) such that $b = h^{x'} \neq h^x$, yet, this prover can wrongly convince the verifier that $\log_g a = \log_h b \mod |\mathsf{QR}_{N^2}|$. Nevertheless, we show that such a pair (h, b) is hard to find, and in particular can be reduced to factoring N.

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}. Specifically, a random element $g' \leftarrow \mathbb{Z}^*_{N^2}$ is sampled and published in the DKG phase, and we set $\tilde{g} = g'^{\Delta_n}$, and $g = \tilde{g}^2 = g'^{2\Delta_n} \in \mathsf{QR}_{N^2}$. 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. g' is *some* element *chosen from* $\mathbb{Z}^*_{N^2}$ in the DKG phase.
2. $a = g^{d_j}$ is the verification key v_j associated with the prover P_j, which is generated in the DKG phase. Namely, $v_j = g'^{2\Delta_n d_j}$.

3. x is a witness known by the prover P_j, namely, $x = d_j$.
4. $\tilde{h} = \mathsf{ct}^{2\Delta_n}$.
5. $\tilde{b}$ is the claimed partial decryption of the prover P_j, namely, $\tilde{b} = \mathsf{ct}_j = \mathsf{ct}^{2\Delta_n d_j}$.

We get that $\log_h b = \log_g a = d_j$ and the new formalization of the language is

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

The witness is now $x = d_j$, which is $\log(\Delta_n)$ bits shorter than the witness as defined in $L_{\mathsf{EDL}'}$.

Regarding the second issue, the language is formalized with $\tilde{g}, \tilde{h}$ rather than g, h. In addition, x is a witness that is disclosed to the prover and we require the prover to use that particular x in the proof. Specifically, $x = d_j$ is P_j's share over the integers of the decryption key d. We expressed this fact by adding x to the parameters of the language L_{EDL^2}. Then, by defining

$$L_{\mathsf{EDL}}[N, g, a; x] = \{(h, b) \mid h \in \mathsf{QR}_{N^2} \wedge a = g^x \wedge b = h^x\},$$

we have $(\tilde{h}, \tilde{b}) \in L_{\mathsf{EDL}^2}[N, \tilde{g}, a; x] \iff (h, b) \in L_{\mathsf{EDL}}[N, g, a; x]$, where $b = \tilde{b}^2$, therefore, we continue with a proof system for the language L_{EDL^2}.

Having established the language and the syntax above, let us describe the proof in the context of threshold decryption of Paillier.

1. The setup phase consists of the DKG and the choice of the ciphertext to decrypt. That is, the results of the setup phase are the bi-prime modulus N, the base point $\tilde{g} \in \mathbb{Z}^*_{N^2}$, $a = v_j$ is the verification key associated with P_j, $\tilde{h} = \mathsf{ct}$ is the ciphertext that the parties agree to decrypt, and finally, $x = d_j$ is P_j's share over the integers of d - the secret decryption key.
2. The prover P_j sends $\tilde{b}$ and proves that $(\tilde{h}, \tilde{b}) \in L_{\mathsf{EDL}^2}$.

In sub-section 4.2 we describe the setup phase in more detail; we present and prove a Σ-protocol for L_{EDL^2} in sub-sections 4.3; in sub-sections 4.4-4.5 we show how batching of the protocol is possible over multiple ciphertexts; and finally we show security of the protocol under the Fiat-Shamir transrorm in sub-section 4.6.

4.2 Setup Phase

Let us define the Setup algorithm for generating the parameters of the language:

$$(N, \tilde{g}, a; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$$

This algorithm takes 1^κ and 1^σ as inputs and outputs:

– A conforming bi-prime $N = P \cdot Q$.

– g' is drawn uniformly at random from $\mathbb{Z}^*_{N^2}$ and computes $\tilde{g} = g'^{\Delta_n}$ and $g = \tilde{g}^2$. We stress that g is not necessarily a generator of QR_{N^2}. Note that g'^2 is a random element in QR_{N^2}. Setting $g = (g'^2)^{\Delta_n}$ and $\beta = 2^{\sigma+2}\log(|\mathsf{QR}_{N^2}|)$, Lemma 4.6 and Corollary 4.7 imply that $\frac{|\mathsf{QR}_{N^2}|}{\mathsf{ord}(g)}$ is β-smooth with probability $> 1 - 2^{-\sigma-1}$. We define $\beta_\sigma = 2^{\sigma+2}\log|\mathsf{QR}_{N^2}|$ and observe that when $n < \beta_\sigma$ it holds that $\beta_\sigma \leq 2^{\sigma+3}\log N$.
– x is drawn from $[-D, +D]$; and
– $a := g^x$.

The algorithm may fail with probability $2^{-\sigma}$. Specifically, failure means that either N is not a bi-prime or $\frac{|\mathsf{QR}_{N^2}|}{\mathsf{ord}(g)}$ is not β_σ-smooth. The former happens when N passes the bi-primality Jacobi test $\sigma+1$ times even though it is not a bi-prime. Such N passes the Jacobi test with probability at most $1/2$, so after $\sigma+1$ independent tests it fails with probability $2^{-(\sigma+1)}$. The latter happens with probability $2^{-(\sigma+1)}$ by Lemma 4.6. By the union bound, Setup fails with with probability at most $2^{-\sigma}$.

4.3 The Protocol

In the following we give some explanation about the values in the proof, in order to put it in the right context of threshold Paillier decryption. The global parameters consist of $(N, \tilde{g}, a; x)$ output from the setup phase, where $g = \tilde{g}^2 \in \mathsf{QR}_{N^2}$ is a base element for the verification keys of all parties, $a = g^x$ is the verification key associated with the specific prover, where $x = d_j$ is the decryption key share held by the prover. Then, $\tilde{h} = \mathsf{ct}^{2\Delta_n}$ where ct is the ciphertext to be decrypted, and $\tilde{b} = \tilde{h}^{d_j} = \mathsf{ct}^{2\Delta_n d_j}$.

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

Parameters. The output of $(N, \tilde{g}, a; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$.

Protocol.

1. $\mathcal{P}$ samples $r \leftarrow [-2^{2\kappa}D, +2^{2\kappa}D)$ and chooses $\tilde{h} \in \mathbb{Z}^*_{N^2}$, and sends $\tilde{h}, \tilde{b} = [\tilde{h}^x \mod N^2], u = [g^r \mod N^2], v = [h^r \mod N^2]$ 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. $\mathcal{V}$ computes $b = [\tilde{b}^2 \mod N^2]$, $h = [\tilde{h}^2 \mod N^2]$ and verifies that:
   – $\tilde{h}, \tilde{b}, u, v \in (0, N^2) \setminus \{N\}$,
   – $z \in (-D(2^{2\kappa} + 2^\kappa), +D(2^{2\kappa} + 2^\kappa))$,
   – $u = g^z \cdot a^e \mod N^2$ and $v = h^z \cdot b^e \mod N^2$.

We prove the following.

Theorem 4.2. *A protocol Π_{EDL^2} (Protocol 4.1) is a Σ-protocol for relation EDL^2.*

Proof. In the following we show that all Σ-protocol's properties are satisfied.

Completeness.

Let $(N, \tilde{g}, a; x)$ 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 $\tilde{h}, \tilde{b}, u$ 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

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

and

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

Honest-verifier zero-knowledge (HVZK). We show that for every $(\tilde{h}, \tilde{b}) \in L_{\mathsf{EDL}^2}[N, g, a; x]$ there exists a PPT simulator $\mathcal{S}$, such that

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

The simulator $\mathcal{S}$ samples $z' \leftarrow (-D(2^{2\kappa} + 2^\kappa), +D(2^{2\kappa} + 2^\kappa))$, $e' \leftarrow [0, 2^\kappa)$ and computes $u' = [g^{z'} \cdot a^{e'} \mod N^2]$ and $v' = [h^{z'} \cdot b^{e'} \mod N^2]$. We argue that for every the statistical distance between (u, v, e, z) (of the real execution) and (u', v', e', z') (of the simulation) is negligible in κ. First, we have that e and e' are identically distributed. Then, (u, v) and (u', v') are fully determined by $(N, g, a, \tilde{h}, \tilde{b}, e, z)$ and $(N, g, a, \tilde{h}, \tilde{b}, e', z')$, respectively. Therefore, to complete the proof it is sufficient to show that for every $\tilde{e} \in [0, 2^\kappa)$ showing that the statistical distance between $[z|e = \tilde{e}]$ and $[z'|e' = \tilde{e}]$ is negligible in κ.

We have that z' (in simulation) is uniformly distributed from $(-D(2^{2\kappa} + 2^\kappa), +D(2^{2\kappa} + 2^\kappa))$ independent of e', whereas z (in real execution) is computed by $z = r - ex$, 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. Specifically, fix x and e and consider the probability $\Pr[z = \zeta]$:

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

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

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

where ζ iterates over $(-D(2^{2\kappa} + 2^\kappa), +D(2^{2\kappa} + 2^\kappa))$, which concludes the proof.

Soundness. Notably, since g is not necessarily a generator of QR_{N^2}, we cannot prove soundness for every statement $(\tilde{g}, a, \tilde{h}, \tilde{b})$. Indeed, for any $y \in \mathbb{Z}$ such that $y = x \mod \mathsf{ord}(g)$ and for any $\tilde{h} \in [N^2]$, setting $\tilde{b} := \tilde{h}^y$ yields an accepting transcript. This does not constitute a break to the soundness property (see Definition 2.11) since finding such group elements $\tilde{h}, \tilde{b}$, implies (as we see shortly) finding a group element $\tilde{b}/\tilde{h}^x$ with "small" order, which translates to factoring N. Therefore, in the definition let the $\mathcal{P}^*$ choose the statement $(\tilde{h}, \tilde{b})$, and show a reduction from cheating prover to factoring of N. Formally, we prove

Theorem 4.1 *Let $\mathcal{P}^* = (\mathcal{P}_1^*, \mathcal{P}_2^*)$ be a stateful PPT prover such that*

$$\Pr[(\mathcal{P}_2^*() \leftrightarrow \mathcal{V}(\tilde{h}, \tilde{b})) = 1 \mid (h, b) \notin L_{\mathsf{EDL}}[N, g, a; x], \\ (\tilde{h}, \tilde{b}) \leftarrow \mathcal{P}_1^*(1^\kappa, 1^{2^\sigma}, N, \tilde{g}, a; x), \\ (N, \tilde{g}, a; x) \leftarrow \mathit{Setup}(1^\kappa, 1^\sigma)] = \varepsilon.$$

Then there exists an adversary $\mathcal{A}$ that solves the factorization problem (Definition 2.10) with probability $1 - 2^{-\kappa}$ in time $\mathcal{O}(\frac{\kappa}{\varepsilon'} \cdot (T + \log(D)\log^2(N)) + M) + \beta_\sigma \log^3(N))$, where $M = \max\{\mathsf{time}(\mathsf{Factor}(m)) \mid m \in [2^\kappa]\}$, $\varepsilon' = \varepsilon^2 - \varepsilon \cdot 2^{-\kappa}$ and $T = \mathsf{time}(\mathcal{P}^*(1^\kappa, 1^{2^\sigma}, N, \tilde{g}, a; x))$.

Intuitively, we show that finding a pair $(h, b) \notin L_{\mathsf{EDL}}[N, g, a; x]$ that leads to an accepting transcript with a non-negligible probability, yields an element $\frac{b}{h^x}$ of "small" order, which translates to factorization. Therefore, finding such h, b given N should be computationally as hard as factoring, and in turn it is infeasible to state a false claim and prove it assuming factorization is hard.

Proof. First, we introduce an algorithm called $\mathcal{A}_1$ with a time complexity of $\mathcal{O}(T + \log(D)\log^2(N))$. The purpose of algorithm $\mathcal{A}_1$ is to find an element $c \in \mathsf{QR}_{N^2}$ and an exponent $0 < e' < 2^\kappa$, such that $c \neq 1$ and $\mathsf{ord}(c^{e'})$ is a β_σ-smooth number. We show that $\mathcal{A}_1$ does so with probability at least ε' Then, using $\mathcal{A}_1$, algorithm $\mathcal{A}$ factors N in time $\mathcal{O}(\frac{\kappa}{\epsilon'} \cdot (T + \log(D)\log^2(N)) + M)$. Algorithm $\mathcal{A}_1$ works as follows:

1. **Setup**: Let $(N, \tilde{g}, a; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$. By Lemma 4.6 and Corollary 4.7 we have that $\frac{|\mathsf{QR}_{N^2}|}{\mathsf{ord}(g)}$ is β_σ-smooth with probability at least $1 - 2^{-\sigma-1}$ and by Lemma 4.2 there exists an element $\eta \in \mathsf{QR}_{N^2}$ such that $\langle g, \eta \rangle = \mathsf{QR}_{N^2}$, and for every prime p such that $p | \mathsf{ord}(\eta)$ we have that $p | \frac{|\mathsf{QR}_{N^2}|}{\mathsf{ord}(g)}$, which implies $\mathsf{ord}(\eta)$ is β_σ-smooth. Note that η might not be known to $\mathcal{A}_1$.
2. **Receiving the claim**: $\mathcal{A}_1$ calls $\mathcal{P}_1^*(1^\kappa, 1^{2^\sigma}, N, \tilde{g}, a; x)$ and receives $\tilde{h}$ and $\tilde{b}$. If $\tilde{b}^2 = \tilde{h}^{2x}$ then output $\perp$ (failure, as we are looking for a prover that breaks soundness).
3. **Receiving proofs**: $\mathcal{A}_1$ calls $\mathcal{P}_2^*()$ and receives a randomized claim u, v. $\mathcal{A}_1$ then sends a random challenge $e_1 \in [0, 2^\kappa)$ and receives the response z_1. $\mathcal{A}_1$ rewinds $\mathcal{P}_2^*()$ and calls it again with a fresh challenge $e_2 \leftarrow [0, 2^\kappa)$ and receives another response z_2. If both proofs are accepting and $e_1 \neq e_2$, then $\mathcal{A}_1$ outputs $c = \frac{b}{h^x}$ and $e' = |e_1 - e_2|$, otherwise it outputs $\perp$.

By the rewinding lemma (see [BS20], Lemma 19.2), the probability that $\mathcal{A}_1$ returns an output other than $\perp$ is at least $\varepsilon' = \varepsilon^2 - \varepsilon 2^{-\kappa}$. Moreover, its time complexity is $\mathcal{O}(T + \log(D)\log^2(N))$, since the most expensive part of $\mathcal{A}_1$ except for calling $(\mathcal{P}_1^*, \mathcal{P}_2^*)$ is an exponentiation by x modulo N^2. Let us prove that whenever $\mathcal{A}_1$ does not return $\perp$, its output is 'valid'. Namely, $\mathsf{ord}(c^{e'})$ is a β_σ-smooth number, where $c \in \mathsf{QR}_{N^2}, e' \in (0, 2^\kappa)$, and $c \neq 1$.

We have four equations (all are computed modulo N^2):

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

Assume wlog that $e_1 > e_2$ and denote $e' = e_1 - e_2, z' = z_1 - z_2$, then

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

Since there exists η such that $\langle g, \eta \rangle = \mathsf{QR}_{N^2}$, there exists α, δ such that $h = g^\alpha \eta^\delta$ and

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

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

$$1 = \frac{g^{\alpha \cdot z'} \eta^{\delta \cdot z'} \cdot b^{e'}}{g^{\alpha \cdot z'} \cdot a^{\alpha \cdot e'}} = \eta^{\delta z'} \left(\frac{b}{a^\alpha}\right)^{e'} \mod N^2. \tag{3}$$

Observe that

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

then, substituting $\frac{b}{a^\alpha} = \frac{b}{h^x} \eta^{\delta x}$ in Eq. (3),

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

Therefore, raising to the power of $\mathsf{ord}(\eta)$:

$$1 = \left(\eta^{\delta(z' + x e')} \left(\frac{b}{h^x}\right)^{e'} \mod N^2\right)^{\mathsf{ord}(\eta)} = \left(\frac{b}{h^x}\right)^{\mathsf{ord}(\eta) e'} \mod N^2.$$

Algorithm $\mathcal{A}_1$ outputs $c = [\frac{b}{h^x} \mod N^2] \neq 1$ (since if $b = h^x$ the algorithm fails), and $0 < e' < 2^\kappa$, where so it's order divides $\mathsf{ord}(c^{e'}) | \mathsf{ord}(\eta)$, so it is β_σ-smooth.

Now we describe the required algorithm $\mathcal{A}$: It calls $\mathcal{A}_1$ up to $1 + \frac{3ln(2)}{2} \frac{\kappa}{\varepsilon'}$ times, until it obtains c and e' as above. If all invocations yield $\perp$ then $\mathcal{A}$ outputs $\perp$. By Chernoff's bound, $\mathcal{A}$ outputs $\perp$ with probability $\leq 2^{-\kappa}$. Otherwise (c and e' were obtained), if $c^{e'} = 1 \mod N^2$ then by Lemma 4.3 $\mathcal{A}$ may find the factors P, Q of N in time complexity $\mathsf{time}(\mathsf{Factor}(e')) + \mathsf{polylog}(N)$. Otherwise, $\left(c^{e'}\right)^{\mathsf{ord}(\eta)} = 1$ and by Lemma 4.5 $\mathcal{A}$ may find P, Q in time $\beta_\sigma \log^3(N)$. ■

In the following we present and prove the lemmas used in the reduction above.

Lemma 4.2 *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 $i \in [k]$. For every $a \in \mathcal{G}$, there exists $b \in \mathcal{G}$ such that $\langle a, b\rangle = \mathcal{G}$ and $\mathsf{ord}(a) = \prod_{i=1}^{k} p_i^{\alpha_i}$ where $\alpha_i \leq r_i$ for all i and $\mathsf{ord}(b) = \prod_{i:\alpha_i \neq r_i} p_i^{r_i}$.*

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 a and b, namely, $\langle a\rangle$ and $\langle b\rangle$, are subgroups of $\langle a, b\rangle$, which implies that $|\langle a, b\rangle|$ is divisible by both $\mathsf{ord}(a)$ and $\mathsf{ord}(b)$. Thus, $|\langle a, b\rangle|$ is also divisible by $\mathsf{lcm}\,(\mathsf{ord}(a), \mathsf{ord}(b)) = |\mathcal{G}|$. ■

Lemma 4.3 *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;*
- *$0 < e < N$; and*
- *$x \neq \pm 1 \mod N$ but $x^e = 1 \mod N$,*

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

Proof. Clearly, $e \neq 1$. If $e = 2$ then x is a non-trivial square root of 1, and we can compute $\gcd(N, x \pm 1)$ to get a factor 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 $\mathsf{time}(\mathsf{Factor}(e)) + \mathsf{polylog}(N)$. ■

Lemma 4.4 *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;*
- $0 < e < N$*; and*
- $x \in \mathsf{QR}_{N^2}$ *satisfies* $x \neq 1 \mod N^2$ *but* $x^e = 1 \mod N^2$*,*

outputs P, Q *and has a time complexity* $\mathsf{time}(\mathsf{Factor}'(N, x, e)) \leq \mathsf{time}(\mathsf{Factor}(e)) + \mathit{polylog}(N)$. Note that the conditions on x between this Lemma and Lemma 4.3 are different.

Proof. $x^e = 1 \mod N^2$ implies $x^e = 1 \mod N$, therefore, if $x \neq \pm 1 \mod N$ then we apply algorithm Factor' from the proof of Lemma 4.3.

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 $P = \gcd(N, e)$. ■

Lemma 4.5 *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* $\mathsf{time}(\mathsf{Factor}'(N, x, \beta)) \leq \beta \log^3(N)$.

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 $x^{p_1} = 1 \mod N^2$ 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 $p_i < \beta < N$, Lemma 4.4 implies that, wlog, either $x = 1 \mod N$ and $\gcd(p_i, N) = P$, or $x \neq 1 \mod N$ and $\gcd(x-1, N) = P$. ■

We now present a lemma for the key generation:

Lemma 4.6 *Let $\mathcal{G}$ be a cyclic group and let $a \in \mathcal{G}$ be a uniformly random element, then $\frac{|\mathcal{G}|}{ord(a)}$ is β-smooth with probability at least $1 - \frac{\log ord(\mathcal{G})}{\beta \log \beta}$.*

Proof. Let $g \in \mathcal{G}$ be a generator, and fix a prime factor $p > \beta$ of $\mathsf{ord}(\mathcal{G})$. Given an element $a \in \mathcal{G}$, we can write $a = g^i$ for $0 \leq i < \mathsf{ord}(\mathcal{G})$. Clearly, p divides $\frac{|\mathcal{G}|}{\mathsf{ord}(a)}$ if and only if $p|i$. Therefore, the probability that p divides $\frac{|\mathcal{G}|}{\mathsf{ord}(a)}$ 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 $\prod_i p_i \leq \mathsf{ord}(\mathcal{G})$. By the assumption, $p_i > \beta$ for all i, so $\beta^k \leq \mathsf{ord}(\mathcal{G})$ and $k \leq \frac{\log \mathsf{ord}(\mathcal{G})}{\log \beta}$.

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

By Lemma 4.6, if we define $\beta_\sigma := 2^{\sigma+2} \log(|\mathcal{G}|)$, then the probability that $\frac{|\mathcal{G}|}{\mathsf{ord}(g)}$ is β_σ smooth is greater than $1 - 2^{-(\sigma+1)}$. We take $(\sigma+1)$ bits of statistical soundness in order to end up with σ bits of soundness for the key generation overall.

Corollary 4.7 *Let $\mathcal{G}$ be a cyclic group, $a \in \mathcal{G}$ be a uniformly random element, and $\beta > n$, then $\frac{|\mathcal{G}|}{ord(a^{\Delta_n})}$ is β-smooth with probability at least $1 - \frac{\log ord(\mathcal{G})}{\beta \log \beta}$.*

Proof. By Lemma 4.6, it follows that $\frac{|\mathcal{G}|}{\mathsf{ord}(a)}$ is β-smooth with probability $1 - \frac{\log \mathsf{ord}(\mathcal{G})}{\beta \log \beta}$. We note that $\frac{|\mathcal{G}|}{\mathsf{ord}(a^{\Delta_n})} = \frac{|\mathcal{G}|}{\mathsf{ord}(a)/\gcd(\Delta_n, \mathsf{ord}(a))} = \frac{|\mathcal{G}| \cdot \gcd(\Delta_n, \mathsf{ord}(a))}{\mathsf{ord}(a)}$. In particular, since Δ_n is a β-smooth number (as all factors of Δ_n are smaller than n), so is $\gcd(\Delta_n, \mathsf{ord}(a))$ and the product of two β-smooth numbers is a β-smooth number.

4.4 Batching

Batching techniques allows 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 $(\tilde{h}, \tilde{b}) \in \mathsf{EDL}^2$ requires raising g and h to the power of r, which is a large exponents. Then, without batching, proving B statements $(\tilde{h}_i, \tilde{b}_i)$ requires raising g, h_i to the power of large exponents r_i. To improve efficiency, we use the 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.

Intuitively, the soundness of the batched protocol relies on the fact that it is not possible for the prover to pick statements $(\tilde{h}_i, \tilde{b}_i)$, of which at least one is incorrect, such that their random combination $(\tilde{h}, \tilde{b})$ 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 4.6).

The batched proof of equality of discrete logs is formally described in Protocol 4.3.

PROTOCOL 4.3 ($\Pi^B_{\mathsf{EDL}^2}$*: Batched Proof for* EDL^2*.*)

Parameters. The output of $(N, \tilde{g}, a; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma)$.

Protocol.

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

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 $(\tilde{h}, \tilde{b}) \in \mathsf{EDL}^2$.

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

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

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 (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 4.2.

Next, we argue soundness:

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

$$\begin{aligned}\Pr[(\mathcal{P}_2^*()) \leftrightarrow \mathcal{V}((\tilde{h}_i, \tilde{b}_i)_{i\in[B]})) = 1 \;|\exists i \in [B] : (\tilde{h}_i, \tilde{b}_i) \notin L_{\mathsf{EDL}^2}[N, \tilde{g}, a; x],\\ (\tilde{h}_i, \tilde{b}_i)_{i\in[B]} \leftarrow \mathcal{P}_1^*(1^\kappa, 1^{2^\sigma}, N, \tilde{g}, a; x),\\ (N, \tilde{g}, a; x) \leftarrow \mathit{Setup}(1^\kappa, 1^\sigma)] = \varepsilon.\end{aligned}$$

Then assuming factorization is hard, $\varepsilon = \mathit{neg}(\kappa)$.

Proof. For brevity, denote the event that the prover attempted cheating given a correct setup by

$$\begin{aligned}\mathsf{Cheat} = \exists i \in [B] : (\tilde{h}_i, \tilde{b}_i) \notin L_{\mathsf{EDL}^2}[N, \tilde{g}, a; x] \;\wedge\\ (\tilde{h}_i, \tilde{b}_i)_{i\in[B]} \leftarrow \mathcal{P}_1^*(1^\kappa, 1^{2^\sigma}, N, \tilde{g}, a; x) \;\wedge\\ (N, \tilde{g}, a; x) \leftarrow \mathsf{Setup}(1^\kappa, 1^\sigma),\end{aligned}$$

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:

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

Denote $\varepsilon_1 = \Pr\left[\mathsf{Break} | (\tilde{h}, \tilde{b}) \notin L_{\mathsf{EDL}^2}[N, \tilde{g}, a; x]\right]$ and $\varepsilon_2 = \Pr[(\tilde{h}, \tilde{b}) \in L_{\mathsf{EDL}^2} | \mathsf{Cheat}]$. We have $\varepsilon_1 = \mathsf{neg}(\kappa)$ by Theorem 4.1 (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 4.9 below, assuming factorization is hard, which concludes the proof. ■

Lemma 4.9 *Let $\mathcal{P}_1^*$ be a PPT algorithm for which*

$$\Pr\Bigg[(\tilde{h},\tilde{b}) \in L_{\mathsf{EDL}}[N,\tilde{g},a;x] \mid \exists i_0 \in [B] : (\tilde{h}_{i_0},\tilde{b}_{i_0}) \notin L_{\mathsf{EDL}^2}[N,\tilde{g},a;x], \\ \{t_i\} \leftarrow [0,M), \\ \{(\tilde{h}_i,\tilde{b}_i)\}_{i\in[B]} \leftarrow \mathcal{P}_1^*(1^\kappa, 1^{2^\sigma}, N, \tilde{g}, a; x), \\ (N,\tilde{g},a;x) \leftarrow \mathsf{Setup}(1^\kappa,1^\sigma)\Bigg] = \varepsilon$$

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 $\varepsilon > \frac{1}{M}$ then there exist an algorithm that factors N with a time complexity $\mathcal{O}(\sqrt{M} + \log^3 N) + \max\{\mathsf{time}(\mathsf{Factor}(m)) \mid m \in [M]\}$.

Proof. Wlog, we may assume that $i_0 = 1$. By definition, $\tilde{h}^x = \tilde{b}$ if and only if $(\Pi_{i\in[B]}\tilde{h}_i^{t_i})^x = \Pi_{i\in[B]}\tilde{b}_i^{t_i}$, which is equivalent to $\Pi_{i\in[B]}\left(\frac{\tilde{h}_i^x}{\tilde{b}_i}\right)^{t_i} = 1$. Note that we can divide by the $\tilde{b}_i$'s since if one of the $\tilde{b}_i$'s has no inverse in $\mathbb{Z}_{N^2}^*$ then $\gcd(\tilde{b}_i, N^2) \neq 1$. However, if $\gcd(\tilde{b}_i, N^2) \neq 1$ then either the verifier rejects or we can factor N. By isolating the $i_0 = 1$ term we get:

$$\left(\frac{\tilde{h}_1^x}{\tilde{b}_1}\right)^{t_1} = \prod_{i\in[B]\setminus\{1\}} \left(\frac{\tilde{h}_i^x}{\tilde{b}_i}\right)^{-t_i}. \tag{4}$$

Now, if $\varepsilon > \frac{1}{M}$, there must exist $t_2,\ldots,t_B$ and $t_1' \neq t_1$, such that both $(t_1,t_2,\ldots,t_B)$ and $(t_1',t_2,\ldots,t_B)$ satisfy Equation (4). Otherwise (for every $t_2,\ldots,t_B$ there is up to only one t_1 that satisfies the equation), denote by E the event that the equation holds, we get

$$\varepsilon = \Pr[\mathsf{E}] = \sum_{t_2,\ldots,t_B} \Pr_{t_1}[\mathsf{E}|t_2,\ldots,t_B]\Pr[t_2,\ldots,t_B] \leq \sum_{t_2,\ldots,t_B} (1/M)\Pr[t_2,\ldots,t_B] = 1/M$$

by contradiction.

This implies $\left(\frac{\tilde{h}_1^x}{\tilde{b}_1}\right)^{\Delta t} = 1$, where $\Delta t := t_1 - t_1'$, and Δt may not be known to $\mathcal{A}$. Nevertheless, we can find Δt using generic order finding algorithms in complexity $\mathcal{O}(\sqrt{M}\log(N))$ (Pollard's rho algorithm). Given Δt, we can factor N in time complexity $\mathsf{Factor}(\Delta t) + \mathcal{O}(\log^3 N)$ using Lemma 4.4. ■

4.5 Batch Verification

Batch verification is a technique that allows a verifier to simultaneously verify proofs from multiple non-interactive provers, thereby reducing computational load. This 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 $g^x = 1$ and $h^y = 1$ separately, we may sample a randomizers $r_1, r_2 \in [0, 2^\kappa]$ and verify $g^{r_1 x} \cdot y^{r_2 y} = 1$, 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 C exponentiations $(g_i^{t_i})$ 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 where merged into one. In that case, the verifier 'falls back' to verifying each proof individually.

In Algorithm 4.4 below we present the batched verification algorithm, which takes as input a B-batched proof from C provers.

ALGORITHM 4.4 (*C-Batched Verification of B-Batched Proofs*)

Input: The transcripts $\left(\tilde{g}_j, a_j, (\tilde{h}_{i,j}, \tilde{b}_{i,j})_{i\in[B]}, (t_{i,j})_{i\in[B]}, u_j, v_j, e_j, z_j\right)_{j\in[C]}$. [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 4.3.
   (b) Samples $s_j \leftarrow [0, 2^\kappa)$ for each $j \in [C]$.
   (c) Computes using multi-exponentiation (all computation is modulo N^2):
   $\hat{\tilde{g}} = \prod_{j\in[C]} \tilde{g}_j^{z_j \cdot s_j}$, $\hat{\tilde{h}} = \prod_{j\in[C]} \tilde{h}_j^{z_j \cdot s_j}$, $\hat{u} = \prod_{j\in[C]} u_j^{s_j}$, $\hat{v} = \prod_{j\in[C]} v_j^{s_j}$,
   $\hat{a} = \prod_{j\in[C]} a_j^{e_j \cdot s_j}$, $\hat{\tilde{b}} = \prod_{j\in[C]} \tilde{b}_j^{e_j \cdot s_j}$, and $\hat{g} = \hat{\tilde{g}}^2$, $\hat{h} = \hat{\tilde{h}}^2$, $\hat{b} = \tilde{b}^2$.
   (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,
   - $t_i, e_j \in [0, 2^\kappa)$ for all i and j,
   - $\hat{u} = \hat{g} \cdot \hat{a} \mod N^2$ and $\hat{v} = \hat{h} \cdot \hat{b} \mod N^2$.

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

The complexity of the batch verification in Algorithm 4.4 consists of a $2C$-multi-exponentiation with large exponents for $\hat{\tilde{g}}, \hat{\tilde{h}}$ (actually, $C+1$-multi exponentiation, since $\tilde{g}_j = \tilde{g}$ can be merged into one exponentiation, by computing $\hat{\tilde{g}} = \tilde{g}^{\sum_j z_j s_j}$), another 4 C-multi-exponentiation with small exponents for $(\hat{u}, \hat{v}, \hat{a}, \hat{b})$, plus the additional cost of computing $\tilde{h}_j, \tilde{b}_j$ per batch proof (which incurs B-multi-exponentiation with small exponents C times). [8]

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 $\mathcal{O}(\frac{C}{\log(C)} \cdot \log D + \frac{BC}{\log(B)} \cdot \kappa)$.

[8] We note that we can combine all 6 multi exponentiations (in Step (c) into a single one.

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 $\log_{\tilde{g}}(a_j) \neq \log_{\tilde{h}_i}(\tilde{b}_{i,j})$. Then by Theorem 4.8, assuming factorization is hard, $\log_{\tilde{g}}(a_j) \neq \log_{\tilde{h}_j} \tilde{b}_j$. Therefore, $\tilde{h}_j^{x_j \cdot e_j} \neq \tilde{b}_j^{e_j}$. Analogously to the proof of Lemma 4.9,

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

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 Lemma 4.4. 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 4.10 (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 $2C$ 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 (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 $\hat{u} = \prod_{j \in [C]} u_j^{s_j}$, $\hat{v} = \prod_{j \in [C]} v_j^{s_j}$, $\hat{a} = \prod_{j \in [C]} a_j^{e_j \cdot s_j}$, $\hat{b} = \prod_{j \in [C]} b_j^{e_j \cdot s_j}$, and verify that $\hat{u} = g^z \cdot \hat{a}$ 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.*

4.6 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 $Q^\mu \epsilon$ 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 our batched ZKP protocol (Protocol 4.3) 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 $2^{-\kappa}$. However, in our case this could not work since the reduction to factoring itself takes $2^{\kappa/2}$ time, using Pollard's rho algorithm.

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 4.3 fits that definition.

Definition 4.11 (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 4.5 (Thm. 2 in [AFK22]). *The Fiat-Shamir transformation $FS[\Pi]$ 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 4.3 (and particularly the proof of Theorem 4.8) 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.

5 Optimization & Evaluation

5.1 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 $0 \le j \le t$, and each $0 \le k \le B$, compute $y_{j,k} = \mathsf{ct}_j^k$. This takes Bt multiplications.
2. For each $0 \le i < m$, 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:

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

   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.

5.2 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 5.1 ([BDO22, Definition 8]) *A secret sharing scheme is statistically private if for any set of corrupted parties $C \subseteq \{P_1, \dots, P_n\}$ with $|C| \le t$, 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 5.2 ([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., $h_{max} \ge \max\{|h_{C,i}| \mid i \in [0, t], C \subseteq [n], |C| = t\}$.*

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

Theorem 5.1 ([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 $I(\sigma, n, b) \ge 2^{\sigma+2} b h_{max} t$.*

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 5.3 (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 5.4 $h_{max} < (t+1)\Delta_n$.

Proof. Let $C \subseteq [n]$ with $|C| = t$. By Lemma 5.3, 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

$$\sum_{i=0}^{t} a_i x^i \leq \sum_{i=0}^{t} I(\sigma, n, b) \cdot n^i = I(\sigma, n, b) \sum_{i=1}^{t} n^i = I(\sigma, n, b) \frac{n^{t+1} - 1}{n - 1} \leq I(\sigma, n, b) \frac{n^{t+1}}{\frac{n}{2}} = 2In^t$$

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.

5.3 Evaluation

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[9] for constant-time computations over sensitive data to avoid leakage. In addition, we use rayon[10] 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[11] with 96 3$^{\text{rd}}$ generation Intel Xeon Scalable vCPUs @ 3.50GHz and 192GB RAM, and (2) MacBook Pro Apple M1 Max with 10 Cores @ 3.22GHz and 64GB RAM.

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 Section 5.2 above. All presented runtimes are the average over 10 runs.

Figures & tables. In Figure 1 and the supporting Table 1 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 2 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 3 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 $\{12743, 13772, 28385\}$.

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 Sections 2.1 and 5.2), 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

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

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.

Our batching technique (along the multi exponentiation described in Section 5.1) proves itself necessary if a truly scalable solution is needed. For decryption share computation (Table 1) 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, resp. 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 3).

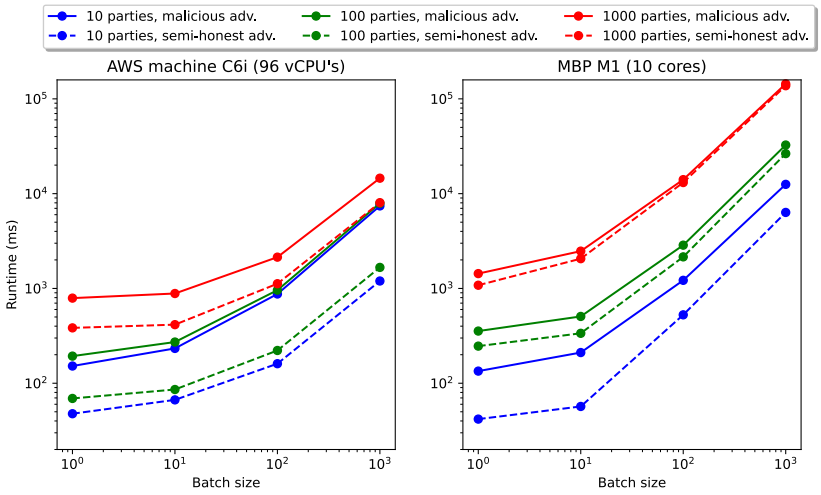

Fig. 1. Time in milliseconds to generate a decryption share with and without proof of equality of discrete logs (e.g., in the presence of a malicious and semi-honest adversaries, resp.).

5.4 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.

Acknowledgements We thank Zeev Manilovich for his support with setting up an accurate benchmarking environment on AWS.

		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 1. 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.

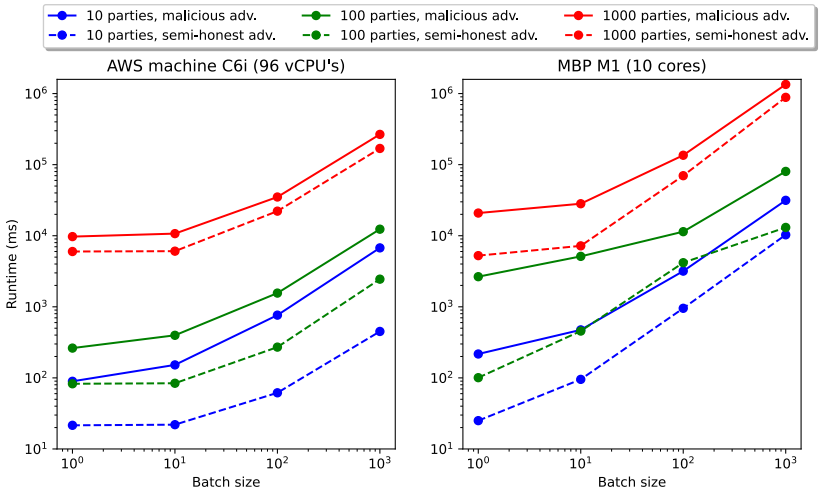

Fig. 2. Time in milliseconds to combine decryption shares from $t = 2n/3$ parties of B ciphertexts, with and without proof of equality of discrete logs (e.g., in the presence of a malicious and semi-honest adversaries, resp.).

		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 2. 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.

		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 3. 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.

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.

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.

BS20. Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

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.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

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.

GG20. Rosario Gennaro and Steven Goldfeder. One Round Threshold ECDSA with Identifiable Abort. 2020.

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.

HL10. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

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.

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 Key Generation Ideal Functionality

The ideal functionality $\mathcal{F}_{\mathsf{KeyGen}}$ is parameterized by a computational security parameter κ, from which $\ell(\kappa)$, the size of the primes, is derived. The functionality interacts with an adversary $\mathcal{A}$ who can corrupt a subset $U \subset [n]$ of the parties chosen at the beginning of the execution. $\mathcal{A}$ my choose its shares $((x_j, y_j))_{j \in U}$. The functionality then samples shares for the honest parties $((x_j, y_j))_{j \notin U}$ such that $x := \sum_{j \in [n]} x_j, y := \sum_{j \in [n]} y_j$ and $N := xy$ is a conforming bi-prime. It then calculates $d = \phi(N) \cdot [\phi(N)^{-1} \mod (N)]$, samples shares d_j of a (t, n) Shamir Secret Sharing over the integers. Lastly, it samples a random element $\tilde{v} \leftarrow \mathbb{Z}_{N^2}$, sets $v = \tilde{v}^2$ and computes $v_j = v^{d_j}$. The functionality sends d_j to party P_j and $(N, \tilde{v}, (v_j)_{j \in [n]})$ to all parties.

FUNCTIONALITY A.1 (*Key Generation Functionality* $\mathcal{F}_{\mathsf{KeyGen}}$)

1. **Receive Adversary Inputs**:
   Upon receiving $(\textbf{input}, \mathsf{sid}, \mathsf{ssid}, ((x_j, y_j))_{j \in U})$ from $\mathcal{A}$, record $(\textbf{input}, \mathsf{sid}, \mathsf{ssid}, P_j, (x_j, y_j))$ for each $j \in U$.
2. **Output Request**:
   Upon receiving $(\textbf{output}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{A}$:
   (a) If a record $(\textbf{input}, \mathsf{sid}, \mathsf{ssid}, P_j)$ is missing for some P_j, ignore.
   (b) Sample shares $x_j, y_j \leftarrow [2^\ell]$ for each $j \notin U$ such that $N = xy$ is a conforming bi-prime, where $x = \sum_{j \in [n]} x_j, y = \sum_{j \in [n]} y_j$.
   (c) Calculate $d = \phi(N) \cdot [\phi(N)^{-1} \mod N]$ and sample shares d_j of a (t, n) Shamir Secret Sharing over the integers of d.
   (d) Sample $\tilde{v} \leftarrow \mathbb{Z}^*_{N^2}$, set $v = \tilde{v}^2$ and $v_j = v^{d_j}$.
   (e) Record all values and send $(\textbf{output}, \mathsf{sid}, \mathsf{ssid}, N, (d_j)_{j \in U}, \tilde{v}, (v_j)_{j \in [n]})$ to $\mathcal{A}$.
3. **Output Denial**:
   Upon receiving $(\textbf{output}, \ldots)$ $\mathcal{A}$ may send:
   – $(\textbf{continue}, \mathsf{sid}, \mathsf{ssid})$ – Send $(\textbf{output}, \mathsf{sid}, \mathsf{ssid}, N, \tilde{v}, (v_j)_{j \in [n]})$ to all parties and $(\textbf{share}, \mathsf{sid}, \mathsf{ssid}, (x_j, y_j), d_j)$ to P_j for each $j \in [n]$.
   – $(\textbf{denial}, \mathsf{sid}, \mathsf{ssid}, U')$ where $\emptyset \neq U' \subseteq U$ – Record $(\textbf{cheaters}, \mathsf{sid}, U')$.
4. **Corrupt**:
   Upon receiving $(\textbf{corrupt}, \mathsf{sid}, \mathsf{ssid}, N', \tilde{v}')$ from $\mathcal{A}$:
   (a) If $(\textbf{denial}, \mathsf{sid}, \ldots)$ is not recorded, ignore the message.
   (b) Re-calculate d', d'_j, v', v'_j as before but for $N', \tilde{v}'$.
   (c) Send $(\textbf{output}, \mathsf{sid}, \mathsf{ssid}, N', \tilde{v}', (v'_j)_{j \in [n]})$ to all parties and $(\textbf{share}, \mathsf{sid}, \mathsf{ssid}, d'_j)$ to P_j for each $j \in [n]$.
5. **Certification**:
   After an $(\textbf{output}, \ldots)$ (and $(\textbf{share}, \ldots)$) message was sent to all parties, each party may send $(\textbf{certify}, \mathsf{sid}, \mathsf{ssid}, P_j)$. Upon receiving this message:
   (a) If $j \in U'$, send $(\textbf{cheaters}, \mathsf{sid}, P_j)$ to all parties and ignore any messages from P_j.
   (b) Else, upon receiving $(\textbf{certify}, \ldots)$ from all parties, if there is no record of $(\textbf{cheaters}, \mathsf{sid}, \ldots)$, send $(\textbf{certify}, \mathsf{sid}, \mathsf{ssid})$ to all parties.