# Fast Two-party Threshold ECDSA with Proactive Security

Brian Koziel[1,2], S. Dov Gordon[1,3], Craig Gentry[1,4]

[1]TripleBlind, Inc., Kansas City, MO

[2]Ideem, Inc., Kansas City, MO, kozielbrian@gmail.com

[3]George Mason University, Fairfax, VA, gordon@gmu.edu

[4]Cornami, Inc., Dallas, TX, craigbgentry@gmail.com

**Abstract**

We present a new construction of two-party, threshold ECDSA, building on a 2017 scheme of Lindell and improving his scheme in several ways.

ECDSA signing is notoriously hard to distribute securely, due to non-linearities in the signing function. Lindell's scheme uses Paillier encryption to encrypt one party's key share and handle these non-linearities homomorphically, while elegantly avoiding any expensive zero knowledge proofs over the Paillier group during the signing process. However, the scheme pushes that complexity into key generation. Moreover, avoiding ZK proofs about Paillier ciphertexts during signing comes with a steep price – namely, the scheme requires a "global abort" when a malformed ciphertext is detected, after which an entirely new key must be generated.

We overcome all of these issues with a proactive Refresh procedure. Since the Paillier decryption key is part of the secret that must be proactively refreshed, our first improvement is to radically accelerate key generation by replacing one of Lindell's ZK proofs – which requires 80 Paillier ciphertexts for statistical security $2^{-40}$ – with a much faster "weak" proof that requires only 2 Paillier ciphertexts, and which proves a weaker statement about a Paillier ciphertext that we show is sufficient in the context of our scheme. Secondly, our more efficient key generation procedure also makes frequent proactive Refreshes practical. Finally, we show that adding noise to one party's key share suffices to avoid the need to reset the public verification key when certain bad behavior is detected. Instead, we prove that our Refresh procedure, performed after each detection, suffices for addressing the attack, allowing the system to continue functioning without disruption to applications that rely on the verification key.

Our scheme is also very efficient, competitive with the best constructions that do not provide proactive security, and state-of-the-art among the few results that do. Our optimizations to ECDSA key generation speed up runtime and improve bandwidth over Lindell's key generation by factors of 7 and 13, respectively. Our Key Generation protocol requires 20% less bandwidth than existing constructions, completes in only 3 protocol messages, and executes much faster than all but OT-based key generation. For ECDSA signing, our extra Refresh protocol does add a 10X latency and 5X bandwidth overhead compared to Lindell. However, this still fits in 150 ms runtime and about 5.4 KB of messages when run in our AWS cluster benchmark.

**Key Words**: Threshold signatures, MPC, ECDSA *

*Work primarily done as part of TripleBlind

# 1 Introduction

## 1.1 ECDSA

Cryptographic signatures enable us to digitally "sign" messages, authenticating that the key owner created the message and that the message was not altered in any way. Our digital infrastructure is built on these signatures, enabling TLS-protected sessions over the internet, digital user authentication, bank transaction authorization, digital contract agreements, and more recently, blockchain transactions. The Elliptic Curve Digital Signature Algorithm (ECDSA) is among the most commonly used signature schemes, as it provides strong security, fast performance, and efficient key/signature sizes. For instance, an ECDSA signature over the elliptic curve secp256r1 (P-256) requires only 64 bytes with sub-millisecond signing/verification operations. ECDSA is widely deployed and also the underlying signature scheme used in Bitcoin and Ethereum.

## 1.2 Threshold Signatures

In classical cryptography, a signer holds and protects their single signing key. The threshold cryptography paradigm, on the other hand, allows this signing key to be split between two or more ("$n$") parties as key shares, such that a threshold ("$t$") number of parties are required to collaborate to sign a message. This $t$-out-of-$n$ threshold signature scheme enforces that a quorum of "$t$" key-share holders must participate. In particular, this has applications to code signing, blockchain operation authorization, and so on. Threshold ECDSA has gained popularity over the past decade for its use in authorizing Bitcoin and Ethereum transactions amongst two or more entities. In one scenario, a cryptocurrency owner may distribute their key to two or more devices as key shares, ensuring that a malicious adversary would have to recover all key shares before depleting funds.

Threshold signatures can be viewed as a particular application of secure multiparty computation, where the parties compute the signing algorithm without revealing their shares of the signing key. However, applying a generic solution for secure computation to the ECDSA signing algorithm would be slow. Instead, prior work has designed custom protocols for threshold ECDSA.

**ECDSA Signatures**

**Input:** Message $M$, secret key $d$, hash function $H$,
**Input:** Elliptic curve $E$ with generator $G$ and order $q$
Compute: $m = H(M)$
Sample: $k \leftarrow [q]$
Compute: $R = k \times G$
Set: $r$ as the $x$-coordinate of $R \mod q$
Compute: $s = k^{-1}(m + rd) \mod q$

**Signature:** $(r, s)$

Figure 1: Single party, standardized ECDSA Signature Generation.

Figure 1 illustrates the ECDSA signing algorithm. ECDSA is not "threshold-friendly" as it requires a non-linear modular multiplication with secret values $k, x$ and a non-linear modular inversion $k^{-1} \mod q$, where $k$ is a randomly chosen signing nonce, $x$ is the master signing key, and $q$ is the order of the elliptic curve group. Generally, one common approach in the literature is to

apply homomorphic multiplication while also requiring expensive zero-knowledge proofs to achieve malicious security.

## 1.3 Lindell's Threshold ECDSA Breakthrough

In 2017, Lindell [33] made a breakthrough with the first truly efficient 2-out-of-2 ("two-party") threshold ECDSA signature. This work featured two-party ECDSA signing in 37 ms, even with communication overhead. Lindell claims to be about two orders of magnitude faster than the best prior work, by Gennaro et al. [23]. The main idea employed by Lindell is to use Paillier homomorphic encryption, which supports homomorphic addition of ciphertexts and multiplication by a known constant, but to avoid (as much as possible) expensive zero knowledge proofs in the signing procedure.

We present a simplification here, to convey Lindell's main idea. The two parties randomly sample an additive sharing of the secret signing key: $x_1, x_2 \leftarrow [q]$. The public key is then determined as $X = X_1 + X_2$ for $X_1 = (x_1 \times G)$ and $X_2 = (x_2 \times G)$. Additionally, party $P_1$ samples a Paillier key $N$, encrypts $C \leftarrow \mathsf{Enc}_N(x_1)$, and sends this to $P_2$. When it is time to sign message $m = H(M)$, the parties sample $k_1, k_2 \leftarrow [q]$, compute $(k_1 \times (k_2 \times G)) = (r_x, r_y)$, and derive $r = r_x \bmod q$, as in standard ECDSA protocol. $P_2$ then computes a "partial signature", homomorphically:

$$\begin{aligned}\mathsf{Enc}_N(k_2^{-1}(m + rx_2)) \oplus (C \odot k_2^{-1}r) \\ &= \mathsf{Enc}_N(k_2^{-1}(m + rx_2) + k_2^{-1}rx_1) \\ &= \mathsf{Enc}_N(k_2^{-1}(m + rx)).\end{aligned}$$

$P_1$ then decrypts and completes the signature by multiplying by $k_1^{-1}$, and reducing modulo $q$.

Lindell's scheme uses a ZK proof in key generation to establish that the same $x_1$ is inside $X_1$ and $C$. This makes key generation slow, but this is a one-time procedure. Signing, on the other hand, uses ZK proofs over the elliptic curve group, but none at all for the (much more expensive) Paillier group! $P_2$ uses $P_1$'s encryption of $x_1$ homomorphically to compute an encrypted partial signature, and sends it without proof! To counteract a malicious $P_2$, $P_1$ globally aborts if $P_2$'s partial signature does not result in a valid completed signature (requiring a fresh key generation). With this trick, signing in Lindell's threshold ECDSA scheme is very fast.

## 1.4 Shortcomings of Lindell's Threshold ECDSA

Lindell's threshold ECDSA has some shortcomings – namely: 1) the expensive key generation procedure, 2) the severe countermeasure to an invalid partial signature from $P_2$ (aborting and requiring a fresh key generation), and 3) the lack of proactive security. Let us look at each of these shortcomings in more detail.

### 1.4.1 Expensive Key Generation

In Lindell's scheme, key generation includes ZK proofs about the Paillier modulus, and that $C$ correctly encrypts $P_1$'s key share. The proof about the Paillier modulus is surprisingly lightweight: Lindell shows that he only needs to prove that $\gcd(N, \phi(N)) = 1$, whereas other schemes typically require stronger properties, such as $N$ being the product of strong primes. However, the proof that $C$ correctly encrypts $P_1$'s key share is expensive. This proof requires $2\tau$ Paillier encryptions

for statistical security $2^{-\tau}$. As a consequence, key generation requires 880 milliseconds on modern AWS clusters, which is nearly $65X$ longer than signing. This range proof also dominates the total bandwidth of keygen, accounting for about 55 KB of the total estimated 62 KB (89%).

Why is Lindell's consistency proof so expensive? Lindell's proof has two parts: 1) A proof that the values inside $X_1$ and $C$ are the same modulo $q$, and 2) A range proof that the value encrypted by $C$ is in $\{1, \ldots, q\}$. One of Lindell's innovations over prior work is the first part of this proof, which is quite fast and elegant, though it requires one more round than we would like. Unfortunately, the range proof, which comes from prior work, is much more expensive. A simple Schnorr-style proof, with just a couple of Paillier ciphertexts, does not seem to be sufficient, because the elliptic curve and Paillier groups have entirely different orders; indeed $q$ and $N$ are co-prime. Instead, the range proof uses bit-wise techniques and amplification; this is where the $2\tau$ Paillier ciphertexts come from.

### 1.4.2 Global Abort and Fresh Key Generation

Lindell makes signing very fast by observing that neither $P_1$ nor $P_2$ needs to generate any ZK proofs over the Paillier group. Instead, $P_1$ handles its ZK proofs over the Paillier group in key generation, while in signing $P_2$ sends its encrypted partial signature without a ZK proof, and $P_1$ decrypts and "verifies" it by confirming that it completes to a verifiable ECDSA signature. If verification fails, $P_1$ globally aborts. After the global abort, if the parties wish to continue signing messages, they must first perform key generation again, replacing the public verification key, and (presumably) updating the PKI to reflect the change. This is a severe countermeasure. However, it is needed because otherwise there is a selective-failure attack, where $P_2$ can potentially recover $P_1$'s key share by observing whether or not $P_1$ outputs a valid signature. This attack is analogous to well-known chosen ciphertext attacks, such as Bleichenbacher's attack [4], where an adversary can recover the secret key by observing decryption error messages.

### 1.4.3 Lack of Proactive Security

Threshold security strengthens a user's key protection by splitting the key into two key shares hosted on different devices, eliminating a single point of failure. Proactive security goes beyond this, allowing the parties to re-randomize their shares (while leaving the public verification key untouched), so that even if an adversary corrupts both parties, the system remains secure as long as the adversary does not corrupt both at the same time. Looking at the simplification of Lindell's scheme above (where sharing is additive versus Lindell's multiplicative), it may appear that making it proactively secure is trivial: Is it not simply a matter of performing a secure coin flip to agree on a uniform $r \leftarrow [q]$, and then storing $x_1 + r \bmod q$ and $x_2 - r \bmod q$? A malicious third-party that recovers one key share would lose their progress after the key shares have changed.

Unfortunately, it is not so simple. $P_1$'s secret includes the factorization of $N$. So, a proactive refresh must also change the Paillier modulus. Here, the first shortcoming of Lindell's scheme – the expensive key generation procedure – rears its head again. Lindell's expensive zero knowledge proof for the correctness of the new ciphertext $C$ would now have to be executed at every key refresh. In some applications, this is prohibitively expensive. For example, imagine a user that signs so infrequently that a refresh is needed before each signature; on a mobile device, spending nearly a full second for every sign-and-refresh operation is simply too much for an end-user. It is not immediately clear how to make Lindell's scheme both proactively secure and practical.

Table 1: Comparison of malicious security two-party ECDSA key generation schemes over curve secp256r1 and the 112-bit NIST security level. Latency was captured on AWS EC2 c5.2xlarge servers.

| Work | Security | Latency [ms] | Bandwidth [KB] | #Messages | Statistical Security $\tau$ |
|---|---|---|---|---|---|
| Castagnos et al. 2020 [10] | Hash Proof | 359 | 5.8 | 4 | $2^{-40}$ |
| Doerner et al. 2018 [17] | OT | **54.2** | 41 | 5 | $2^{-80}$ |
| Lindell 2017 [33] | Paillier-EC | 880 | 62 | 6 | $2^{-40}$ |
| This Work | Paillier-EC-Refresh | 136 | **4.6** | **3** | $2^{-80}$ |

## 1.5 Contributions

Building on Lindell's threshold ECDSA, we design a new threshold ECDSA that solves all three shortcomings simultaneously with a practical proactive refresh procedure. We provide more details below.

**A Lighter Consistency Proof.** As mentioned above, the expensive part of Lindell's construction lies in the consistency proof in key generation that $\exists x_1, \rho$ such that $x_1 \in \{1, \ldots, q\}$, $X_1 = x_1 \cdot G$ and $C = \mathsf{Enc}_N(x_1; \rho)$. This range proof uses amplification, and basically consists of $\tau$ Schnorr-like range proofs.

We replace this with a *single* Schnorr-like range proof, consisting of only two Paillier ciphertexts! Our proof does *not* prove consistency as defined above. Instead, it proves something weaker – namely, as described in Section 3, it proves that $X_1$ encodes $x_1$ and $C$ encrypts $\hat{x}_1$, such that there exists small $\delta$ where the value $[\delta(\hat{x}_1 - x_1)]_N$ is both "small" and divisible by $q$. Notice that, when $\delta = 1$, this closely corresponds with what is proven in Lindell's strong proof, while our weak proof allows other values of $\delta$. So, we allow $P_1$ to cheat to a certain extent. Indeed, if we did not further modify the scheme, $P_1$ could obtain $x_2 \bmod \delta$ for various values of $\delta$, and eventually extract $x_2$. However, we show that the weak proof of consistency is sufficient for security when $P_2$ applies additional perturbations to its encrypted partial signature to protect against a malicious $P_1$. Naturally, our use of the weak consistency proof complicates our security analysis.

Performance-wise, our new approach allows us to improve the runtime of Lindell's Key Generation by a factor of 7X, and his bandwidth by 13X. (Our comparison to other, more recent constructions follows later, and is more mixed. We compare to Lindell here as our techniques are otherwise very similar, so the comparison provides some measure of the impact of our contribution.)

**No Global Abort.** While pursuing proactive security, one might hope to address the second insufficiency of Lindell's construction: the global abort. As mentioned above, by using a bad partial signature, $P_2$ can learn something damaging about the signing key. However, since this misbehavior is detectable, one might hope that $P_1$ could simply call for a refresh operation in place of a global abort. During refresh, new entropy is introduced into the key share, and with it, possibly, the chance to begin again, without resetting the verification key and impacting the PKI. However, this is not as easy as it sounds, as $P_2$ may remain corrupt through the refresh procedure, and will see the randomness used to re-randomize $x_1$.

Nonetheless, we show how to make this work. Instead of encrypting $x_1 \in [q]$ as in Lindell's scheme, $P_1$ instead encrypts a perturbation of $x_1$ – namely, $x_1 + tq$ where $t$ is small but has sufficient min-entropy. When $P_1$ detects that $P_2$ has sent a bad partial signature, $P_1$ changes its Paillier modulus and encrypts a new value $x_1 + t'q$, for a fresh $t'$. The added noise has no impact

Table 2: Comparison of malicious security two-party ECDSA signing schemes over secp256r1 and the 112-bit NIST security level. Latency was captured on AWS EC2 c5.2xlarge servers. A ✓ indicates that a key refresh is included.

| Work | Latency [ms] | Bandwidth [KB] | #Msgs | Proactive? |
|---|---|---|---|---|
| DKLS2018 [17] | **3.0** | 170 | **2** | ✗ |
| Lin2017 [33] | 14.0 | **0.9** | 4 | ✗ |
| CCLST [10, 11] | 113 | 5.3 | 8 | ✗ |
| | 175 | 10.8 | 8 | ✓ |
| This Work | 16.1 | 1.1 | 3 | ✗ |
| | 144 | 5.4 | 3 | ✓ |

on the signature, since it is a multiple of $q$ (the order of the ECDSA group). However, we show that the noise suffices to ensure that an adversary obtains negligible information about $x_1$ even if its corruption of $P_2$ continues across refreshes and it repeatedly sends bad partial signatures, as long as new noise is added between each attempt detected by $P_1$. This allows us to continue using the same public verification key even if $P_2$ sends bad messages.

We base simulation security on an assumption we call "Paillier-EC-Refresh", which is closely related to Lindell's Paillier-EC assumption. Both of these assumptions are stronger than the ordinary notions of security of ECDSA and Paillier. The stronger assumption is needed for simulation security to allow the simulator, as $P_1$, to detect bad partial signatures from $P_2$ using a very limited decryption oracle. Nonetheless, both assumptions seem plausible. Additionally, we provide evidence for the Paillier-EC-Refresh assumption by showing it holds in a generic model for the elliptic curve and Paillier groups. This evidence includes a highly non-trivial characterization of what is revealed about $x_1$ given access to a sequence of limited oracles, each of which leaks some information about a perturbation of $x_1$ by a different multiple of $q$.

**Proactive Security.** With our faster (weak) consistency proof, it becomes much faster to generate a Paillier modulus, $P_1$'s encrypted share under the new modulus, and a weak consistency proof for it. In fact, it becomes practical to do all of this not just in a "one-time" Key Generation procedure, but instead inside a more-frequent Key Refresh procedure, potentially even after every signature.

## 1.6 Other Related Work

Threshold cryptography was initially investigated in the early 1980s through the early 2000s [14, 15, 24, 40, 39, 5, 12, 35] as a method for $n$ users to share a common key, but restricting any use of this key to any subset of $t$ parties. Then, with the rising popularity of Bitcoin in the mid 2010s, a resurgence of threshold research emerged, this time focusing on ECDSA. There is no best-case solution for efficient threshold ECDSA: the latest and greatest research results feature trade-offs between signature generation latency, setup time, bandwidth, and underlying security models/assumptions.

The focus of this work is on the highly optimized, proactive-secure, two-party setting. In the two-party setting, there is no honest majority, which is a specific pain point for some scheme constructions. The very first proven secure protocol threshold signing was proposed by Gennaro et al. in 1996 [24], which used DSA in an honest majority setting. Then in 2001, MacKenzie and Reiter [35] modified the threshold DSA to operate without the honest majority and thus also

allow the two-party setting. Fifteen years later in 2016, Gennaro et al. [23] revisited the threshold DSA/ECDSA signatures to secure Bitcoin wallets, with particular optimizations to the two-party case and thresholds in an honest majority setting.

In 2017, Lindell [33] released his 37 ms two-party ECDSA signing scheme based on various Paillier homomorphic encryption optimizations, shattering previous threshold signing performance metrics by over two orders of magnitude. Shortly after Lindell's work, Doerner et al. [17] published a new oblivious transfer (OT) multiplication technique to achieve two-party ECDSA signatures in less than 10 ms of computation time, but at the expense of almost 200 KB of bandwidth. Most recently in 2019, Castagnos et al. [9] generalized Lindell's scheme to use a hash proof system, effectively tweaking the security model and reducing the two-party key generation and signing bandwidth, at the expense of a 5X increase to sign latency. In follow-up work, Castagnos et al. [10] further reduce the bandwidth requirements by relying on new hardness assumptions that allow them to reduce the number of proof repetitions. Interestingly, reducing the proof repetitions is precisely the approach that we take in our own work, though the techniques are otherwise quite different. Finally, in their most recent work, the authors add proactive security (as well as some other properties) [11]. These three lines of work, by Lindell et al., Castagnos et al., and Doerner et al., represent the pinnacle of speed, efficiency, and security tradeoffs for two-party ECDSA. There are several other recent works that investigate the general $t$-out-of-$n$ case for threshold ECDSA [23, 22, 34, 17, 8, 13, 21, 3, 16], but these are generally comparatively slower in the two-party setting.

Of the three optimized two-party ECDSA schemes in the literature, each original work did not feature proactive security. Only Castagnos et al. [9] released a proactive version of their construction in [11], augmenting their original scheme with proactive security, adaptive security, and identifiable aborts. Otherwise, the only other recent proactive security scheme in the literature is Canetti et al. [8], which features a general threshold ECDSA construction with proactive security, identifiable aborts, and UC-security. Neither work provides performance numbers, and they do not appear to by highly efficient, so we do not provide a performance comparison.

Typically, proactive security is an afterthought, despite its implications on the long-term security of key shares, especially for blockchain applications. The concept of a proactive refresh to thwart a dynamic adversary from corrupting each party one-by-one was first proposed by Ostrovsky and Yung [36] in 1991. The primary idea is that each key share holder will refresh their secret key shares without changing the public key in certain intervals called epochs. Now, an attacker cannot break the scheme unless he corrupts all parties within the epoch. This original work sparked new proactive security works that created, secured, and applied a refresh process to specific threshold schemes, including general threshold construction [29, 28, 2, 6], RSA [18, 19, 38, 20, 1, 30], and ECDSA [8, 32, 11].

## 1.7 Performance Highlights

In the following paragraphs, we compare the performance of our scheme with other recent optimized two-party ECDSA signing schemes. We benchmarked these schemes on an AWS EC2 c5.2xlarge server in loopback mode (i.e. simulating the computations of both parties with no network delay). This AWS server was running Ubuntu 22.04.2 LTS on an Intel Xeon Platinum 8275CL CPU running at 3.0 GHz. These are single-thread benchmarks. We implemented our proposed proactive ECDSA scheme as a proof of concept in Rust 1.77 using the Rust wrapper for OpenSSL 3.0.2 for all underlying arithmetic. All results are for elliptic curve secp256r1.

In Tables 1 and 2, we provide a comparison of key generation and sign protocols, respectively.

Doerner et al.'s* work was run from their public Git repositories, which may include various optimizations since their original paper release. Lindell's work was benchmarked using the same Rust framework as our scheme's results, but public source code can additionally be found on Git.† For Castagnos et al. [10, 11] we used the BICYCL framework.‡ As Table 1 shows, our new Paillier-EC key generation requires only 4.6 KB of bandwidth which is over 20% less than the other schemes. Furthermore, our key generation only requires 1.5 round trips and 136 ms, which is almost 85% less than Lindell's Paillier-EC key generation.

For signature generation, our sign+refresh scheme requires about 144 ms and 5.4 KB total bandwidth. In comparison to our sign performance, our scheme shows the additional overhead for refresh (about 128 ms and 4.3 KB bandwidth). When compared to Lindell's sign procedure, our scheme's sign modifications add about 2.1 ms (15%) and 0.2 KB (22%).

As Table 1 highlights, modern two-party ECDSA focuses on three different types of security foundations: Paillier-EC, Hash Proof, and Oblivious Transfer. The security setting to achieve malicious security creates a tradeoff between performance, bandwidth, number of messages, and various security parameters. There is no clear winner, but each scheme suits a different use case. For instance, the OT-based Doerner et al. [17] features extremely fast performance, but struggles with the largest bandwidth for signing. Meanwhile, Castagnos et al. [10, 11] is a general threshold scheme that features a strong balance of latency and bandwidth, but at the expense of many signing messages. Lastly, our Paillier-EC-based scheme exemplifies good latency, low bandwidth, and a small number of messages, but struggles at higher security levels.

## 1.8 Higher Security Levels

Of the three lines of work previously considered, Castagnos et al.'s scheme [9, 10, 11] scales best with an increasing security level. Our ECDSA scheme was implemented over the elliptic curve secp256r1 with a Paillier modulus of 2048 bits. According to NIST's security level assignment, a 256-bit curve corresponds to the 128-bit security level (roughly equivalent to AES with 128-bit keys) while an RSA modulus of 2048 bits corresponds to the 112-bit security level.

Unfortunately, our scheme's latency and bandwidth do not scale well with the NIST security level as the size of the RSA/Paillier modulus does not scale well. The Paillier modulus $N$ (composed of the product of two large primes) requires a bitlength of 2048, 3072, 7680, and 15360 to correspond to a NIST security level of 112, 128, 192, and 256 bits, respectively. These Paillier moduli require a huge amount of time to find the primes as well as compute exponentiation modulo $N^2$. Furthermore, the Paillier ciphertexts also dominate the total bandwidth. As an example, at the 192-bit security level, a 7680-bit Paillier modulus took a median of 10 seconds to find on our c5.2xlarge platform. When combined with just our Paillier operations, we estimate that Keygen, Sign, and Sign+Refresh executed in 11.3 seconds, 0.47 seconds, and 11.8 seconds, respectively. In this scenario, the bandwidth tripled from our 112-bit security results to 15.8 KB, 2.76 KB, and 18.1 KB for Keygen, Sign, and Sign+Refresh, respectively.

*https://gitlab.com/neucrypt/mpecdsa

†https://github.com/unboundsecurity/blockchain-crypto-mpc

‡https://gite.lirmm.fr/crypto/bicycl

# 2 Preliminaries

## 2.1 Notation

Let $G$ be a generator of the elliptic curve group $\mathbb{G}$ of order $q$. Let $[N]$ be a set of $N$ consecutive integers – for example, the set $\{1, \ldots, N\}$. Let $[\cdot]_N$ denote reduction modulo $N$ into $[N]$. Decryption for a Paillier modulus $N$ will also be into $[N]$. Let $\tau$ be the statistical security parameter while $\kappa$ is the computational security parameter.

## 2.2 Paillier Cryptosystem

The Paillier cryptosystem is a public-key encryption system that enables additive homomorphic computations over its ciphertexts. Originally invented by Pascal Paillier in 1999 [37], the scheme is protected by the decisional composite residuosity assumption, which states that given a composite $n$ and an integer $z$, it is hard to decide whether $z$ is an $n$-residue modulo $n^2$.

Here, we present its simplified implementation. Similar to RSA, two large prime numbers $p$ and $q$ are randomly sampled. Let $N = pq$, $g = N + 1$, $\lambda = \phi(N)$, $\mu = \phi(N)^{-1} \bmod N$. The public encryption key is $(N, g)$ and the private decryption key is $(\lambda, \mu)$.

To encrypt a message $m$ in the range $0 \leq m < N$, select a random $r \in [N]$ with $\text{GCD}(r, N) = 1$ and compute the ciphertext $\text{Enc}(m; r) = C = g^m r^N \bmod N^2$. This ciphertext can be decrypted back to its plaintext message by computing $m = \frac{(C^\lambda \bmod N^2)-1}{N} \mu \bmod N$ (note that the division by $N$ is simply computing the quotient).

The Paillier encryption system is incredibly useful for multiparty computations because of its additive homomorphic nature. Notably, the product of two ciphertexts results in a new ciphertext representing the sum of its plaintexts: $C_1 \oplus C_2 = \text{Enc}(m_1)\text{Enc}(m_2) \bmod N^2 = \text{Enc}(m_1 + m_2 \bmod N)$, and a ciphertext raised to the power of a plaintext results in a new ciphertext representing the product of its plaintexts: $C_1 \odot m_2 = \text{Enc}(m_1)^{m_2} \bmod N^2 = \text{Enc}(m_1 m_2 \bmod N)$. Within this work, the most expensive operations include the generation of two random primes for a new Paillier modulus, exponentiation modulo $N^2$, and exponentiation modulo $N$. For our implemented scheme, $N$ is a 2048-bit modulus.

## 2.3 Active, Proactive Security.

Proactive security ensures unforgeability even when the adversary is able to recover individual shares of the signing key. To ensure this property, the parties holding the shares of the signing key must periodically engage in a key refresh procedure. Security is typically defined using a game-based security definition, where the adversary is allowed to request shares of the secret key after each refresh procedure, and then attempts to create a forgery. (Additionally, as in the standard definition of unforgeability, the adversary can also request signatures on messages of its choosing.) However, we are interested in a stronger security definition: we wish to claim unforgeability, even if the adversary acts maliciously during key generation, signing, or key refresh. As such, we depart from the game-based security definition, and define security through the real/ideal paradigm that is commonly used in secure computation protocols. This ensures security under sequential composition.

We provide a description of our security modeling in Appendix B, and only include a few key details here.

**Admissible adversaries:** Note that security is clearly impossible if we do not place any constraints on the command sequence. In the real-world execution, given the sequence $\big((\mathsf{ssid}, \mathsf{Corrupt}, P_1), (\mathsf{KeyGen}, \mathbb{G}, G, q), (\mathsf{ssid}, \mathsf{De\text{-}corrupt}), (\mathsf{ssid}, \mathsf{Corrupt}, P_2)\big)$, the adversary $\mathcal{A}$ will recover both shares of the secret key $x = x_1 + x_2$. $\mathcal{D}$ can trivially check these values against the reported public key $X$. On the other hand, in the ideal-world, the simulator $\mathcal{S}$ is not given access to the state of $P_1$ or $P_2$, but rather has to provide simulations of those states. A correct simulation would imply finding the discrete log of $X$. Of course, this is the very purpose of the Refresh protocol: we impose the constraint that immediately prior to the command $(\mathsf{ssid}, \mathsf{Corrupt}, P_b)$, we must have the sub-sequence $(\mathsf{ssid}, \mathsf{De\text{-}corrupt})$, $(\mathsf{ssid}, \mathsf{Refresh})$. We note that even this would not suffice, unless we assume that the adversary cannot observe the value of the coin flip used in refreshing the key shares. We therefore assume that the protocol is executed over a private channel, and that the adversary can only observe messages received by a party that is currently corrupt.

We note that our definition is very similar to the standard definition of secure computation with an adaptive adversary. However, in addition to the constraint described above, we also only allow corruptions to occur in-between protocol executions. (This is implicit in our description of command sequences.) In contrast, a truly adaptive adversary can change the corrupted party in the middle of a protocol. Clearly we would face the same issue just described if we allowed the corruption to change in the middle of signing or refresh.

Additionally, as discussed earlier, if $P_1$ receives a bad ciphertext during the execution of $(\mathsf{ssid}, \mathsf{Sign}, m)$, then we require that the parties perform $(\mathsf{ssid}, \mathsf{Refresh})$ before attempting to sign again.

## 2.4 Zero Knowledge

We use two existing zero knowledge proofs from the literature. The first is a simple zero knowledge proof of knowledge of a discrete logarithm. Proofs of knowledge are stronger than standard zero knowledge proofs in that, in addition to guaranteeing soundness, completeness, and zero knowledge, they also provide the additional guarantee that there exists a polynomial-time extraction algorithm that can recover the witness after interacting with the prover. (Of course, a malicious verifier should not be able to do this, but the extractor might re-wind the prover, or use some trapdoor in the parameter setup.) The extraction property also provides the benefit that the proof can be modeled using an ideal functionality, as the simulator can extract the witness and submit it to the functionality. As Lindell did, we therefore ignore the implementation details for this zero knowledge proof, and provide a very simple $\mathcal{F}_{\mathsf{zk}}$ functionality in its place (Figure 2). For more information about secure computation with hybrid functionalities, see Appendix B.

**The Zero-Knowledge Functionality $\mathcal{F}_{\mathsf{zk}}^{R}$ for Relation $R$**

Upon receiving (prove, *sid*, $x$, $w$) from a party $P_i$(for $i \in 1, 2$): if $(x, w) \notin R$ or *sid* has been previously used then ignore the message. Otherwise, send (proof, $sid, x$) to party $P_{3-i}$.

Figure 2: The Zero-Knowledge Functionality $\mathcal{F}_{\mathsf{zk}}$ for Relation $R$ (copied verbatim from [33])

When $P_2$ constructs proofs of knowledge for discrete log, we need to commit to the proof prior to sending it. $P_2$ only sends its proof after receiving and verifying the proof of $P_1$; this ensures independence of the two instances in their proof. Again, as Lindell did before us, we simplify this into a single functionality that combines the commitment and the proof of knowledge (Figure 3). This simplifies the presentation, removing the need to specify how commitments are implemented.

**The Committed NIZK Functionality $\mathcal{F}^R_{\mathsf{com-zk}}$ for $R$**

Functionality $\mathcal{F}_{\mathsf{com-zk}}$ works with parties $P_1$ and $P_2$ as follows:

- Upon receiving (com-prove, $sid, x$) from a party $P_i$(for $i \in 1, 2$): if $(x, w) \notin R$ or $sid$ has been previously used then ignore the message. Otherwise, store $(sid, i, x)$ and send (proof-receipt, $sid$) to $P_{3-i}$.
- Upon receiving (decom-proof, $sid$) from party $P_i$: if $(sid, i, x)$ has been stored then send (decom-proof, $sid, x$) to party $P_{3-i}$.

Figure 3: The Committed NIZK Functionality $\mathcal{F}^R_{\mathsf{com-zk}}$ for $R$ (copied verbatim from [33])

### 2.5 Zero Knowledge Proof of GCD

Our protocols rely on zero knowledge proofs for several languages, which we present independently, in order to simplify the presentation of the signature scheme itself.

During key generation and key refresh, the server samples a new Paillier modulus $N$. Although this modulus is supposed to be the product of two large primes, as shown by Lindell [33], if the server proves only that $\mathsf{GCD}((N), \phi(N)) = 1$, this suffices for ensuring the expected homomorphic properties of the Paillier encryption scheme. (Other complications arise from this relaxation, but we address those below.)

$\mathsf{L}_{\mathsf{GCD}}$, is the set of positive integers $N$ such that $\mathsf{GCD}(N, \phi(N)) = 1$. The zero knowledge proof that we use is from Goldberg et al. [26]. We present it in Appendix C for completeness.

## 3 Zero Knowledge Proof of Consistency

The next zero knowledge proof that we present is a proof of "loose consistency" between a discrete log instance, and a Paillier plaintext. Although the proof has been presented elsewhere, here we change the standard soundness claim in order to avoid full soundness amplification. We therefore present the result in its own Section. Concretely, a tuple $(C, X_1, q, N) \in \mathsf{L}_{\mathsf{eq}}$ if there exist $\hat{x}_1 \in [N], \delta, x_1 \in [q]$ such that the following hold:

- $C = \mathsf{Enc}_N(\hat{x}_1)$,
- $X_1 = x_1 \times G$,
- $[\delta(\hat{x}_1 - x_1)]_N \in [3q^2 2^{2(\tau+\kappa)}]$. ($[\delta(\hat{x}_1 - x_1)]_N$ is "small".)
- $[[\delta(\hat{x}_1 - x_1)]_N]_q = 0$. ($[\delta(\hat{x}_1 - x_1)]_N$ is a multiple of $q$.)

In honest executions of key generation and key refresh, $x_1 \in [q]$ and $\hat{x}_1$ is a small positive integer satisfying $\hat{x}_1 = x_1 \bmod q$, so that the above conditions are trivially satisfied with $\delta = 1$. Suppose we wished to prove this stronger claim of honest behavior and attempted to prove it as follows. The prover samples a random value $b \leftarrow [q^2 2^{2(\tau+\kappa)}]$, and sends $\mathsf{Enc}_N(b)$ and $b \times G$ to the verifier. The verifier replies with a random challenge $\sigma$, and, finally, the prover reveals $z_1 = \hat{x}_1\sigma + b$, together with the encryption randomness that results from computing $\mathsf{Enc}_N(\hat{x}_1\sigma + b)$ homomorphically. The verifier checks that the two prover messages are consistent with the instance and $\sigma$ – that is, $(\mathsf{Enc}_N(\hat{x}_1) \odot \sigma) \oplus \mathsf{Enc}_N(b) \stackrel{?}{=} \mathsf{Enc}_N(z_1)$, and $X_1 \times \sigma + (b \times G) \stackrel{?}{=} (z_1 \times G)$. However, the prover can cheat if there is a small factor $\delta$ such that $[\delta(\hat{x}_1 - x_1)]_N$ nontrivially satisfies the conditions above.

For example, if the adversary is lucky and $\sigma$ is divisible by $\delta$, then we have that $[\hat{x}_1 \cdot \sigma + b]_N$ might actually be small and congruent to $[x_1 \cdot \sigma + b]_N$ modulo $q$, because $[\sigma(\hat{x}_1 - x_1)]_N$ may be small and congruent to 0 modulo $q$. (The prover can also make its attack less detectable by not requiring $\sigma$ be divisible by $\delta$, but instead congruent to a certain value modulo $\delta$, where that value depends on a tweak the prover has made to $b$.) As in the proof presented previously that $N \in \mathsf{L}_{\mathsf{GCD}}$, we could amplify soundness through repetition. However, we wish to avoid that.

Instead, our main technical contribution is to recognize that we can relax the claim, as described above. Rather than requiring equivalence of $\hat{x}_1$ and $x_1$ over the integers, we instead consider what guarantees *are* provided when we forgo any amplification. We can show that there exist small integers $\delta, k \ll N$ such that $\delta(\hat{x}_1 - x_1) = kq + \ell N$. The fact that these values are small will facilitate our main technical argument (Claim 7 in Section 7) that any leakage stemming from $\hat{x}_1$ can be statistically hidden by $P_2$, simply by adding sufficient noise $\rho q$ to the plaintext, homomorphically.

$\Pi_{\mathsf{eq}}$

**Inputs:** Prover: $\mathsf{ssid}, N, C, X_1, \hat{x}_1, \rho$ (where $C = \mathsf{Enc}_N(\hat{x}_1; \rho)$).
Verifier: $\mathsf{ssid}, N, C, X_1$.

**Prover:** Sample: $\delta \leftarrow \mathbb{Z}_N^*$ ; $b \leftarrow [q^2 \cdot 2^{2(\tau+\kappa)}]$.
Compute: $\gamma_1 = \mathsf{Enc}_N(b; \delta)$ ; $\gamma_2 = b \times G$.
Compute: $\sigma = H(\mathsf{ssid}, C, X_1, \gamma_1, \gamma_2)$.
Compute: $z_1 = (\hat{x}_1 \sigma + b)$ ; $z_2 = (\rho^\sigma \delta \bmod N)$.
Send: $\psi = \gamma_1 || \gamma_2 || z_1 || z_2$.

**Verifier:** Parse $\Psi$ as $\gamma_1$, $\gamma_2$, $z_1$, $z_2$.
Verify: $\gamma_1 \in \mathbb{Z}_{N^2}^*$ ; $z_1 \in [q^2 2^{2(\tau+\kappa)} + (q^2 - q)2^{\tau+2\kappa} - q + 1]$ ; $z_2 \in \mathbb{Z}_N^*$.
Verify: $\gamma_2 \neq \mathsf{0}$ ; $\gamma_2 \in \mathbb{G}$.
Verify: $\mathsf{GCD}(C, N) = 1$, $\mathsf{GCD}(\gamma_1, N) = 1$.
Compute: $\sigma = H(\mathsf{ssid}, C, X_1, \gamma_1, \gamma_2)$.
Verify: $\gamma_1 \oplus C^\sigma = \mathsf{Enc}_N(z_1; z_2)$.
Verify: $\gamma_2 + (X_1 \times \sigma) = z_1 \times G$.

Figure 4: Zero knowledge proof of loose-consistency between a discrete log instance and a Paillier plaintext. We assume $H$ is a random oracle with output in $[q]$ and apply the Fiat-Shamir transform.

**Claim 1** (Soundness.)**.** If $\mathsf{GCD}(N, \phi(N)) = 1$ and if it holds in $\Pi_{\mathsf{eq}}$ that $\Pr_{\sigma \leftarrow \mathbb{Z}_q}[\mathsf{Vrfy}(C, \sigma, \psi) = 1] \geq \frac{1}{\delta'}$ for $\delta' < q$, then there exists $\delta \leq \delta'$ such that:

a) $[\delta y]_N \in [3q^2 2^{2(\tau+\kappa)}]$.

b) $[\delta y]_N \equiv 0 \bmod q$.

*Proof.* By assumption, at least $\frac{q}{\delta'}$ of the $q$ possible challenges lead to verification. It follows that there exist two such challenges $\sigma_1, \sigma_2$ such that $[|\sigma_1 - \sigma_2|]_q \leq \delta'$. Let $\delta = \sigma_1 - \sigma_2$.

The fact that $\mathsf{GCD}(N, \phi(N)) = 1$ implies that every value in $\mathbb{Z}_{N^2}^*$ is a valid Paillier ciphertext with a well-defined encrypted value. Accordingly, from $\Pi_{\mathsf{eq}}$, let $x_1$ be the value encoded by $X_1$, and $\hat{x}_1$ be the value encrypted by $C$. Further, let $b$ be value encoded by $\gamma_2$, and $\hat{b}$ be the value encrypted by $\gamma_1$.

Let $Z_1, Z_2 \in [N]$ be the corresponding integer proof terms corresponding to these challenges,

which must satisfy:

$$\begin{aligned} Z_1 &= [\hat{x}_1\sigma_1 + \hat{b}]_N \\ &= [x_1\sigma_1 + b]_q \\ Z_2 &= [\hat{x}_1\sigma_2 + \hat{b}]_N \\ &= [x_1\sigma_2 + b]_q. \end{aligned}$$

When $\mathsf{GCD}(N, \phi(N)) = 1$, Paillier encryption supports the homomorphic properties "as expected" [31]. Subtracting, we have:

$$\begin{aligned} Z_2 - Z_1 &= [\hat{x}_1\delta]_N \\ &= [x_1\delta]_q, \end{aligned}$$

which implies

$$\begin{aligned} Z_2 - Z_1 &= \hat{x}_1\delta + \ell N \\ &= x_1\delta + kq, \end{aligned}$$

for integers $k, \ell$, and therefore

$$\delta y = (\hat{x}_1 - x_1)\delta = kq - \ell N.$$

As a condition of verification, $Z_1, Z_2 \in [2q^2 2^{2(\tau+\kappa)}]$. Because $|Z_2 - Z_1| \in [2q^2 2^{2(\tau+\kappa)}]$ and $x_1, \delta \leq q$, we have that $|kq| = |Z_2 - Z_1 - x_1\delta| < 3q^2 2^{2(\tau+\kappa)}$. Since $[\delta y]_N = kq$, the claim follows. □

**Claim 2** (Zero Knowledge.)**.** If $\mathsf{GCD}(N, \phi(N)) = 1$, there exists a probabilistic, polynomial time algorithm $\mathcal{S}$ that on input $(\mathsf{ssid}, N, C, X_1)$ outputs a transcript $\mathsf{view}$ that is indistinguishable from the view of the verifier when interacting with the real prover.

*Proof.* Note that when $\mathsf{GCD}(N, \phi(N)) = 1$, homomorphic operations are "erased" by the encryption randomness. $\mathcal{S}$ simulates $\psi$ as follows:

- Sample $\sigma \leftarrow \mathbb{Z}_q$, $z_1 \leftarrow [q^2 + q^2 2^{\tau+2\kappa}(1 + 2^\tau)]$, and $z_2 \leftarrow \mathbb{Z}_N^*$.
- Compute $C_{z_1} = \mathsf{Enc}_N(z_1; z_2)$.
- Set $\gamma_1 = C^{-\sigma} \oplus C_{z_1}$
- Set $\gamma_2 = (-\sigma \times X_1) + (z_1 \times G)$.
- Set $\psi = \gamma_1 || \gamma_2 || z_1 || z_2$.
- Program $H(\mathsf{ssid}, C, X_1, \gamma_1, \gamma_2) = \sigma$.

Note that the encrypted/encoded values are identical in the true execution and simulation with $b = z_1 - \hat{x}_1\sigma$. Regarding the encryption randomness, in both the true execution and simulation, $C$ has some randomness $\rho$, and $z_2$ is the randomness revealed in the proof. In the true execution, $\gamma_1$ has randomness $\delta$ which satisfies $z_2 = \rho^\sigma \delta \bmod N$. In the simulation, $\gamma_1$ has randomness $z_2/\rho^\sigma$, which is identical. □

# 4 Key Generation

Key Generation is essentially a Diffie-Hellman key exchange resulting in $(x_1+x_2)\times G$, accompanied by a Paillier encryption of $P_1$'s share of the secret key: $\mathsf{Enc}_N(x_1)$. However, in order to prove security of the scheme, we require several commitments, and zero knowledge proofs.

$P_2$ begins the protocol by sampling a random $x_2 \leftarrow [q]$, computing $X_2 = x_2 \times G$, and sending a commitment to both $X_2$ and its proof of knowledge of $x_2$. This is modeled by a call to the $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$ functionality.

After receipt of the commitment, $P_1$ samples $x_1 \leftarrow [q]$ and computes $x_1 \times G$. It samples a new Paillier key $N = PQ$ and creates a proof $\pi$ that $\mathsf{GCD}(N, \phi(N)) = 1$ (See Section 2.5). $P_1$ samples a perturbation factor $t \leftarrow [2^{\tau+2\kappa}]$ to compute $\hat{x}_1 = x_1 + tq$. Finally, it samples $C = \mathsf{Enc}_N(\hat{x}_1)$, and creates a proof of consistency for $C$ and $X_1$: $\Psi \leftarrow \Pi_{\mathsf{eq}}.\mathsf{Prv}(N, C, X_1, \hat{x}_1)$. It sends $(C, N, \pi, \Psi)$.

After verifying the proofs sent by $P_1$, $P_2$ decommits to $X_2$, and the proof of knowledge of $x_2$ (again through the $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$ functionality). At the end of the protocol, each party stores their key share, $x_b$, $N$ and $X$ and $X_1$ for use in signing and refresh. $P_2$ additionally stores $C$ for use in signing, and $P_1$ stores $P, Q$ for signing.

**Key Generation**

**Input:** None.

$P_2$: Sample $x_2 \leftarrow [q]$, compute $X_2 = x_2 \times G$ and submit $(\mathsf{ssid}_1, \mathsf{proof\text{-}commit}, X_2, x_2)$ to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$.

$P_1$:
- Sample $x_1 \leftarrow [q]$, compute $X_1 = x_1 \times G$ and submit $(\mathsf{ssid}_2, \mathsf{Prove}, X_1, x_1)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$.
- Sample Paillier public key $N = PQ$ and $\pi \leftarrow \Pi_{\mathsf{GCD}}.\mathsf{Prv}(N, \phi(N))$.
- Sample $t \leftarrow [2^{\tau+2\kappa}]$, $C \leftarrow \mathsf{Enc}_N(x_1 + tq)$, and $\Psi \leftarrow \Pi_{\mathsf{eq}}.\mathsf{Prv}(N, C, X_1, x_1)$.
- Send $(C, N, \pi, \Psi)$.

$P_2$: If $\Pi_{\mathsf{GCD}}.\mathsf{Ver}(N, \pi) = 0$ or $\Pi_{\mathsf{eq}}.\mathsf{Ver}(C, X_1, \Psi) = 0$, abort. Otherwise, submit $(\mathsf{ssid}_1, \mathsf{decom\text{-}proof})$ to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$.

$P_1$: Receive $(\mathsf{ssid}_1, \mathsf{decom\text{-}proof}, X_2)$, or else abort.

**Store:** $P_1$: $(x_1, X_1, X_2, X = X_2 + X_1, N, P, Q)$.
$P_2$: $(x_2, X_1, X_2, X, C, N)$.

Figure 5: Message specification for two-party ECDSA key generation.

# 5 Signing

Recall that an ECDSA signature has the form $(r, s)$, where $r$ is derived by sampling a random nonce $k \leftarrow [q]$, and taking the x-coordinate of $k \times G$. $s$ is then computed as $s = k^{-1}(m + rx) \bmod q$. Revealing the nonce $k$ to either party would leak information about $x$, so we instead use $k = k_1 k_2 \bmod q$, where party $b$ samples $k_b$.

$P_2$ begins by sampling $k_2 \leftarrow [q]$, computing $K_2 = k_2 \times G$, and committing to $K_2$ and the proof of knowledge of $k_2$ (again through $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$). $P_1$ responds by sampling $k_1$, computing $k_1 \times G$, and proving knowledge of both $k_1$ and $x_1$. As $P_1$ acts second, there is no need for a commitment to these proofs, so it instead uses the simpler $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$ functionality.

**Signing Message Specification**

**Input:** $P_1$: $(x_1, X, X_1, X_2, M, N, P, Q, \mathsf{ssid})$.
$P_2$: $(x_2, X, X_1, X_2, C, M, N, \mathsf{ssid})$.
If $\mathsf{ssid}$ has been used before, quit.

$P_2$: Sample $k_2 \leftarrow [q]$, compute $K_2 = k_2 \times G$, and submit $(\mathsf{ssid}||1, \mathsf{proof\text{-}commit}, K_2, k_2)$ and $(\mathsf{ssid}||2, \mathsf{proof\text{-}commit}, X_2, x_2)$ to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$.

$P_1$: Sample: $k_1 \leftarrow [q]$, compute $K_1 = k_1 \times G$, and submit $(\mathsf{Prove}, \mathsf{ssid}||3, K_1, k_1)$ and $(\mathsf{Prove}, \mathsf{ssid}||4, X_1, x_1)$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$.

$P_2$:
- Receive $(\mathsf{Proof}, \mathsf{ssid}||3, K_1)$ and $(\mathsf{Proof}, \mathsf{ssid}||4, X_1)$ from $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$, or else abort.
- Compute $K = (k_2 \times K_1) = (r_x, r_y)$ ; $r = r_x \bmod q$ ; $m = H(M)$.
- Sample $\tilde{\rho} \leftarrow [q]$ and $\rho \leftarrow [3q^2 2^{3\tau+2\kappa}]$.
- Compute $\tilde{k}_2^{-1} = [k_2^{-1}]_q + \tilde{\rho} q$, and $C' = (\mathsf{Enc}_N(\rho \cdot q + (\tilde{k}_2^{-1}(m + rx_2)))) \oplus (C \odot r\tilde{k}_2^{-1})$.
- Submit $(\mathsf{ssid}||1, \mathsf{decom\text{-}proof})$ and $(\mathsf{ssid}||2, \mathsf{decom\text{-}proof})$ to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$, and send $C'$ to $P_1$.

$P_1$:
- Receive $(\mathsf{ssid}||1, \mathsf{decom\text{-}proof}, K_2)$ and $(\mathsf{ssid}||2, \mathsf{decom\text{-}proof}, X_2)$, or else abort.
- Compute $K = (k_1 \times K_2) = (r_x, r_y)$ ; $r = r_x \bmod q$ ; $m = H(M)$.
- Sample $\ell \leftarrow [q 2^{2(\tau+\kappa)}]$.
- Compute $s_0 = \mathsf{Dec}(C')$ ; $s_1 = [s_0]_q$; $s_2 = s_0 - s_1 + \ell q$; $s_3 = [k_1^{-1} s_1]_q$.
- Verify that $s_2 < \frac{N}{2^{\tau+2\kappa}}$, $s_2$ is divisible by $q$, and that $\min(s_3, q - s_3)$ is a valid signature on $M$. If not, abort until the next $\mathsf{Refresh}$.

**Output:** $(r, s)$.

Figure 6: Two party protocol for threshold signing of a message $M$.

$P_2$ then computes a "partial signature", homomorphically. If $P_2$ knew that $C = \mathsf{Enc}_N(x_1)$ (as it should), it would simply compute

$$\begin{aligned}\mathsf{Enc}_N(k_2^{-1}(m + rx_2)) \oplus (C \odot k_2^{-1} r) &= \mathsf{Enc}_N(k_2^{-1}(m + rx_2) + k_2^{-1} r x_1) \\ &= \mathsf{Enc}_N(k_2^{-1}(m + rx)).\end{aligned}$$

The resulting "partial signature" would be sent to $P_1$, who decrypts and multiplies by $k_1^{-1}$ to complete the signing process.

Recall, however, that $\Pi_{\mathsf{eq}}$ does not ensure that $C$ is (entirely) consistent with $X_1$. Because $P_2$ cannot trust that $C = \mathsf{Enc}_N(x_1)$, we require some additional noise in the plaintext. $P_2$ samples random multiples of $q$, and uses one to "smudge" the value of $k_2^{-1}$, and the other to smudge the whole plaintext. Specifically, $P_2$ sends a ciphertext encrypting $\rho q + \tilde{k}_2^{-1}(m + rx_2) + \tilde{k}_2^{-1} r x_1)$, where $\rho \leftarrow [q^3 \cdot 2^\tau]$, and $\tilde{k}_2^{-1} = k_2^{-1} + \tilde{\rho} q$ for random $\tilde{\rho} \leftarrow [q]$. We will prove in Section 7 that, conditioned on a verifying proof from $\Pi_{\mathsf{eq}}$, which would have been provided either during key generation, or doing the previous refresh operation, these two random multiples of $q$ suffice for ensuring that nothing about $k_2$ or $x_2$ leaks to $P_1$, even if $C$ encrypts $\hat{x}_1 \neq x_1$.

After $P_1$ decrypts to recover $s_2 = \mathsf{Dec}_N(C')$, it completes the signing operation by multiplying $s = k_1^{-1} s_2$, and setting the signature to be the minimum of $(s, q - s)$. Finally, and importantly, $P_1$ verifies the signature against the public key $X$. If the signature fails to verify, then $P_1$ must refuse to perform additional signatures prior to a refresh operation. This is because a bad ciphertext

generated by $P_2$ could leak a few bits of information about $\rho q + x_1 \bmod q$. Under the Paillier-EC-Refresh assumption, we prove in Section 7 that a refresh, with a new $\rho q$, suffices to ensure security going forward, even without changing $x_1$ or the public key.

# 6 Key Refresh

The Refresh operation re-randomizes the secret key of the two parties, as well as the Paillier modulus and ciphertext used to encrypt $x_1$. The parties flip a random coin $r \leftarrow [q]$ in order to re-randomize their shares. We treat this as an ideal function call for simplicity, though it can be trivially realized with a standard commit-and-reveal protocol. Once they've agreed on random $r$, $P_1$ updates $x_1$ by subtracting $r$, while $P_2$ updates $x_2$ by adding $r$; $\hat{x}_1 = x_1 - r$, and $\hat{x}_2 = x_2 + r$. This results in a new, uniform secret sharing $x$, without any changes to the public key. They each update $X_1$ as well, by subtracting $r \times G$: $\widehat{X}_1 = X_1 - (r \times G)$.
Just as in Key Generation, $P_1$ then samples a new $N = PQ$, $\pi \leftarrow \Pi_{\mathsf{GCD}}.\mathsf{Prv}(N, \phi(N))$, $C \leftarrow \mathsf{Enc}_N(\hat{x}_1)$, and $\Psi \leftarrow \Pi_{\mathsf{eq}}.\mathsf{Prv}(N, C, \widehat{X}_1, \hat{x}_1)$. $P_2$ verifies the proofs, and the protocol terminates.
Finally, $P_1$ samples new noise for protecting the Paillier plaintext, sampling $t \leftarrow [2^{\tau+2\kappa}]$, and creating a new ciphertext $C \leftarrow \mathsf{Enc}_N(\hat{x}_1 + tq)$.

**Key Refresh Message Specification**

**Input**: $P_1$: $(x_1, X_1, X)$.
$P_2$: $(x_2, X_1, X)$.

$P_2$: Send $\mathsf{ssid}$ to $\mathcal{F}_{\mathsf{coin}}$ and receive $r$ in response.
Compute $\hat{x}_2 = x_2 + r \bmod q$ and $\widehat{X}_1 = X_1 - (r \times G)$.

$P_1$:
- Send $\mathsf{ssid}$ to $\mathcal{F}_{\mathsf{coin}}$ and receive $r$ in response.
- Compute $\hat{x}_1 = x_1 - r \bmod q$ and $\widehat{X}_1 = X_1 - (r \times G)$.
- Sample Paillier public key $N = PQ$ and $\pi \leftarrow \Pi_{\mathsf{GCD}}.\mathsf{Prv}(N, \phi(N))$.
- Sample $t \leftarrow [2^{\tau+2\kappa}]$, $C \leftarrow \mathsf{Enc}_N(\hat{x}_1 + tq)$, and $\Psi \leftarrow \Pi_{\mathsf{eq}}.\mathsf{Prv}(C, \widehat{X}_1)$.
- Send $(C, N, \pi, \Psi)$.

$P_2$: If either $\Pi_{\mathsf{GCD}}$ or $\Pi_{\mathsf{eq}}$ fail to verify, abort.

**Store:**
$P_1$ stores $(\hat{x}_1, \widehat{X}_1, X, C, N, P, Q)$.
$P_2$ stores $(\hat{x}_2, \widehat{X}_1, X, C, N)$.

Figure 7: Message specification for 2-party ECDSA Key Refresh.

# 7 Simulation-based Proof of Security

In this section, we will prove that our threshold signature scheme securely realizes the $\mathcal{F}_{\mathsf{TS}}$ functionality described in Figure 8, assuming the Paillier-EC-Refresh assumption holds. As described

in Section 2.3, we consider stand-alone security against a proactive, admissible adversary. We begin by describing the new assumption.

$\mathcal{F}_{\mathsf{TS}}$

**KeyGen:** Upon receiving input $(\mathsf{ssid}, \mathsf{KeyGen})$ from both parties,

- Sample $x_1, x_2 \leftarrow \mathbb{G}$.
- Set $x = x_1 + x_2$ ; $X = g^x$.
- Store: $(\mathsf{ssid}, x)$.
- Output: $X$ to both parties.

**Sign:** Upon receiving input $(\mathsf{ssid}, M)$ from both parties,

- Retrieve $(\mathsf{ssid}, x)$ from storage.
- Output $(r, s) \leftarrow \mathsf{ECDSA.Sign}(x, M)$ to both parties.

Figure 8: Reactive functionality for threshold signatures

## 7.1 The Paillier-EC-Refresh Assumption

First, let us review Lindell's Paillier-EC assumption [33].

Paillier-EC Assumption.

Consider the following experiment with an adversary $\mathcal{A}$, denoted $\mathsf{Expt}_{\mathcal{A}}(1^\kappa)$:

1. Generate a Paillier key pair $(N, (P, Q))$.
2. Choose random $w_0, w_1 \leftarrow [q]$ and compute $Q = w_0 \times G$.
3. Choose a random bit $b \leftarrow \{0, 1\}$ and compute $C = \mathsf{Enc}_N(w_b)$.
4. Let $b' = \mathcal{A}^{\mathcal{O}_C(\cdot,\cdot,\cdot)}(N, C, Q)$, where $\mathcal{O}_C(C', \alpha, \beta) = 1$ for ciphertext $C'$ and $\alpha, \beta$ if and only if $\mathsf{Dec}_{\phi(N)}(C') = \alpha + \beta \cdot w_b \bmod q$; otherwise, $\mathcal{O}_C$ returns 0 and permanently aborts.
5. The output of the experiment is 1 if and only if $b' = b$.

The Paillier-EC assumption is that, for every PPT algorithm $\mathcal{A}$, there is a negligible function $\mu$ such that $\Pr[\mathsf{Expt}_{\mathcal{A}}(1^\kappa) = 1] \leq \frac{1}{2} + \mu(\kappa)$.
Note that the oracle $\mathcal{O}_C$ stops working after the first time it returns 0. When that happens, the simulator must also abort. Therefore, the scheme itself aborts upon an incorrect partial signature from $P_2$.
Compare Paillier-EC to Paillier-EC-Refresh, given below.

Paillier-EC-Refresh Assumption.

Let $k$ be an integer representing the number of oracles. (We will later set this to $2^\kappa$ as a bound on the number of queries an adversary can make.) Let $I_1, I_2$ be the intervals of consecutive integers, such that $|I_1|$ is divisible by $q$, $|I_1| \geq k \cdot q \cdot 2^{\tau+\kappa}$ for statistical and computational security parameters $\tau$ and $\kappa$, and (for all $N_i$ used in the system) $I_1$ and $I_2$ are inside $[N_i]$ and $2|I_1||I_2| < N_i q$.
Consider the following experiment with an adversary $\mathcal{A}$, denoted $\mathsf{Expt}_{\mathcal{A}}(1^\kappa)$:

1. Choose random $w_0, w_1 \leftarrow \mathbb{Z}_q$ and compute $Q = w_0 \cdot G$.

2. Give Adversary $\mathcal{A}$ the value $Q$.
3. Choose a random bit $b \leftarrow \{0,1\}$.
4. For $i \in [k]$:
   (a) Generate Paillier key pair $(N_i, (P_i, Q_i))$.
   (b) Sample $\tilde{w}_i$ uniformly from $I_1$ subject to $\tilde{w}_i = w_b \bmod q$.
   (c) Compute $C_i = \mathsf{Enc}_{N_i}(\tilde{w}_i)$.
5. Give Adversary $\mathcal{A}$ the public keys $\{N_i\}$ and ciphertexts $\{C_i\}$.
6. Give Adversary $\mathcal{A}$ access to oracles $\mathcal{O}_{C_i}$ for $i \in [k]$, where $\mathcal{O}_{C_i}(C', \alpha, \beta) = 1$ iff $[(\mathsf{Dec}_{\phi(N)}(C') - (\alpha + q\ell + \beta \cdot \tilde{w}_i))]_N$ is in $I_2$ and divisible by $q$, for perturbation factor $\ell$ chosen by the oracle uniformly randomly from $[k \cdot q \cdot 2^{2\tau+\kappa}]$. Otherwise, $\mathcal{O}_{C_i}$ returns 0 and permanently aborts.
7. The output of the experiment is 1 if and only if $\mathcal{A}$ outputs $b' = b$.

The Paillier-EC-Refresh assumption is that, for every PPT algorithm $\mathcal{A}$, there is a negligible function $\mu$ such that $\Pr[\mathsf{Expt}_{\mathcal{A}}(1^\kappa) = 1] \leq \frac{1}{2} + \mu(\kappa)$.
We have characterized Paillier-EC-Refresh as specifying the number of oracles $k$ up front, but $k$ can be arbitrary and indefinite, and oracles can be spun up dynamically.

## 7.2 Security Proof

**Theorem 1.** *Under the Paillier-EC-Refresh assumption, for any admissible, mobile PPT adversary $\mathcal{A}$ with auxiliary input $z$, attacking the $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com-zk}}, \mathcal{F}_{\mathsf{coin}})$-hybrid model protocol $\pi$, there exists an ideal world adversary $\mathcal{S}$ such that $\mathsf{Hyb}_{\pi,\mathcal{A}}(1^\kappa, z) \stackrel{c}{=} \mathsf{Ideal}_{\mathcal{F}_{\mathsf{TS}},\mathcal{S}}(1^\kappa, z)$.*

*Proof.* We proceed to describe the simulator $\mathcal{S}$ that produces a simulated transcript for keygen, and, subsequently, for some polynomial number of executions of sign and refresh. Throughout, the adversary may change which of the two parties it chooses to corrupt, as long as the corruption pattern is *admissible* (Section 2.3); the simulator will shift to produce the view of the appropriate party, as described below. Upon a new corruption of party $P_b$, $\mathcal{S}$ also generates a simulated state for that party, $\mathsf{state}_b$, and provides it to the adversary.

*KeyGen* (Malicious $P_1$.)

- On input $(\mathsf{KeyGen}, \mathbb{G}, g, q)$, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{TS}}$ and invokes $\mathcal{A}$ with the same input. $\mathcal{S}$ receives $X$ from $\mathcal{F}_{\mathsf{TS}}$.
- $\mathcal{S}$ sends $\mathsf{Proof\text{-}receipt}$, to $\mathcal{A}$ as the first message from $P_2$.
- $\mathcal{S}$ receives from $\mathcal{A}$:
  - $(\mathsf{prove}, X_1, x_1)$ as $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$. If $X_1 \neq (x_1 \times G)$, $\mathcal{S}$ terminates the execution.
  - $(C, N, \pi, \Psi)$. If $\Pi_{\mathsf{GCD}}.\mathsf{Ver}(N, \pi) = 0$ or $\Pi_{\mathsf{eq}}.\mathsf{Ver}(C, X_1, \Psi) = 0$, $\mathcal{S}$ terminates the execution.
- $\mathcal{S}$ computes $\widetilde{X}_2 = X + (-x_1 \times G)$, and sends $(\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{X}_2)$ to $P_1$.

- $\mathcal{S}$ sets $\tilde{x}_1 = x_1$, and $\widetilde{N} = N$. It stores $(X, \tilde{x}_1, \widetilde{X}_2, \widetilde{N}, C)$ for future signature and refresh simulations and instructs $\mathcal{F}_{\mathsf{TS}}$ to provide output to honest $P_2$.

*KeyGen* (Malicious $P_2$.)

- On input $(\mathsf{KeyGen}, \mathbb{G}, g, q)$, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{TS}}$ and invokes $\mathcal{A}$ with the same input. $\mathcal{S}$ receives $X$ from $\mathcal{F}_{\mathsf{TS}}$.
- $\mathcal{S}$ receives $(\mathsf{proof\text{-}commit}, X_2, x_2)$ as $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$.
  If $X_2 \neq (x_2 \times G)$, $\mathcal{S}$ terminates the execution.
- To simulate $P_1$'s message,
  - $\mathcal{S}$ computes $\widetilde{X}_1 = X + (-x_2 \times G)$.
  - $\mathcal{S}$ honestly samples $\widetilde{N} = \widetilde{P}\widetilde{Q}$, and generates a simulated proof $\tilde{\pi} \leftarrow \mathcal{S}_{\mathsf{GCD}}(\widetilde{N})$.
  - $\mathcal{S}$ samples $\tilde{x}_1 \leftarrow [q]$, $t \leftarrow [2^{\tau+2\kappa}]$, and computes $\widetilde{C} \leftarrow \mathsf{Enc}_{\widetilde{N}}(\tilde{x}_1 + tq)$.
  - $\mathcal{S}$ samples $\widetilde{\Psi} \leftarrow \mathcal{S}_{\mathsf{eq}}(\widetilde{N}, \widetilde{C}, \widetilde{X}_1)$ (Section 3).

  $\mathcal{S}$ sends $(\widetilde{C}, \widetilde{N}, \tilde{\pi}, \widetilde{\Psi})$ to $P_2$ to simulate $P_1$'s message, and sends $(\mathsf{ssid}_2, \mathsf{Proof}, \widetilde{X}_1)$ to $P_2$ to simulate the output of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$.
- If $\mathcal{S}$ does not receive $\mathsf{decom\text{-}proof}$, it aborts.
- $\mathcal{S}$ sets $\tilde{x}_1 = x_1$, and $\widetilde{X}_2 = X_2$. It stores $(X, \tilde{x}_1, \widetilde{X}_2, \widetilde{P}, \widetilde{Q})$ for future signature and refresh simulations and instructs $\mathcal{F}_{\mathsf{TS}}$ to provide output to honest $P_1$.

*Sign.* (Malicious $P_1$.)

- If this is a new corruption, $\mathcal{S}$ samples random $\tilde{x}_1, \widetilde{P}, \widetilde{Q}$, computes $\widetilde{X}_1 = \tilde{x}_1 \times G$, $\widetilde{X}_2 = X + (-\tilde{x}_1 \times G)$ and $\widetilde{N} = \widetilde{P}\widetilde{Q}$. (Recall, $X$ was stored previously by $\mathcal{S}$.) $\mathcal{S}$ uses these values to simulate the state of the malicious $P_1$, sending $\mathsf{state_1} = (X, \tilde{x}_1, \widetilde{X}_2, \widetilde{N}, \widetilde{P}, \widetilde{Q})$ to $\mathcal{A}$.
- On input $(\mathsf{ssid}, \mathsf{Sign}, m)$, $\mathcal{S}$ sends $m$ to $\mathcal{F}_{\mathsf{TS}}$, and invokes $\mathcal{A}$ with the same input. $\mathcal{S}$ receives signature $(r, s)$ from $\mathcal{F}_{\mathsf{TS}}$.
- $\mathcal{S}$ sends $(\mathsf{ssid}, \mathsf{Proof\text{-}receipt})$ to $\mathcal{A}$ as the output from $\mathcal{F}_{\mathsf{com-zk}}$.
- $\mathcal{S}$ receives $(\mathsf{ssid}, \mathsf{prove}, K_1, k_1)$ and $(\mathsf{ssid}, \mathsf{prove}, X_1, x_1)$ as input to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$. If $K_1 \neq (k_1 \times G)$, $\tilde{x}_1 \neq x_1$, or $X_1 \neq (x_1 \times G)$, $\mathcal{S}$ aborts the execution.
- Let $y = \hat{x}_1 - x_1 \bmod N$ denote the difference in the Paillier plaintext corresponding to $C$, sent by $\mathcal{A}$ either during key generation or during the last key refresh, and the discrete log of $X_1$. (Note, $\hat{x}_1$ is unknown to $\mathcal{S}$.)
  - $\mathcal{S}$ samples $\hat{k}_2 \leftarrow [q]$ and $\rho \leftarrow [3q^2 2^{3\tau+2\kappa}]$, and homomorphically computes $\widetilde{C}' = \big((k_1 s \bmod q) + \hat{k}_2 r y + \rho q\big) \bmod N$.
  - $\mathcal{S}$ computes $K = k \times G$ from $(r, s)$ exactly as is done during signature verification: $K = \frac{(r \times X) + (m \times G)}{s}$. It then sets $\widetilde{K}_2 = \frac{K}{k_1}$.

  $\mathcal{S}$ sends $\big((\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{X}_2), (\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{K}_2), \widetilde{C}'\big)$ to $\mathcal{A}$.

- $\mathcal{S}$ continues to store $(X, \tilde{x}_1, \widetilde{X}_2, \widetilde{N}, C)$ for future signature and refresh simulations and instructs $\mathcal{F}_{\mathsf{TS}}$ to provide $(r, s)$ to honest $P_2$.

*Sign.* (Malicious $P_2$)

- If this is a new corruption, $\mathcal{S}$ samples random $\tilde{x}_1, \tilde{x}_2 \leftarrow [q]$, and $\widetilde{P}, \widetilde{Q}$. It computes $\widetilde{N} = \widetilde{P}\widetilde{Q}$, and samples $\widetilde{C} \leftarrow \mathsf{Enc}_{\widetilde{N}}(\tilde{x}_1)$. $\mathcal{S}$ computes $\widetilde{X}_2 = \tilde{x}_2 \times G$, and $\widetilde{X}_1 = X - (\tilde{x}_2 \times G)$. (Recall, $X$ was stored previously by $\mathcal{S}$.) $\mathcal{S}$ uses these values to simulate the state of the malicious $P_2$, sending $\mathsf{state}_2 = (X, \tilde{x}_2, \widetilde{X}_1, \widetilde{N}, \widetilde{C})$ to $\mathcal{A}$.
- On input $(\mathsf{ssid}, \mathsf{Sign}, m)$, $\mathcal{S}$ sends $m$ to $\mathcal{F}_{\mathsf{TS}}$, and invokes $\mathcal{A}$ with the same input. $\mathcal{S}$ receives signature $(r, s)$ from $\mathcal{F}_{\mathsf{TS}}$.
- $\mathcal{S}$ receives $(\mathsf{ssid}||1, \mathsf{proof\text{-}commit}, X_2, x_2)$ and $(\mathsf{ssid}||2, \mathsf{proof\text{-}commit}, K_2, k_2)$ as $\mathcal{A}$ invokes $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$. If $K_2 \neq (k_2 \times G)$, $\tilde{x}_2 \neq x_2$, or $X_2 \neq (x_2 \times G)$, $\mathcal{S}$ aborts the simulation.
- $\mathcal{S}$ computes $K = k \times G$ from $(r, s)$ exactly as is done during signature verification: $K = \frac{(r \times X) + (m \times G)}{s}$. It then sets $\widetilde{K}_1 = \frac{K}{k_2}$. It sends $(\mathsf{ssid}||3, \mathsf{Proof}, \widetilde{K}_1)$, and $(\mathsf{ssid}||4, \mathsf{Proof}, \widetilde{X}_1)$ to simulate the outputs of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$.
- $\mathcal{S}$ receives $(\mathsf{ssid}||1, \mathsf{decom\text{-}proof})$, $(\mathsf{ssid}||2, \mathsf{decom\text{-}proof})$ and $C'$ from $\mathcal{A}$. It uses $\widetilde{P}, \widetilde{Q}$ to compute $s_2 = \mathsf{Dec}(C')$ ; $s = k_1^{-1} s_2 \bmod q$ ; $s = \min(s, q - s)$.
  $\mathcal{S}$ verifies that $s$ is a valid signature on $m$. If not, it refuses additional sign commands until a refresh is executed.
- $\mathcal{S}$ continues to store $(X, \tilde{x}_2, \widetilde{X}_1, \widetilde{P}, \widetilde{Q})$ for future signature and refresh simulations and instructs $\mathcal{F}_{\mathsf{TS}}$ to provide $(r, s)$ to honest $P_1$.

*Refresh.* (Malicious $P_1$)

- If this is a new corruption, $\mathcal{S}$ samples random $\tilde{x}_1, \widetilde{P}, \widetilde{Q}$, computes $\widetilde{X}_1 = (\tilde{x}_1 \times G)$, $\widetilde{X}_2 = X - (\tilde{x}_1 \times G)$ and $\widetilde{N} = \widetilde{P}\widetilde{Q}$. (Recall, $X$ was stored previously by $\mathcal{S}$.) $\mathcal{S}$ uses these values to simulate the state of the malicious $P_1$, sending $\mathsf{state}_1 = (X, \tilde{x}_1, \widetilde{X}_2, \widetilde{N}, \widetilde{P}, \widetilde{Q})$ to $\mathcal{A}$. It stores $\tilde{x}_1, \widetilde{P}, \widetilde{Q}$.
- On input $(\mathsf{ssid}, \mathsf{Refresh})$, $\mathcal{S}$ invokes $\mathcal{A}$ with the same input.
- $\mathcal{S}$ samples $r \leftarrow [q]$ and sends it to $\mathcal{A}$ in simulation of the output of $\mathcal{F}_{\mathsf{coin}}$.
  It updates $\tilde{x}_1 = \tilde{x}_1 - r$, and $\widetilde{X}_1 = (\tilde{x}_1 \times G)$.
- $\mathcal{S}$ receives $(C, N, \pi, \Psi)$ from $\mathcal{A}$. If $\Pi_{\mathsf{GCD}}.\mathsf{Ver}(N, \pi) = 0$ or $\Pi_{\mathsf{eq}}.\mathsf{Ver}(C, X_1, \Psi) = 0$, $\mathcal{S}$ terminates the execution.
- $\mathcal{S}$ stores $(X, \tilde{x}_1, \widetilde{X}_2, \widetilde{N}, C)$ for future signature and refresh simulations.

*Refresh.* (Malicious $P_2$)

- If this is a new corruption, $\mathcal{S}$ samples random $\tilde{x}_1, \tilde{x}_2, \widetilde{N}$, and computes $\widetilde{X}_1 = X - (\tilde{x}_2 \times G)$. (Recall, $X$ was stored previously by $\mathcal{S}$.) It then samples $\widetilde{C} \leftarrow \mathsf{Enc}_{\widetilde{N}}(\tilde{x}_1)$, and uses these values to simulate the state of the malicious $P_2$, sending $\mathsf{state}_2 = (X, \tilde{x}_2, \widetilde{X}_2, \widetilde{N}, \widetilde{C})$ to $\mathcal{A}$.

- $\mathcal{S}$ samples $r \leftarrow [q]$ and sends it to $\mathcal{A}$ in simulation of the output of $\mathcal{F}_{\mathsf{coin}}$.
  It updates $\tilde{x}_2 = \tilde{x}_2 + r$, and $\widetilde{X}_2 = (\tilde{x}_2 \times G)$.
- To simulate $P_1$'s message,
  - $\mathcal{S}$ honestly samples $\widetilde{N} = \widetilde{P}\widetilde{Q}$, and $\tilde{\pi} \leftarrow \mathcal{S}_{\mathsf{GCD}}(\widetilde{N})$.
  - $\mathcal{S}$ samples $\tilde{x}_1 \leftarrow [q]$, $t \leftarrow [2^{\tau+2\kappa}]$, and $\widetilde{C} \leftarrow \mathsf{Enc}_{\widetilde{N}}(\tilde{x}_1 + tq)$.
  - $\mathcal{S}$ samples $\widetilde{\Psi} \leftarrow \mathcal{S}_{\mathsf{eq}}(\widetilde{N}, \widetilde{C}, \widetilde{X}_1)$ (Section 3).

  $\mathcal{S}$ sends $(\widetilde{C}, \widetilde{N}, \tilde{\pi}, \widetilde{\Psi})$ to $P_2$.
- $\mathcal{S}$ stores $(X, \tilde{x}_2, \widetilde{X}_1)$ for future signature and refresh simulations.

To prove Theorem 1, we have to prove that the two join distributions, $\mathsf{Hyb}_{\pi,\mathcal{A}}(1^\kappa, z) = (\mathsf{view}_{\mathsf{Hyb}}, \mathsf{out}_{\mathsf{Hyb}})$ and $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{TS}},\mathcal{S}}(1^\kappa, z) = (\mathsf{view}_{\mathsf{Ideal}}, \mathsf{out}_{\mathsf{Ideal}})$ are indistinguishable. We first look at the adversarial views before considering the joint distribution.

**Claim 3.** If $\mathcal{A}$ is an admissible adversary, then $\mathsf{view}_{\mathsf{Hyb}} \stackrel{c}{=} \mathsf{view}_{\mathsf{Ideal}}$.

*Proof.* To simplify the presented argument that the simulated view is indistinguishable from a hybrid-world view, we rewrite here, more concisely, the messages generated by $\mathcal{S}$ in the simulated view of the adversary. We use $\mathsf{KG_b}$, $\mathsf{Sig_b}$ and $\mathsf{Rfsh_b}$ to denote the view of party $b$ in the corresponding protocol.

$\mathsf{KG}_1 = \big(\mathsf{Proof\text{-}receipt}, (\mathsf{ssid}_1, \mathsf{Decom\text{-}proof}, \widetilde{X}_2)\big)$
$\mathsf{KG}_2 = \big((\widetilde{C}, \widetilde{N}, \tilde{\pi}, \tilde{\Psi}), (\mathsf{ssid}_2, \mathsf{Proof}, \widetilde{X}_1)\big)$
$\mathsf{Sig}_1 = \big(\mathsf{state}_1, \mathsf{Proof\text{-}receipt}, (\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{X}_2), (\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{K}_2), \widetilde{C}'\big)$
$\mathsf{Sig}_2 = (\mathsf{state}_2, (\mathsf{ssid}||3, \mathsf{Proof}, \widetilde{K}_1), (\mathsf{ssid}||4, \mathsf{Proof}, \widetilde{X}_1)$
$\mathsf{Rfsh}_1 = (\mathsf{state}_1, r)$
$\mathsf{Rfsh}_2 = \big(\mathsf{state}_2, r, (\widetilde{C}, \widetilde{N}, \tilde{\pi}, \tilde{\Psi})\big)$

Recall, $\mathsf{state}_1 = (X, \tilde{x}_1, \widetilde{X}_2, \widetilde{N}, \widetilde{P}, \widetilde{Q})$, $\mathsf{state}_2 = (X, \tilde{x}_2, \widetilde{X}_1, \widetilde{N}, \widetilde{C})$, and each is only included when there is a new corruption. We make a few simple observations before proceeding. First, in the $(\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}, \mathcal{F}_{\mathsf{zk}})$-hybrid model, the simulation of $\mathsf{Proof\text{-}receipt}$ as output from $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$ in $\mathsf{KG}_1$ and $\mathsf{Sig}_1$ is perfect. Although $\mathcal{S}$ does not know $\tilde{x}_2$, the simulation of $(\mathsf{ssid}_1, \mathsf{Decom\text{-}proof}, \widetilde{X}_2)$ as output from $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$ in $\mathsf{KG}_1$ is also perfect. This is because, after extracting $x_1$ from $\mathcal{A}$, the value $\widetilde{X}_2 = X - (x_1 \times G)$ is consistent with the (unknown) $x_2$. The same argument holds for $(\mathsf{ssid}_2, \mathsf{proof}, \widetilde{X}_1)$ in $\mathsf{KG}_2$, $\widetilde{X}_2$ in $\mathsf{state}_1$, $\widetilde{X}_1$ in $\mathsf{state}_2$, and $(\mathsf{ssid}||3, \mathsf{Proof}, \widetilde{K}_1), (\mathsf{ssid}||4, \mathsf{Proof}, \widetilde{X}_1)$ in $\mathsf{Sig}_2$. Finally, because we have an admissible adversary, at the time of any new corruption of $P_1$, the only thing known about $\mathsf{state}_1$ is $X$. Therefore, the marginal distribution of the simulated $\mathsf{state}_1$ is identical to that of the real world state.
Putting these observations together, we now re-write the simulated variables, marking those with the correct distributions in green, and the ones that we need to address in red. We will explain the variables marked in red as we address them.

$\mathsf{KG}_1 = \big(\mathsf{Proof\text{-}receipt}, (\mathsf{ssid}_1, \mathsf{Decom\text{-}proof}, \widetilde{X}_2)\big)$
$\mathsf{KG}_2 = \big((\widetilde{C}, \widetilde{N}, \tilde{\pi}, \tilde{\Psi}), (\mathsf{ssid}_2, \mathsf{Proof}, \widetilde{X}_1)\big)$
$\mathsf{Sig}_1 = \big(\mathsf{state}_1, \mathsf{Proof\text{-}receipt}, (\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{X}_2), (\mathsf{ssid}, \mathsf{Decom\text{-}proof}, \widetilde{K}_2), \widetilde{C}'\big)$

$\mathsf{Sig}_2 = (\mathsf{state}_2, (\mathsf{ssid}||3, \mathsf{Proof}, \widetilde{K}_1), (\mathsf{ssid}||4, \mathsf{Proof}, \widetilde{X}_1)$
$\mathsf{Rfsh}_1 = (\mathsf{state}_1, r)$
$\mathsf{Rfsh}_2 = \big(\mathsf{state}_2, r, (\widetilde{C}, \widetilde{N}, \tilde{\pi}, \tilde{\Psi})\big)$

**Hybrid $\mathsf{H}_1$:** In this hybrid step, if $P_2$ is malicious during key generation, we replace $\widetilde{C}$ in key generation, which encrypts a random $\tilde{x}_1 \in [q]$, for $C$ that encrypts the discrete log of $X_1$. (If $P_2$ is not malicious during key generation, we do nothing, and Hybrid $\mathsf{H}_1$ is identical to the distribution of views generated by $\mathcal{S}$.)

**Claim 4.** If the distribution of views generated in Hybrid $\mathsf{H}_1$ is distinguishable from that generated by $\mathcal{S}$ in $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{TS}},\mathcal{S}}$, then there exists an adversary $\mathcal{R}$ that breaks the Paillier-EC-Refresh assumption.

*Proof.* $\mathcal{R}$ receives (first) challenge $N_1^*, C_1^*, X^*$, and plays the part of $\mathcal{S}$, interacting with $\mathcal{A}$. After $\mathcal{R}$ receives $(\mathsf{proof\text{-}commit}, X_2, x_2)$ from $\mathcal{A}$ in the first message of key generation, it sets $X = X^* + (x_2 \times G)$. It uses $C_1^*$ and $N_1^*$ to construct $P_1$'s message in key generation, creating simulated proofs $\tilde{\pi}$ and $\tilde{\Psi}$: $(C_1^*, N_1^*, \tilde{\pi}, \widetilde{\Psi})$. In the $i$th refresh command prior to the first de-corrupt command, $\mathcal{R}$ queries its challenger and receives $N_i^*, C_i^*$. It updates $X^* = X^* + (r \times G)$, where $r \leftarrow [q]$ is the simulated output of $\mathcal{F}_{\mathsf{coin}}$ created by $\mathcal{R}$. It simulates $P_1$'s message using $(N_i^*, C_i^*, \tilde{\pi}, \widetilde{\Psi})$. In any sign commands prior to the next de-corrupt command, $\mathcal{R}$ verifies the correctness of $C'$ sent by $P_2$ by using the current instance of the Paillier-EC-Refresh oracle: it extracts $k_2$ and $x_2$ from $P_2$'s calls to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$, sets $\alpha = k_2^{-1} m \bmod q$ and $\beta = k_2^{-1} r x_2 \bmod q$, and submits query $(C', \alpha, \beta)$ to its oracle $\mathcal{O}_{C_{i^*}}$. For the remainder of the simulation, $\mathcal{R}$ behaves exactly as $\mathcal{S}$; in particular, during subsequent refresh operations where $P_2$ is malicious, and during new corruptions of $P_2$ during signing and refresh, it samples a new, random $\tilde{x}_1$ and sets $\widetilde{C} = \mathsf{Enc}_{\widetilde{N}}(\tilde{x}_1)$. Just as $\mathcal{S}$ does, it uses its knowledge of the simulated $\widetilde{N} = \widetilde{P}\widetilde{Q}$ to decrypt $C'$ and verify validity of the plaintext during any signing executions. If the challenge bit $b$ in the Paillier-EC-Refresh game is 1, then $\mathcal{R}$ produces the same distribution as $\mathcal{S}$. Otherwise, it produces that of Hybrid $\mathsf{H}_1$. □

**Hybrid $\mathsf{H}_2^{(i)}$:** In this sequence of hybrid steps, we replace $\widetilde{C}$, which encrypts a random $\tilde{x}_1 \in [q]$, for $C$ that encrypts the discrete log of $X_1$. This is needed in every execution of refresh for which $P_2$ is malicious, and upon every new corruption of $P_2$ during refresh or signing, when $\mathsf{state}_2$ is simulated. We proceed through these events in order of their occurrence, defining $\mathsf{H}_2^{(i)}$ as the distribution in which the first $i-1$ instances of these events use $C = \mathsf{Enc}_{\widetilde{N}}(x_i)$, and the remaining events use $\widetilde{C} = \mathsf{Enc}_{\widetilde{N}}(\tilde{x}_i)$.

**Claim 5.** If the distribution of views generated in Hybrid $\mathsf{H}_2^{(i)}$ is distinguishable from that generated in $\mathsf{H}_2^{(i-1)}$, then there exists an adversary $\mathcal{R}$ that breaks the Paillier-EC-Refresh assumption.

*Proof.* $\mathcal{R}$ queries the challenger to receive challenge $N_1^*, C_1^*, X^*$, and plays the part of $\mathcal{S}$, interacting with $\mathcal{A}$. $\mathcal{R}$ simulates key generation by running the honest protocol. It stores $X, x_1$ for use in what follows.
Until the $i$th event, when $P_2$ is malicious during refresh, and each time there is a new corruption of $P_2$, $\mathcal{R}$ uses its knowledge of $x_1$ (which might be updated during refresh procedures) to run the protocol honestly. In particular, it constructs $C = \mathsf{Enc}_N(x_1)$ honestly. Note that it still uses simulated proofs $\tilde{\pi}$ and $\widetilde{\Psi}$.

In the $i$th event, $\mathcal{R}$ uses its challenge to construct $(C_1^*, N_1^*, \tilde{\pi}, \widetilde{\Psi})$, using simulated proofs $\tilde{\pi} \leftarrow \mathcal{S}_{\mathsf{GCD}}(N^*)$ and $\widetilde{\Psi} \leftarrow \mathcal{S}_{\mathsf{eq}}(N_1^*, C_1^*, X^*)$; this is done whether the $i$th event is an execution of refresh with malicious $P_2$, or a new corruption of $P_2$ during refresh or sign. In the latter case, $\mathcal{R}$ uses $X^*$ when simulating $X_1$ in $\mathsf{state}_2$.
In the $i$th refresh command prior to the next de-corrupt command, $\mathcal{R}$ queries its challenger and receives $N_i^*, C_i^*$. It updates $X^* = X^* + (r \times G)$, where $r \leftarrow [q]$ is the simulated output of $\mathcal{F}_{\mathsf{coin}}$ created by $\mathcal{R}$. It simulates $P_1$'s message using $(N_i^*, C_i^*, \tilde{\pi}, \widetilde{\Psi})$. In any sign commands prior to the next de-corrupt command, $\mathcal{R}$ verifies the correctness of $C'$ sent by $P_2$ by using the current instance of the Paillier-EC-Refresh oracle: it extracts $k_2$ and $x_2$ from $P_2$'s calls to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$, sets $\alpha = k_2^{-1} m \bmod q$ and $\beta = k_2^{-1} r x_2 \bmod q$, and submits query $(C', \alpha, \beta)$ to its oracle $\mathcal{O}_{C_{i^*}}$.
In the remaining events, $\mathcal{R}$ proceeds as $\mathcal{S}$ does (and as $\mathcal{R}$ did in Hybrid $\mathsf{H}_1$). If the challenge bit $b$ in the Paillier-EC-Refresh game is 1, then $\mathcal{R}$ produces the same distribution as in Hybrid $\mathsf{H}_2^{(i-1)}$. Otherwise, it produces that of Hybrid $\mathsf{H}_2^{(i)}$. □

**Hybrids $\mathsf{H}_3^{(i)}$ and $\mathsf{H}_4^{(i)}$:** In these steps we replace the simulated proofs, $\tilde{\pi}$ and $\widetilde{\Psi}$, first in key generation, and then in the same sequence of events defining $\mathsf{H}_2^{(i)}$: when $P_2$ is malicious during key refresh, and upon a new corruption of $P_2$ during refresh or sign. More specifically, in $\mathsf{H}_3^{(i)}$, we use an honestly generated $\pi \leftarrow \Pi_{\mathsf{GCD}}.\mathsf{Prv}()$ in key generation, and in the first $i$ events in the above sequence of events; we use $\tilde{\pi}$ in the $i+1$st event, and in every one after. For simplicity, we only prove that $\mathsf{H}_3^{(i)}$ is indistinguishable from $\mathsf{H}_3^{(i-1)}$. The proof for $\mathsf{H}_4^{(i)}$ is nearly identical.

**Claim 6.** If the distribution of views generated in Hybrid $\mathsf{H}_3^{(i)}$ is distinguishable from that generated in $\mathsf{H}_3^{(i-1)}$, then there exists an adversary $\mathcal{R}$ that breaks zero knowledge of $\Pi_{\mathsf{GCD}}$.

*Proof.* $\mathcal{R}$ plays the role of $P_1$ honestly through key generation and the first $i$ events. In the $i$th event, it receives a challenge instance $N^*$, and a challenge proof, $\pi^*$. Additionally, we assume that $\mathcal{R}$ is given auxiliary information containing the witness $\phi(N)$.[§] It embeds $\pi$ in the $i$th event, using $(C, N^*, \pi^*, \widetilde{\Psi})$. For each signing operation prior to event $i+1$, $\mathcal{R}$ uses auxiliary information $\phi(N)$ to decrypt $C'$ and verify correctness of the plaintext, just as $P_1$ and $\mathcal{S}$ would do. If it fails to verify, $\mathcal{R}$ refuses to perform additional signatures until another refresh occurs. (When making the same claim for substituting $\widetilde{\Psi}$ for $\Psi$ in $\mathsf{H}_4^{(i)}$, note that $\phi(N)$ is not needed, but otherwise the reduction is identical.) □

After this set of changes, we note that $\mathsf{state}_2 = (X, \tilde{x}_2, \widetilde{X}_1, \widetilde{C}, \widetilde{N})$ have the correct marginal distribution. As with $\mathsf{state}_1$, this holds because $\mathcal{A}$ is admissible, and, therefore, no prior information about these uniform values is known.

**Hybrid $\mathsf{H}_5$:** Here we replace $\widetilde{C}'$ with $C'$ when $P_1$ is malicious during signing. Recall that when $\mathcal{S}$ creates the message containing $\widetilde{C}'$ on behalf of $P_2$, $\mathcal{S}$ does not know $x_2$ or $k_2$. Nevertheless, we claim that this change in the distribution has small statistical distance.

**Claim 7.** Hybrid $\mathsf{H}_5$ is statistically indistinguishable from the last of Hybrids $\mathsf{H}_4^{(i)}$.

*Proof.* The claim is a bit challenging to prove for the following reasons:

[§]Note that zero knowledge is expected to hold even when arbitrary auxiliary information is provided to the distinguisher, including full knowledge of the witness itself.

1. Recall, $P_1$ is NOT forced to prove that $N$ is the product of two large primes. It is forced to prove that $GCD(N, \phi(N)) = 1$, and as part of that proof it proves that $GCD(N, \alpha) = 1$, where $\alpha$ is the product of small primes up to some bound. But, otherwise, we allow $P_1$ to take $N$ to be a product of (possibly several) smallish primes above that bound.

2. If $P_1$ chooses $N$ maliciously, or even if it doesn't, it can encrypt a value of $\hat{x}_1 \in [N]$ that does NOT satisfy $\hat{x}_1 = x_1 \bmod q$, and its ZK proof about the Paillier encryption of $\hat{x}_1$ may nonetheless verify with non-negligible probability. We allow this. But we also show that, if the probability is non-negligible, then $\hat{x}_1$ must have some well-defined relationship with $x_1$. Using that relationship, $\mathcal{S}$ is able to mimic $P_2$'s distribution (without knowing $x_2$).

In the simulation, $\mathcal{S}$ obtains a signature $s = k^{-1}(m+rx) \bmod q$ from the challenger. Via extraction, $\mathcal{S}$ obtains $x_1$ and $k_1$ as integers in $[q]$. Finally, $\mathcal{S}$ obtains an encryption of $\hat{x}_1 \in [N]$ from $P_1$. Recall $y = \hat{x}_1 - x_1$. Using Paillier's homomorphism, $\mathcal{S}$ obtains a Paillier encryption of $y$. $\mathcal{S}$ generates $\hat{k}_2$ and $\rho$ from appropriate ranges and then produces a Paillier ciphertext that encrypts $\hat{s} = \big((k_1 s \bmod q) + \hat{k}_2 ry + \rho q\big) \bmod N$. Because $\mathsf{GCD}(N, \phi(N)) = 1$, the ciphertext does not leak any information about the homomorphic evaluation beyond the plaintext, so it suffices to analyze the content of the plaintexts.
Rewriting $k = k_1 k_2 \bmod q$ and $x = x_1 + x_2 \bmod q$, where $k_1$ and $x_1$ were both extracted from $\mathcal{A}$'s messages, we have:

$$\hat{s} = \big((k_2^{-1}(m + rx_1 + rx_2) \bmod q) + \hat{k}_2 ry + \rho q\big) \bmod N.$$

In the true execution, $k_2^{-1} \leftarrow [q]$, $\tilde{\rho} \leftarrow [q]$, and $\tilde{k}_2^{-1} = k_2^{-1} + \tilde{\rho} q$. We have:

$$\begin{aligned}
s &= \big((\tilde{k}_2^{-1}(m + r\hat{x}_1 + rx_2) + \rho q\big) \bmod N \\
&= (\tilde{k}_2^{-1}(m + r(x_1 + y) + rx_2) + \rho q\big) \bmod N \\
&= (\tilde{k}_2^{-1}(m + rx_1 + rx_2) + \tilde{k}_2^{-1} ry + \rho q) \bmod N \\
&\overset{s}{\approx} \big((\tilde{k}_2^{-1}(m + rx_1 + rx_2) \bmod q) + \tilde{k}_2^{-1} ry + \rho q\big) \bmod N \\
&= \big((k_2^{-1}(m + rx_1 + rx_2) \bmod q) + \tilde{k}_2^{-1} ry + \rho q\big) \bmod N
\end{aligned}$$

The second-to-last equivalence follows from the statistical equivalence of $\tilde{k}_2^{-1}(m + rx_1 + rx_2) + \rho q$ and $(\tilde{k}_2^{-1}(m + rx_1 + rx_2) \bmod q) + \rho q$ over $\mathbb{Z}$, as shown in Claim 8; as they are equivalent modulo $q$ and as $\rho q$ translates by a random multiple of $q$ that is much larger in expected magnitude than $\tilde{k}_2^{-1}(m + rx_1 + rx_2)$. The last equality holds because $\tilde{k}_2^{-1} \equiv k_2^{-1} \bmod q$.

So, it remains to show the following distributions are statistically close:

$$\begin{aligned}
&((k_2^{-1}(m + rx_1 + rx_2) \bmod q) + \hat{k}_2 ry + \rho q) \bmod N \\
\overset{s}{\approx}&((k_2^{-1}(m + rx_1 + rx_2) \bmod q) + \tilde{k}_2^{-1} ry + \rho q) \bmod N,
\end{aligned} \tag{1}$$

where the randomness is over $\hat{k}_2, \tilde{k}_2^{-1}$ and $\rho$. Note that on the right hand side, $\tilde{k}_2^{-1} = k_2^{-1} + \tilde{\rho} q$, so there is a correlation between the first and second terms, while on the left hand side, $\hat{k}_2$ was sampled independently from $k_2^{-1}$ by the simulator.

Yet, as we will show, the fact that $\tilde{k}_2^{-1}$ was generated as $k_2^{-1} + \tilde{\rho}q$ for random $\tilde{\rho}$ will "disrupt" this correlation sufficiently that it allows the argument of a well-distributed simulation.
By the soundness guarantee (Claim 1), assuming $P_1$'s ZK proof about its encryption of $\hat{x}_1$ verifies with non-negligible probably, there exists a small integer $\delta$, relatively prime to $q$, such that $\delta y = c + \ell N$ for some $c \in [3q^2 2^{2(\tau+\kappa)}]$, with $c \equiv 0 \bmod q$. Since we want to use this fact about $\delta y$, it is convenient to multiply the distributions we are comparing by $\delta$ – in particular, we will prove Equation 1 by proving the following equivalent statement:

$$\begin{aligned}&(\delta(k_2^{-1}(m + rx_1 + rx_2) \bmod q) + \delta\hat{k}_2 ry + \delta\rho q) \bmod \delta N\\ \overset{s}{\approx}&(\delta(k_2^{-1}(m + rx_1 + rx_2) \bmod q) + \delta\tilde{k}_2^{-1} ry + \delta\rho q) \bmod \delta N.\end{aligned} \tag{2}$$

The two statistical claims are equivalent, because multiplying or dividing by $\delta$ to go from one statement to another is a bijection here. For convenience, let $m' = m + rx_1 + rx_2$. We have:

$$\begin{aligned}&(\delta(k_2^{-1}m' \bmod q) + \delta\tilde{k}_2^{-1} ry + \delta\rho q) \bmod \delta N\\ &= ((\delta k_2^{-1}m' \bmod \delta q) + \tilde{k}_2^{-1} rc + \tilde{k}_2^{-1} r\ell N + \delta\rho q) \bmod \delta N\\ &\overset{s}{\approx} ((\delta k_2^{-1}m' + \tilde{k}_2^{-1} rc \bmod \delta q) + \tilde{k}_2^{-1} r\ell N + \delta\rho q) \bmod \delta N\\ &\overset{s}{\approx} ((\delta k_2^{-1}m' + (\tilde{k}_2^{-1} \bmod \delta) rc \bmod \delta q) + \tilde{k}_2^{-1} r\ell N + \delta\rho q) \bmod \delta N\\ &= ((\delta k_2^{-1}m' + (\tilde{k}_2^{-1} \bmod \delta) rc \bmod \delta q) +\\ &\qquad\qquad (\tilde{k}_2^{-1} r\ell \bmod \delta) N + \delta\rho q) \bmod \delta N\end{aligned}$$

In the third line, after observing that $\rho \cdot \delta q \gg \tilde{k}_2^{-1} rc$ (recall, $\rho \in [3q^2 2^{3\tau+2\kappa}]$ while $\tilde{k}_2^{-1} rc \in [3q^4 2^{2(\tau+\kappa)}]$), and (trivially) $\tilde{k}_2^{-1} rc \equiv [\tilde{k}_2^{-1} rc]_{\delta q} \bmod \delta q$, it then follows from Claim 8 that $\tilde{k}_2^{-1} rc + \rho\delta q \overset{s}{\approx} \tilde{k}_2^{-1} rc \bmod \delta q + \rho\delta q$. The claim of statistical closeness in the fourth line follows from the fact that $c$ is divisible by $q$ (see Claim 1). The one-to-one correspondence in the fifth line follows from the Chinese remainder theorem.
We also have:

$$\begin{aligned}&(\delta(k_2^{-1}m' \bmod q) + \delta\hat{k}_2 ry + \delta\rho q) \bmod \delta N\\ &= ((\delta k_2^{-1}m' \bmod \delta q) + \hat{k}_2 rc + \hat{k}_2 r\ell N + \delta\rho q) \bmod \delta N\\ &\overset{s}{\approx} ((\delta k_2^{-1}m' + \hat{k}_2 rc \bmod \delta q) + \hat{k}_2 r\ell N + \delta\rho q) \bmod \delta N\\ &\overset{s}{\approx} ((\delta k_2^{-1}m' + (\hat{k}_2 \bmod \delta) rc \bmod \delta q) + \hat{k}_2 r\ell N + \delta\rho q) \bmod \delta N\\ &= ((\delta k_2^{-1}m' + (\hat{k}_2 \bmod \delta) rc \bmod \delta q) +\\ &\qquad\qquad (\hat{k}_2 r\ell \bmod \delta) N + \delta\rho q) \bmod \delta N\end{aligned}$$

The final distributions in each derivation are statistically close to one another, because both $\hat{k}_2 \bmod \delta$ and $\tilde{k}_2^{-1} \bmod \delta$ are (very nearly) uniform and independent of $k_2^{-1}$ (the value of $\tilde{k}_2^{-1}$ modulo $q$). In particular, $\tilde{k}_2^{-1} = k_2^{-1} + \tilde{\rho}q$, where $q$ is relatively prime to $\delta$, and $\tilde{\rho}$ is statistically close to uniform modulo $\delta$. □

This conclude the proof of Claim 3 that $\mathsf{view}_{\mathsf{Hyb}} \overset{c}{=} \mathsf{view}_{\mathsf{Ideal}}$. □

To complete the proof of Theorem 1, we turn to the output of the executions. We demonstrate that for any $\mathsf{view}$ in the support of either $\mathsf{view}_{\mathsf{Hyb}}$ or $\mathsf{view}_{\mathsf{Ideal}}$, $\mathsf{view}$ causes termination of the simulation if and only if $\mathsf{view}$ causes the honest party to abort without a signature. We make the argument through case analysis:

Case 1: $P_1$ fails to submit a good instance and witness to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R}_{\mathsf{DL}}}$, or $P_2$ fails to submit a good instance and witness to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R}_{\mathsf{DL}}}$. This could occur during key generation or signing. In the hybrid execution, the honest party will fail to receive $\mathsf{proof\text{-}commit}$ or $\mathsf{Proof}$ (respectively) and will terminate the protocol. In the ideal simulation, $\mathcal{S}$ will detect the bad submission to the functionality and will terminate the simulation.

Case 2: $P_1$ provides a proof $\pi \leftarrow \Pi_{\mathsf{GCD}}.\mathsf{Prv}()$, or a proof $\Psi \leftarrow \Pi_{\mathsf{eq}}.\mathsf{Prv}()$ that fails to verify. This could occur during key generation or refresh. Both $P_2$ and $\mathcal{S}$ terminate the protocol.

Case 3: $P_2$ fails to decommit to their proof. This could happen in key generation or signing. Both the honest player and $\mathcal{S}$ abort when this occurs.

Case 4: $P_2$ sends a badly formed $C'$ during signing. Here we rely on the correctness of the Paillier-EC-Refresh oracle (Section 7.1). An honest $P_1$ computes $s_2 = \mathsf{Dec}(C')$, and aborts if and only if $k_1^{-1}s_2 \bmod q$ is an invalid signature. $\mathcal{S}$ makes this determination through the oracle query. Since the response of $\mathcal{O}$ is 1 if and only if $\mathsf{Dec}(C') = k_2^{-1}(r(x_1 + x_2) + m) \bmod q$, correctness of the simulation follows.

□

**Claim 8.** Let $k_1$ and $k_2$ be two integers satisfying $k_1 \equiv k_2 \bmod q$ and $|k_1 - k_2| \leq 2^{\ell} \cdot q$. Let $\rho$ be sampled uniformly from $[2^{\ell+\tau}]$ for statistical security parameter $\tau$. Then, the distributions $k_1 + \rho \cdot q$ and $k_2 + \rho \cdot q$ are statistically indistinguishable.

*Proof.* Without loss of generality, we assume $k_2 > k_1$. Let $D_i$ denote the distribution over the integers that results from sampling $\rho \leftarrow [2^{\ell+\tau}]$ and outputting $k_i + \rho \cdot q$. Let $\mathsf{Good}$ denote the set of integers that are in the support of both $D_1$ and $D_2$, and $\mathsf{Bad}$ the set of integers that are in the support of only $D_2$. The statistical distance between $D_1$ and $D_2$ is

$$
\begin{aligned}
&\frac{1}{2}\sum_{x\in\mathbb{Z}} \big|\Pr[w = x \mid w \leftarrow D_1] - \Pr[w = x \mid w \leftarrow D_2]\big| \\
=&\frac{1}{2}\sum_{x\in\mathsf{Good}} \big|\Pr[w = x \mid w \leftarrow D_1] - \Pr[w = x \mid w \leftarrow D_2]\big| \\
&\quad + \frac{1}{2}\sum_{x\in\mathsf{Bad}} \big|\Pr[w = x \mid w \leftarrow D_1] - \Pr[w = x \mid w \leftarrow D_2]\big| \\
=&\frac{1}{2}\sum_{x\in\mathsf{Bad}} \big|\Pr[w = x \mid w \leftarrow D_1] - \Pr[w = x \mid w \leftarrow D_2]\big| \\
\leq&2^{-\tau}
\end{aligned}
$$

The first equality holds because $\mathbb{Z} \setminus (\mathsf{Good} \cup \mathsf{Bad})$ has no support, by definition. The second equality holds because, for each $w \in \mathsf{Good}$, $\Pr[w \leftarrow D_1] = \Pr[w \leftarrow D_2]$. This is true because the sample randomness mapping $k_1$ to $w \in \mathsf{Good}$ is injective, so each element in $\mathsf{Good}$ has equal probability

weight under both distributions. Finally, the last inequality holds because $\Pr[w \in \mathsf{Bad} \mid w \leftarrow D_2] \leq 2^{-\tau}$, and $\Pr[w \in \mathsf{Bad} \mid w \leftarrow D_1] = 0$. $\square$

# References


[1] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 593–611, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] Boaz Barak, Amir Herzberg, Dalit Naor, and Eldad Shai. The proactive security toolkit and applications. In *Proc. of $6^{th}$ ACM Conference on Computer and Communications Security (CCS)*. ACM, 1999.

[3] Michele Battagliola, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. Threshold ECDSA with an offline recovery party, 2021.

[4] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1. In *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings 18*, pages 1–12. Springer, 1998.

[5] C. Boyd. Digital multisignature. In *Cryptography and Coding*, pages 241–246, 1986.

[6] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 88–97, New York, NY, USA, 2002. Association for Computing Machinery.

[7] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

[8] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1769–1787, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 191–221, Cham, 2019. Springer International Publishing.

[10] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 266–296, Cham, 2020. Springer International Publishing.

[11] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, 939(C):78–104, jan 2023.

[12] R.A. Croft and Harris S.P. Public-key cryptography and reusable shared secret. In *Cryptography and Coding*, pages 189–201, 1989.

[13] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks*, pages 382–400, Cham, 2020. Springer International Publishing.

[14] Yvo Desmedt. Society and group oriented cryptography: a new concept. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 120–127, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[15] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 307–315, New York, NY, 1990. Springer New York.

[16] Jack Doerner, Yash Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA in three rounds. eprint/2023/765, 2023.

[17] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997, 2018.

[18] Y. Frankel, P. Gemmell, P.D. MacKenzie, and Moti Yung. Optimal-resilience proactive public-key cryptosystems. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 384–393, 1997.

[19] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive RSA. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 440–454, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[20] Yair Frankel, Philip Mackenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. pages 180–194, 12 1999.

[21] Adam Gagol, Jedrzej Kula, Damian Straszak, and Michal Swietek. Threshold ECDSA for decentralized asset custody. *IACR Cryptol. ePrint Arch.*, page 498, 2020.

[22] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1179–1194, New York, NY, USA, 2018. Association for Computing Machinery.

[23] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *Applied Cryptography and Network Security*, pages 156–174, Cham, 2016. Springer International Publishing.

[24] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 354–371, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[25] Rosario Gennaro, Daniele Micciancio, and Tal Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *5th ACM Conference on Computer and Communication Security (CCS'98)*, pages 67–72, San Francisco, California, November 1998. ACM, ACM Press.

[26] Sharon Goldberg, Leonid Reyzin, Omar Sagga, and Foteini Baldimtsi. Efficient noninteractive certification of RSA moduli and beyond. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 700–727. Springer, 2019.

[27] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[28] Amir Herzberg, Markus Jakobsson, Stanisłław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, page 100–110, New York, NY, USA, 1997. Association for Computing Machinery.

[29] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology — CRYPT0' 95*, pages 339–352, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[30] Stanisław Jarecki and Josh Olsen. Proactive RSA with non-interactive signing. In Gene Tsudik, editor, *Financial Cryptography and Data Security*, pages 215–230, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[31] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[32] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 608–625. IEEE, 2021.

[33] Yehuda Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II 37*, pages 613–644. Springer, 2017.

[34] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1837–1854, New York, NY, USA, 2018. Association for Computing Machinery.

[35] Philip Mackenzie and Michael Reiter. Two-party generation of dsa signatures. *International Journal of Information Security*, 2, 09 2004.

[36] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '91, page 51–59, New York, NY, USA, 1991. Association for Computing Machinery.

[37] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[38] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 89–104, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[39] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 207–220, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[40] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 1–16, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

# A The Paillier-EC-Refresh Assumption and Robustness to CCA-like Attacks

Lindell's threshold ECDSA scheme [33] uses a severe countermeasure to defend against an attack by a malicious $P_2$. Namely, if $P_2$ sends a partial signature (encrypted under $P_1$'s Paillier key) that leads to an invalid signature, the system aborts, a new signing key must be generated from scratch, and the PKI must be updated with the new signing key. This motivates the question: Can we prevent $P_2$'s attack with a less severe countermeasure – say, just a Refresh instead of a full KeyGen, or maybe even a procedure $P_1$ can perform on its own – while making virtually no compromises on performance? Here, we show that we can.

First, let us understand the attack better. The basic problem is that, in Lindell's threshold ECDSA protocol, $P_1$ decrypts a ciphertext sent by $P_2$ and reacts to the result. This is similar to the classical setting of chosen-ciphertext attacks, where there are well-known attacks that recover the secret key just from decryption error messages. One way to defeat such attacks is to use an encryption scheme with chosen ciphertext security, but Lindell's scheme uses Paillier precisely for its homomorphic properties. Another way to defeat such attacks is to require $P_2$ to use a zero-knowledge proof that its ciphertext is correctly constructed, but Lindell's scheme gets its performance precisely by avoiding expensive zero knowledge proofs. Instead, Lindell proves that his scheme is secure under the Paillier-EC assumption, a plausible assumption that permits very limited access to the decryption oracle. Namely, the oracle will validate that a Paillier ciphertext encrypts some plaintext that satisfies a congruence modulo $q$, but if validation fails the oracle will abort permanently. Unfortunately, this means the scheme must start over as well.

Surprisingly, we show that, in our scheme, a lighter countermeasure is sufficient. Namely, instead of encrypting its share $x_1 \in [q]$ under its Paillier key $N_1$, $P_1$ encrypts some value $X_1 = x_1 + q \cdot f_1$ where $f_1$ has sufficient min-entropy. In other words, it never sends a direct encryption of $x_1$ to $P_2$, but instead always first perturbs $x_1$ by a random multiple of $q$. Then, if $P_1$ detects a malicious partial signature from $P_2$, $P_1$ changes its Paillier modulus to some $N_2$ and encrypts a fresh $X_2 = x_1 + q \cdot f_2$ under $N_2$. Each time it detects an invalid partial signature, $P_1$ updates both its Paillier modulus and the perturbation of $x_1$ that it encrypts. $P_2$ can be continuously corrupted, and this countermeasure will still work to protect $x_1$. During these updates, the value of $x_1$ can remain the same, but for convenience our scheme folds this update into the proactive Refresh procedure.

Our assumption, Paillier-EC-Refresh, makes the expected changes to Lindell's Paillier-EC. Namely, we give the adversary access to several oracles in parallel, each one associated with a different value $X_i = x_1 \bmod q$ and a different Paillier modulus $N_i$. Each oracle permits one invalid query before aborting, and then a new oracle is spun up. Intuitively, the adversary cannot recover information

about $x_1$ because each oracle allows only a few bits about $X_i$ to be revealed before aborting, and these bits are statistically independent of $x_1$. Formally, we prove that the assumption is true in a generic model for elliptic curves and Paillier encryption that allows linear functions to be computed "in the exponent".
We describe Lindell's Paillier-EC assumption and our own Paillier-EC-Refresh Assumption in Section 7.

## A.1 Generic Model Evidence for the Assumption

We provide evidence supporting the Paillier-EC-Refresh assumption in a generic model where one can apply linear functions in $\mathbb{G}$ and in the plaintext space of each Paillier modulus.

**Definition 1** (Generic Model for Paillier and Elliptic Curves)**.** The model provides "handles" for elements, rather than elements themselves. For example, for $\mathbb{G}$, first a random handle for the element $1 \cdot P$ is published. Then, given $t$ previously published handles for $a_1 \cdot P, \ldots, a_t \cdot P$, together with $t$ scalars $u_1, \ldots, u_t \in [q]$ expressing a linear combination, the model publishes a handle for $(a_1u_1 + \cdots + a_tu_t) \cdot P$. The model stores handles together with their associated scalars. Two queries that end up corresponding to the same element in $\mathbb{G}$ are handled with the same response. Paillier ciphertexts are handled generically in a similar way, with complications due to encryption randomness. For Paillier modulus $N_i$, first a random handle for an encryption of 1 is published. The model stores the handle, the value encrypted, and the encryption randomness. Then, given $t$ previously published handles, a $t$-linear combination modulo $N_i$, and some randomness for re-randomization, the model publishes a handle for the resulting ciphertext and stores its associated information. If the resulting ciphertext has the same encrypted value and randomness as one previously handled, the same handle is provided. The handles are distinct across different groups, and for different elements of the same group.

We prove that the Paillier-EC-Refresh Assumption is true when the adversary interacts with the above generic model and the oracles provided in the assumption.

**Theorem 2.** *In the generic model for Paillier and elliptic curves, the Paillier-EC-Refresh assumption holds. More precisely, suppose the adversary makes at most $n_{\mathbb{G}}$ elliptic curve generic model queries, $n_{Pal}$ generic model queries for any particular Paillier modulus, and at most $n_{\mathcal{O}}$ queries to any particular oracle in the Paillier-EC-Refresh Assumption. If $|I_1| \geq k \cdot q \cdot n_{\mathcal{O}} \cdot 2^{\tau}$ and $(n_{\mathbb{G}}^2 + kn_{Pal}^2 + kn_{\mathcal{O}}) < q \cdot 2^{-\tau-2}$, then the adversary's advantage in the Paillier-EC-Refresh Problem in the generic model is at most $2^{-\tau}$.*

*Proof.* The challenger in the Paillier-EC-Refresh Assumption generates $w_0, w_1, b$. It gives the adversary handles for $1 \cdot P$ and $w_0 \cdot P$ in $\mathbb{G}$. It samples $\tilde{w}_i$ (for $N_i$) uniformly and independently from the interval $I_1$ subject to $\tilde{w}_i = w_b \bmod q$, provides handles for encryptions of these values, and stores the associated information (including encryption randomness). The adversary can access the generic model, using the provided handles as starting points. The adversary can also access the oracle $\mathcal{O}_{\tilde{w}_i}$ associated with the value $\tilde{w}_i$ encrypted under $N_i$: the adversary sends $(h, \alpha, \beta)$, where $h$ is some handle it has obtained, and $\mathcal{O}_{\tilde{w}_i}$ responds with 1 iff $h$ corresponds to an encryption of some value $Y$ such that $[Y - (\alpha + q\ell + \beta \cdot \tilde{w}_i))]_N$ is in $I_2$ and divisible by $q$, for perturbation factor $\ell$ chosen by the oracle uniformly randomly from $[k \cdot q \cdot 2^{2\tau+\kappa}]$. The challenger decommissions oracles as appropriate.

Let $\mathsf{views}$ denote the set of views that the adversary has from its generic model accesses and oracle queries. We partition $\mathsf{views}$ into two subsets, $\mathsf{views}_{\mathsf{Bad}}$ and $\mathsf{views}_{\mathsf{Good}}$. In $\mathsf{views}_{\mathsf{Bad}}$, the adversary makes a "bad query". A bad query is one of the following:

- Bad oracle query: By Theorem 3 and Corollary 1, the solution set of each oracle query is an arithmetic progression within $I_1$. We call an oracle query bad if all terms in its arithmetic progression have the same value modulo $q$, and that value is $w_0$ or $w_1$.
- Bad $\mathbb{G}$ query: The adversary obtains handles for elements in $\mathbb{G}$ of the form $\alpha_i + \beta_i \cdot w_0$. If two handles non-trivially collide, they reveal a relation for $w_0$ – namely, $(\alpha_i - \alpha_j) + (\beta_i - \beta_j) \cdot w_0 = 0 \bmod q$. We say that a $\mathbb{G}$ query is bad if, by subtracting it from another $\mathbb{G}$ query, the resulting linear equation is satisfied by $w_0$ or $w_1$.
- Bad Paillier query: For each Paillier modulus separately, the adversary obtains handles for ciphertexts in that Paillier group that are derived from given encryptions of 1 and $\tilde{w}_i$. If two handles non-trivially collide, then it may reveal a relation for $\tilde{w}_i$ and hence $w_b$. We say that a Paillier query is bad if, by subtracting it from another Paillier query, the resulting linear equation is satisfied by some $X$ in $I_1$ that equals $w_0$ or $w_1$ modulo $q$.

Now, the adversary's advantage in distinguishing $b$ is at most:

$$\begin{aligned}
&\frac{1}{2} \sum_{V \in \mathsf{views}} \left| \Pr[b = 0 | \mathsf{view\ is\ V}] - \Pr[b = 1 | \mathsf{view\ is\ V}] \right| \cdot \Pr[\text{view is V}] \\
&= \frac{1}{2} \sum_{V \in \mathsf{views}_{\mathsf{Good}}} \left| \Pr[b = 0 | \mathsf{view\ is\ V}] - \Pr[b = 1 | \mathsf{view\ is\ V}] \right| \cdot \Pr[\text{view is V}] \\
&+ \frac{1}{2} \sum_{V \in \mathsf{views}_{\mathsf{Bad}}} \left| \Pr[b = 0 | \mathsf{view\ is\ V}] - \Pr[b = 1 | \mathsf{view\ is\ V}] \right| \cdot \Pr[\text{view is V}]
\end{aligned}$$

First, consider the case of a good view. In this case, the adversary makes no queries or generic model accesses that relate directly to $w_0$ or $w_1$. By Theorem 7, and a standard hybrid argument across the $k$ oracles in which we sequentially replace $w_0$ with $w_1$ in the construction of the oracles (and with Corollary 1, which describes how to translate Paillier-EC-Refresh oracle queries to the oracles of Theorem 7), the expression above with $\mathsf{views}_{\mathsf{Good}}$ is at most $\frac{k \cdot q \cdot n_{\mathcal{O}}}{2|I_1|}$.

The part of the expression above with $\mathsf{views}_{\mathsf{Bad}}$ is upper bounded by $\frac{1}{2} \sum_{V \in \mathsf{views}_{\mathsf{Bad}}} \Pr[\mathsf{view\ is\ V}]$. It remains to bound the probability of a bad view. Any bad view $V$ has some *first* bad query. The roadmap for the rest of the proof is: 1) Show that, just before the first bad query, the adversary has negligible information about $w_0$ and $w_1$ – both $w_0$ and $w_1$ appear to have an almost uniform distribution over $[q]$, and 2) Conclude that, given its negligible information about $w_0$ and $w_1$, the chance of its next query being bad – of it essentially guessing $w_0$ or $w_1$ – is negligible.

First, consider the adversary's view just before the first bad query. The adversary has made at most $n_{\mathbb{G}}$ accesses to the generic model for $\mathbb{G}$, at most $n_{Pal}$ accesses to each of the $k$ Paillier generic models, and at most $n_{\mathcal{O}}$ queries to each of the $k$ oracles. However, none of the $\binom{n_{\mathbb{G}}}{2}$ pairs of handles for $\mathbb{G}$ correspond to a bad query, so these queries merely reveal at most $\binom{n_{\mathbb{G}}}{2}$ values modulo $q$ that are *not* in $\{w_0, w_1\}$. Similarly, the Paillier handles reveal at most $k \cdot \binom{n_{Pal}}{2}$ more values modulo $q$ that are *not* in $\{w_0, w_1\}$ (being generous to the adversary here). Finally, the adversary has made at most $k$ oracle queries that we call $q$-progression queries – queries whose solution set consists of elements that all equal the same value modulo $q$. There are at most $k$ such queries, at most 1 per oracle, because the all of the responses to them were 0, for otherwise there would be a bad query

among them. These at most $k$ $q$-progression queries reveal $k$ additional values modulo $q$ that are not in $\{w_0, w_1\}$. Altogether, the generic model queries and the $q$-progression oracle queries "knock out" at most $n_G^2 + k \cdot n_{Pal}^2 + k$ values modulo $q$.
Of the subset of $q - n_G^2 - k \cdot n_{Pal}^2 - k$ values remaining, Theorem 7 (with a hybrid argument over the $k$ oracles) states that all of these remaining values look statistically equally likely to be in $\{w_0, w_1\}$, up to an advantage of $k \cdot q \cdot n_{\mathcal{O}}/2|I_1|$. This implies that the maximum likelihood $p_{max}$ of any of these mod-$q$ values being in $\{w_0, w_1\}$ is:

$$\begin{aligned} p_{max} &= \max_{u \in [q]} \Pr[u \in \{w_0, w_1\}] \\ &\leq \frac{2}{q - n_G^2 - k \cdot n_{Pal}^2 - k} \cdot \frac{1 + k \cdot q \cdot n_{\mathcal{O}}/|I_1|}{1 - k \cdot q \cdot n_{\mathcal{O}}/|I_1|} \end{aligned}$$

The term multiplied on the right comes from the fact that, if the maximum advantage for guessing between two choices is $\epsilon$, then probabilities are $1/2 + \epsilon$ and $1/2 - \epsilon$, and the former is bigger by a factor of $(1 + 2\epsilon)/(1 - 2\epsilon)$.
Having characterized the adversary's view before its first bad query, we consider the bad query itself, considering the 3 types of queries (oracle query, $\mathbb{G}$ query and Paillier query). A bad oracle query is one in which the solution set is either entirely $w_0$ or $w_1$ modulo $q$. By the union bound over the adversary's $k \cdot n_{\mathcal{O}}$ oracle queries, the probability of this event is at most $p_{max} \cdot k \cdot n_{\mathcal{O}}$.
A bad $\mathbb{G}$ query is one in which the relation corresponding to a pair of colliding handles is satisfied by $w_0$ or $w_1$. By the union bound, the probability of a bad $\mathbb{G}$ query is at most $p_{max}\binom{n_{\mathbb{G}}}{2} < p_{max} \cdot n_{\mathbb{G}}^2$.
The case of a bad Paillier query is similar.
Putting this together, we have:

$$\begin{aligned} & \frac{1}{2} \sum_{V \in \mathsf{views}_{\mathsf{Bad}}} \Pr[\mathsf{view\ is\ V}] \\ \leq\ & (n_{\mathbb{G}}^2 + k n_{Pal}^2 + k n_{\mathcal{O}}) \cdot \frac{1}{q - n_G^2 - k \cdot n_{Pal}^2 - k} \cdot \frac{1 + k \cdot q \cdot n_{\mathcal{O}}/|I_1|}{1 - k \cdot q \cdot n_{\mathcal{O}}/|I_1|}. \end{aligned}$$

Adding this together with the adversary's advantage coming from $\mathsf{views}_{\mathsf{Good}}$, we have:

$$\begin{aligned} \mathrm{Advantage}_{\mathrm{Adversary}} &\leq \frac{k \cdot q \cdot n_{\mathcal{O}}}{2|I_1|} \\ &+ (n_{\mathbb{G}}^2 + k n_{Pal}^2 + k n_{\mathcal{O}}) \cdot \frac{1}{q - n_G^2 - k \cdot n_{Pal}^2 - k} \cdot \frac{1 + k \cdot q \cdot n_{\mathcal{O}}/|I_1|}{1 - k \cdot q \cdot n_{\mathcal{O}}/|I_1|}. \end{aligned}$$

The first expression on the right hand side of the inequality is at most $2^{-\tau-1}$, because we assumed $|I_1| \geq k \cdot q \cdot n_{\mathcal{O}} \cdot 2^{\tau}$. The second expression is also at most $2^{-\tau-1}$, because we assumed $(n_{\mathbb{G}}^2 + k n_{Pal}^2 + k n_{\mathcal{O}}) < q \cdot 2^{-\tau-2}$, and the ratio $\frac{q}{q - n_G^2 - k \cdot n_{Pal}^2 - k} \cdot \frac{1 + k \cdot q \cdot n_{\mathcal{O}}/|I_1|}{1 - k \cdot q \cdot n_{\mathcal{O}}/|I_1|}$ is less than 2 for our parameters. The theorem follows. □

## A.2 Solution Sets Are Arithmetic Progressions

Below, Theorem 3 states that, when $I_1$ and $I_2$ are sets of consecutive integers inside $[N]$ satisfying certain conditions, the set $\{X \in I_1 : [a + bX]_N \in I_2\}$ is a (finite) arithmetic progression inside

$I_1$. Afterwards, we explain how this result applies to our generic model proof that the Paillier-EC-Refresh Assumption holds.
This arithmetic progression result may seem rather esoteric, but it helps us prove strong statistical uniformity results in Section A.3.

**Theorem 3.** *Let $[N]$ be a set of $N$ consecutive integers – for example, $\{0, \ldots, N-1\}$. Let $[\cdot]_N$ denote reduction modulo $N$ into $[N]$. Let $I_1, I_2 \subset [N]$ be two nonempty sets of consecutive integers satisfying $2|I_1||I_2| < N$. For integers $a, b$, let $S_{a,b,I_1,I_2} = \{X \in I_1 : [a + bX]_N \in I_2\}$. Then, the elements of $S_{a,b,I_1,I_2}$ form a (finite) arithmetic progression.*

We give two proofs of Theorem 3, one elementary, a second based on two-dimensional lattices.

*Proof. (Elementary)* $S_{a,b,I_1,I_2}$ is trivially an arithmetic progression if it has fewer than 3 elements, so assume the contrary. Let distinct $s_1, s_2 \in S_{a,b,I_1,I_2}$ be such that $\delta = |s_2 - s_1|$ is minimized. Let $A$ be the longest arithmetic progression inside $S_{a,b,I_1,I_2}$ containing $s_1$ and $s_2$. The progression $A$ has step size $\delta$. Our claim is that $S_{a,b,I_1,I_2} = A$. Let $A' \subset I_1$ be the continuation of the arithmetic progression $A$ across all of $I_1$.
Some facts:

1. For any arithmetic progression $B$, $\{a + bX \bmod N : X \in B\}$ is an arithmetic progression modulo $N$, though the set of integers $\{[a + bX]_N : X \in B\}$ may not be due to wrapping modulo $N$.

2. For consecutive elements in $A$ like $s_1$ and $s_2$ that differ by $\delta$, their corresponding values $[a + bX]_N$ differ by the same $\Delta = b\delta \bmod N$, where $|\Delta| < |I_2|$. So, for small step size $\delta$ in $A$, the corresponding value of $[a + bX]_N$ makes a "small" step size $\Delta$.

3. We have $b\delta = \Delta + \ell N$ for some $\ell$ co-prime to $\delta$, for otherwise if there were a common divisor $d$ we would have $b(\delta/d) = (\Delta/d) + (\ell/d)N$ with $\Delta/d$ an integer, which would imply that we could have taken smaller steps $\delta/d$ in $I_1$ and induce smaller steps $\Delta/d$ in $I_2$.

4. For arithmetic progression $A$, the set of integers $\{[a + bX]_N : X \in A\}$ is an arithmetic progression within $I_2$. This is because $\{[a + bX]_N : X \in A\}$ is an arithmetic progression modulo $N$ that stays within $I_2 \subset [N]$, with no wrapping because the step size $|\Delta| < |I_2|$ is too small to step over $[N] \setminus I_2$, as $2|I_2| < N$.

5. For elements in $A'$ outside of $A$, the corresponding value of $[a+bX]_N$ is outside of $I_2$ but "close" to $I_2$, since continuing in the arithmetic progression $A'$ beyond $A$ induces $a + bX \bmod N$ to take small steps of size $|\Delta|$ away from $I_2$ without going far enough to wrap. Specifically, since $|A'| \leq |I_1|/\delta + 1$, and since $A'$ has at least two elements in $S_{a,b,I_1,I_2}$, there are at most $|I_1|/\delta - 1$ elements of $A'$ outside of $A$. So, for $X \in A'$, $a + bX$ modulo $N$ can be at most $|\Delta| \cdot (|I_1|/\delta - 1) < |I_1||I_2|/\delta$ away from $I_2$, which is not far enough to wrap, since $(|I_1| + 1)|I_2| < N$. Therefore, $S_{a,b,I_1,I_2}$ contains no elements in $A' \setminus A$.

6. For elements $X$ in $I_1 \setminus A'$, we can express $X$ as $X_0 + X_1$, where $X_0 \in A'$ and $X_1 \in \{-\delta + 1, \ldots, -1, 1, \delta - 1\}$. Then, $a + bX = (a + bX_0) + bX_1 \bmod N$. As mentioned, $a + bX_0 \bmod N$ is either in $I_2$ or "close" to it. We claim that $bX_1 \bmod N$ is so "large" that it is impossible for $a + bX \bmod N$ to be in $I_2$. Recall that we have $b\delta = \Delta + \ell N$ for some $\ell$ co-prime to $\delta$. So, $bX_1 = (X_1/\delta) \cdot (\Delta + \ell N)$, as an integer, where $|(X_1/\delta) \cdot \Delta| < |\Delta|$ is "small" and $(X_1 \cdot \ell/\delta) \cdot N$

equals $\lfloor X_1 \cdot \ell/\delta \rfloor \cdot N + ((X_1 \cdot \ell \bmod \delta)/\delta) \cdot N$, where the former expression is a multiple of $N$ and the latter has the form $(r/\delta) \cdot N$ for $r \in \{1, \ldots, \delta - 1\}$ (using here the fact that $X_1$ and $\ell$ are co-prime to $\delta$). For small $\delta$ and $r \in \{1, \ldots, \delta - 1\}$, $(r/\delta) \cdot N$ is very distant from a multiple of $N$. Pulling this together, $bX_1 \bmod N$ has magnitude at least $N/\delta - |\Delta|$. Therefore, $a + bX$ is at least $N/\delta - |\Delta| - |\Delta| \cdot (|I_1|/\delta - 1) - |I_2|$ "away" from $I_2$, where this quantity is at least $N/\delta - |\Delta||I_1|/\delta - |I_2| = (1/\delta) \cdot (N - |\Delta||I_1| - |I_2|\delta) > (1/\delta) \cdot (N - 2|I_1||I_2|) > 0$. Concluding, $S_{a,b,I_1,I_2}$ contains no elements outside of $A'$.

□

*Proof. (Lattice-Based)* Let $L_a = \{X, Y \in \mathbb{Z}^2 : Y = a + bX \bmod N\}$. This set is a translation of the two-dimensional lattice $L = \{X, Y \in \mathbb{Z}^2 : Y = bX \bmod N\}$. Let $R$ be the two-dimensional "rectangle" $I_1 \times I_2$. Let $S = L_a \cap R$ be their intersection. We claim that $S$ is a (finite) progression of equally spaced points on a 1-dimensional line. Since $S_{a,b,I_1,I_2}$ is the set of first coordinates of points in $S$, the theorem follows from the claim.
The claim is trivially true if $S$ has fewer than 3 points, so assume the contrary. Let $(X_1, Y_1)$, $(X_2, Y_2)$, $(X_3, Y_3)$ be three arbitrary distinct points in $S$. Let $(\delta_1, \Delta_1)$ denote $(X_1 - X_3, Y_1 - Y_3)$, and $(\delta_2, \Delta_2)$ denote $(X_2 - X_3, Y_2 - Y_3)$. We have:

$$\begin{aligned} b \cdot \delta_1 &= \Delta_1 \bmod N \\ b \cdot \delta_2 &= \Delta_2 \bmod N. \end{aligned}$$

Therefore, $b \cdot \delta_1 \cdot \Delta_2 - b \cdot \delta_2 \cdot \Delta_1 = 0 \bmod N$.
If $\gcd(b, N) = 1$, then $\delta_1 \cdot \Delta_2 - \delta_2 \cdot \Delta_1 = 0 \bmod N$. But we also have $|\delta_1 \cdot \Delta_2 - \delta_2 \cdot \Delta_1| < N$ because $2|I_1||I_2| < N$. So, $\delta_1 \cdot \Delta_2 - \delta_2 \cdot \Delta_1 = 0$ over the integers. Hence, $(X_1, Y_1)$, $(X_2, Y_2)$, and $(X_3, Y_3)$ are collinear. Since these points were arbitrary, all solutions are collinear, lying on some line $\ell$. We conclude that $S = L_a \cap R \cap \ell$, where the right hand side is clearly a (finite) progression of equally spaced points on a 1-dimensional line.
For the case $d = \gcd(b, N)$ is nontrivial, let $b' = b/d$, $N' = N/d$, $\Delta_1' = \Delta_1/d$, and $\Delta_2' = \Delta_2/d$. Then, we have $\delta_1 \cdot \Delta_2' - \delta_2 \cdot \Delta_1' = 0 \bmod N'$ and $|\delta_1 \cdot \Delta_2' - \delta_2 \cdot \Delta_1'| < N'$, and hence $\delta_1 \cdot \Delta_2' - \delta_2 \cdot \Delta_1' = 0$ over the integers, so that we obtain the collinearity result again, and proceed as before. □

How is this result relevant to our generic model proof that the Paillier-EC-Refresh Assumption holds?

**Corollary 1.** *The solution sets of oracle queries in our generic model proof are arithmetic progressions.*

*Proof.* Recall that, in Paillier-EC-Refresh, the ciphertext $C_i$ encrypts a value $\tilde{w}_i = w_b \bmod q$. The adversary is given an oracle $\mathcal{O}_{C_i}$ such that $\mathcal{O}_{C_i}(C', \alpha, \beta) = 1$ iff $[(\mathsf{Dec}_{\mathsf{sk}_i}(C') - (\alpha + q\ell + \beta \cdot \tilde{w}_i))]_N$ is in $I_2$ and divisible by $q$, for some perturbation factor $\ell$ chosen by the oracle. But, in the generic model, $C'$ is a handle for a ciphertext and $\mathsf{Dec}_{\mathsf{sk}_i}(C')$ is known as a linear expression $\alpha' + \beta' \cdot \tilde{w}_i$. So, in fact the oracle is testing whether $[(\alpha' - \alpha - q\ell) + (\beta' - \beta) \cdot \tilde{w}_i)]_N$ is $I_2$ and divisible by $q$. Let $I_2' = \{z : zq \in I_2\}$. Then, the oracle is testing whether $[(q^{-1} \bmod N)(\alpha' - \alpha - q\ell) + (q^{-1} \bmod N)(\beta' - \beta) \cdot \tilde{w}_i))]_N$ is in $I_2'$. Now, Theorem 3 applies, using $a = (q^{-1} \bmod N)(\alpha' - \alpha - q\ell)$ and $b = (q^{-1} \bmod N)(\beta' - \beta)$ and the output interval $I_2'$, where $|I_2'|$ equals $|I_2|/q$ (rounded up or down). □

Suppose the adversary makes several oracle queries that get several positive responses. By Theorem 3, the solution set then corresponds to an intersection of (finite) arithmetic progressions. The following theorem states that this intersection is itself a finite arithmetic progression.

**Theorem 4.** *The intersection of (finite) arithmetic progressions is a (finite) arithmetic progression.*

*Proof.* Let $\{A_i\}$ be a set of (finite) integer arithmetic progressions over intervals $\{I_i\}$ – namely, $A_i = \{c_i + \delta_i \cdot k \in I_i : k \in \mathbb{Z}\}$. Unless the intersection is empty, they all have a common element $c$. Set $\delta$ to be the lowest common multiple of $\{\delta_i\}$. Set $I = \cap_i I_i$, which is itself an interval. Then, the intersection is $A = \cap_i A_i = \{c + \delta \cdot k \in I : k \in \mathbb{Z}\}$. □

In Paillier-EC-Refresh, each oracle permits multiple positive responses and one negative response. The solution set here is the difference of two arithmetic progressions.

**Theorem 5.** *The intersection of (finite) arithmetic progressions with the complement of an arithmetic progression is a difference of arithmetic progressions.*

*Proof.* Denote the arithmetic progressions by $A_1, \ldots, A_m$, where $A_m$ is the one we take the complement of. We are interested in $(\cap_{i=1}^{m-1} A_i) \setminus A_m$.
Let $B_i$ denote $\cap_{j=1}^{i} A_j$. By Theorem 4, $B_i$ is an arithmetic progression. And the above set is precisely $B_{m-1} - B_m$. □

**Remark 1.** *For set difference, we use the $A \setminus B$ notation when it is not assumed that $B \subset A$. We use the $A - B$ exclusively when $B$ is known to be a subset of $A$.*

## A.3 Statistical Results

An easy first result is that arithmetic progressions have good statistical uniformity properties modulo $q$, as long as their step size $\delta$ is co-prime to $q$.

**Theorem 6.** *Let $A$ be an arithmetic progression with step size $\delta$ that is co-prime to $q$. Then, for each $x \in [q]$, $A$ has either $\lfloor |A|/q \rfloor$ or $\lceil |A|/q \rceil$ elements congruent to $x$ modulo $q$.*

*Proof.* (Trivial.) □

As a corollary, when it also holds that $q/|A|$ is negligible, $A$ is statistically uniform modulo $q$.
Now, suppose a computationally unbounded algorithm $\mathcal{A}$ interacts with an oracle $\mathcal{O}_X$, $X \in I_1$, that on input any $a, b$, outputs 1 iff $[aX + b]_N \in I_2$, and otherwise outputs 0 and aborts. In the setting here, we will give $\mathcal{A}$ both more and less power than the adversary in the Paillier-EC-Refresh assumption, as follows:

- We let the adversary know $u_0, u_1$ such that $X = u_b \bmod q$ for $b \in \{0, 1\}$.
- We give $\mathcal{A}$ access to $\mathcal{O}_X$, except that we disallow a certain type of "bad oracle query". Namely, each oracle query corresponds to an arithmetic progression $A$, by Theorem 3. We say that a query is a "$q$-progression query" for residue $u$ if its associated progression $A$ has entries all congruent to $u$ modulo $q$. We prohibit $\mathcal{A}$ from making a "bad oracle query", defined as a $q$-progression query on $u_0$ or $u_1$.

Theorem 7 says that $\mathcal{A}$ has negligible advantage in guessing $b$. Note that $\mathcal{A}$ could easily guess $b$ if it were allowed to make a bad oracle query. While the setting is different from the Paillier-EC-Refresh assumption, Theorem 7 will be a component of our proof that the Paillier-EC-Refresh assumption is true in a generic model.

**Theorem 7.** *Let $N, q$ be integers, $q$ prime. Let $[\cdot]_N$ denote reduction modulo $N$ into $[N]$. Let $I_1, I_2 \subset [N]$ be two nonempty sets of consecutive integers satisfying $2|I_1||I_2| < N$, with $|I_1|$ divisible by $q$. Let $n_{\mathcal{O}}$ be a parameter, representing a maximum permitted number of oracle queries. Consider the following interaction with an oracle. The values $u_0, u_1$ are sampled uniformly from $[q]$, $b$ is sampled from $\{0,1\}$, and $X$ is sampled uniformly randomly from $I_1$ subject to $X = u_b \bmod q$. Having fixed $X$, the oracle $\mathcal{O}_X$, on input any $a, b$, outputs 1 iff $[a + bX]_N \in I_2$. Otherwise, $\mathcal{O}_X$ outputs 0 and aborts permanently. Any algorithm $\mathcal{A}$ given access to $u_0, u_1$ and $\mathcal{O}_X$ – subject to the restrictions that it can make at most $n_{\mathcal{O}}$ queries to $\mathcal{O}_X$ and that it cannot make bad oracle queries (as defined above) – has advantage at most $\frac{q \cdot n_{\mathcal{O}}}{2|I_1|}$ in guessing $b$. For example, if $|I_1| \geq q \cdot n_{\mathcal{O}} \cdot 2^{\tau - 1}$ for statistical security parameter $\tau$, $\mathcal{A}$'s advantage is negligible.*

*Proof.* Fix $u_0, u_1$ and fix the algorithm $\mathcal{A}$ and its randomness (but not $b$ or $X$). $\mathcal{A}$ makes a fixed first query $Q_1$ and receives either 0 or 1 in response. If 0, the interaction is terminated. If 1, $\mathcal{A}$ continues with its next fixed query $Q_2$. And so on. $\mathcal{A}$ thereby follows a fixed chain of up to $n_{\mathcal{O}}$ oracle queries.

By Theorem 3, the set of values in $I_1$ consistent with a 1 response to query $Q_i$ is an arithmetic progression $A_i$, while a 0 response is consistent with $I_1 \setminus A_i$. Let $B_i = \cap_{j=1}^{i} A_j$. If $\mathcal{A}$'s $m$-th query is the first 0 response from the oracle, then by Theorems 4 and 5, $X$ is in $B_{m-1} - B_m$. The $B_{i-1} - B_i$ terms form a telescoping sum with $B_0 = I_1$ and $B_i = \emptyset$ for all $i > n_{\mathcal{O}}$. Let $P_0, P_1$ denote the arithmetic progressions $\{X \in I_1 : X = u_0 \bmod q\}$ and $\{X \in I_1 : X = u_1 \bmod q\}$, respectively. $\mathcal{A}$ knows a priori that $X$ is in $P_0 \cup P_1$.

Now, letting $\mathsf{views}$ denote the set of views $\mathcal{A}$ may get from oracle $\mathcal{O}_X$, we have that $\mathcal{A}$'s advantage in guessing $b$ is at most:

$$\frac{1}{2} \sum_{V \in \mathsf{views}} \big| \Pr[b = 0 | \mathsf{view\ is\ V}] - \Pr[b = 1 | \mathsf{view\ is\ V}] \big| \cdot \Pr[\mathsf{view\ is\ V}] \tag{3}$$

$$= \frac{1}{2} \sum_{V \in \mathsf{views}} \big| \Pr[b = 0 \wedge \mathsf{view\ is\ V}] - \Pr[b = 1 \wedge \mathsf{view\ is\ V}] \big| \tag{4}$$

$$= \frac{1}{2} \sum_{V \in \mathsf{views}} \big| \Pr[\mathsf{view\ is\ V} | b = 0] \Pr[b = 0] - \Pr[\mathsf{view\ is\ V} | b = 1] \Pr[b = 1] \big| \tag{5}$$

$$= \frac{1}{4} \sum_{V \in \mathsf{views}} \big| \Pr[\mathsf{view\ is\ V} | b = 0] - \Pr[\mathsf{view\ is\ V} | b = 1] \big| \tag{6}$$

$$= \frac{1}{4} \sum_{m \in [n_{\mathcal{O}}]} \left| \frac{|P_0 \cap (B_{m-1} - B_m)|}{|P_0|} - \frac{|P_1 \cap (B_{m-1} - B_m)|}{|P_1|} \right| \tag{7}$$

$$= \frac{q}{4|I_1|} \sum_{m \in [n_{\mathcal{O}}]} \big| |P_0 \cap (B_{m-1} - B_m)| - |P_1 \cap (B_{m-1} - B_m)| \big| \tag{8}$$

$$\leq \frac{q}{4|I_1|} \sum_{m \in [n_{\mathcal{O}}]} 2 \tag{9}$$

$$= \frac{q \cdot n_{\mathcal{O}}}{2|I_1|}. \tag{10}$$

The above inequality holds for all $\mathcal{A}$ and all its choices of randomness. Equation 7 relabels $\mathcal{A}$'s view depending on which query to $\mathcal{O}_X$ gets a 0 response. The probability that the $m$-th response is the first 0 depends on the set $B_{m-1} - B_m$ of candidate $X$ values and how that set intersects $P_0$ or $P_1$. Equation 8 comes from our requirement that $|I_1|$ is divisible by $q$ so that it hits all residues perfectly evenly; if $I_1$ did not have this property, we would need to make a very small adjustment. We now prove Equation 9.

Let $\delta_{m-1}$, $\delta_m$ be the step sizes of the arithmetic progressions $B_{m-1}$, $B_m$. We claim that $\delta_{m-1}$ is not divisible by $q$. Assume the contrary. Then, since $B_{m-1}$ is the intersection of $A_1, \ldots, A_{m-1}$ and since the step size of the intersection of arithmetic progressions is the LCM of the individual progressions, one of $A_1, \ldots, A_{m-1}$ must have step size divisible by $q$. But this is impossible, since this would imply that one of $\mathcal{A}$'s first $m-1$ queries was a successful $q$-progression query, and thus a $q$-progression query for residue $u_b$, which is prohibited. Contradiction.

If $\delta_{m-1}$ and $\delta_m$ are both co-prime to $q$, then by Theorem 6, each residue $x$ modulo $q$ is represented in $B_{m-1} - B_m$ at least $\lfloor |B_{m-1}|/q \rfloor - \lceil |B_m|/q \rceil$ times and at most $\lceil |B_{m-1}|/q \rceil - \lfloor |B_m|/q \rfloor$ times. These quantities differ by at most 2, proving Equation 9 for this case.

If $\delta_{m-1}$ is co-prime to $q$ and $\delta_m$ is not, then $B_m$'s terms all have the same value (some $c$) modulo $q$. Note that $c$ cannot be $u_0$ or $u_1$, since this is a $q$-progression query. Subtracting $B_m$ from $B_{m-1}$ just depresses $c$'s representation, and otherwise leaves $B_{m-1}$'s representation of the residues modulo $q$ unaffected. In $B_{m-1}$, each residue $x$ modulo $q$ is represented at least $\lfloor |B_{m-1}|/q \rfloor$ times and at most $\lceil |B_{m-1}|/q \rceil$ times. These quantities differ by at most 1, proving Equation 9 for this case. □

### A.4 Would a Weaker Refresh Be Sufficient?

To overcome a malicious $P_2$ who sends an invalid encrypted partial signature, $P_1$ refreshes both its Paillier modulus and the value $X = x_1 \bmod q$ that it encrypts under its modulus to represent its key share $x_1$. Would a weaker refresh be sufficient to overcome $P_2$'s attack? Would it be sufficient for $P_1$ to refresh $X$ while re-using the same Paillier modulus, or to refresh its Paillier modulus but re-use the same $X$? Unfortunately not. Either weakening of Refresh has serious security issues.

Suppose that $P_1$ refreshes just $X$, while keeping the Paillier modulus $N$ the same. Let $X_1, X_2, \ldots$ be the refreshed values, encrypted as $c_1, c_2, \ldots$ under $N$. Upon each Refresh, the adversary is given access to a fresh oracle $\mathcal{O}_{c_i}$. All of these oracles interoperate in the sense that they use the same $N$: a ciphertext derived from $c_i$ could be used to construct an input ciphertext for oracle $\mathcal{O}_{c_j}$ for $i \neq j$. Consequently, the adversary is effectively given a higher oracle budget for $c_1$. It can query $k$ oracles with ciphertexts derived from $c_1$, and it is allowed $k$ '0' responses before the oracles abort. If $k$ is sufficiently large, the adversary will eventually obtain enough information about $X_1$ to recover it completely.

Suppose that $P_1$ refreshes just the Paillier modulus, not the value $X_1 = x_1 \bmod q$ that is encrypted. The problem is similar to above. Again, the adversary is given access to several oracles – albeit tagged to different Paillier moduli – each of which can be used to gain information about $X_1$, and thereby $x_1$.

Changing both the modulus and encrypted value ensures that no information about $X_1$ can be used effectively after the first Refresh, barring some "spooky interaction" among different Paillier moduli.

## B Active, Proactive Security

We provide a refresher of the real/ideal security framework – the reader can find more details in Goldreich's book on secure computation [27] – and then discuss the issues that relate to proactive security. Security is defined by comparing the protocol to an ideal execution of a threshold-ECDSA functionality, $\mathcal{F}_{\mathsf{TS}}$ (Figure 8), which is secure by definition. In the case of threshold signatures, the functionality is *reactive*, which means it might be queried repeatedly, providing new output in response to each query. A distinguisher, $\mathcal{D}$, is provided a transcript of messages that were either

received by an adversary $\mathcal{A}$ while executing the signing protocols in the real execution, or they were generated by a simulator $\mathcal{S}$, who was given only black-box access to $\mathcal{A}$, and access to $\mathcal{F}_{\mathsf{TS}}$. In addition, $\mathcal{D}$ is given the output of the honest party, $\mathcal{H}$. Formally, $\mathcal{D}$ must distinguish whether it is given a sample from

$$\mathsf{Real}_{\pi,\mathcal{A}}(1^\kappa, \tau, z) = (\mathsf{view}_{\pi,\mathcal{A}}(1^\kappa, \tau, z), \mathsf{out}_{\pi,\mathcal{H}}(1^\kappa, \tau))$$

or

$$\mathsf{Ideal}_{\mathcal{F}_{\mathsf{TS}},\mathcal{S}}(1^\kappa, \tau, z) = (\mathsf{view}_{\mathcal{F}_{\mathsf{TS}},\mathcal{S}}(1^\kappa, \tau, z), \mathsf{out}_{\mathcal{F}_{\mathsf{TS}},\mathcal{H}}(1^\kappa, \tau)),$$

where $\pi$ is the real-world execution of the protocol, $\kappa$ is the computational security parameter, $\tau$ is the statistical security parameter, and $z$ is some arbitrary auxiliary information. We provide some of the details of the experiments defining these two joint distributions next.

**Setup:** $\mathcal{D}$ provides an arbitrary, identical sequence of commands to both the adversary and the honest party, drawn from the set: $\{(\mathsf{KeyGen}, \mathbb{G}, G, q)$, $(\mathsf{ssid}, \mathsf{Sign}, m)$, $(\mathsf{ssid}, \mathsf{Refresh})$, $(\mathsf{ssid}, \mathsf{Corrupt}, P_i)$, $(\mathsf{ssid}, \mathsf{De\text{-}corrupt})\}$, with $\mathsf{KeyGen}$, appearing exactly once, either as the first command in the sequence, or immediately after a single $\mathsf{Corrupt}$ command. (We will discuss some additional constraints on the command sequence later in this section.) The values of $m$ that are to be signed can be chosen arbitrarily as well. This sequence can be fixed adaptively, with $\mathcal{D}$ choosing the next command after seeing the result of the previous one, but for simplicity we consider the simpler case. $\mathcal{D}$ does not know whether he has provided the command sequence to $\mathcal{S}$ and $\mathcal{H}$, acting in the ideal world, or to $\mathcal{A}$ and $\mathcal{H}$ acting in the real execution. After the experiment concludes, $\mathcal{D}$ will output a guess as to which was the case.

**Ideal world execution:** In the ideal-world execution, $\mathcal{S}$ is given black-box oracle access to $\mathcal{A}$. $\mathcal{A}$ may or may not follow the protocol honestly. For each command in the command sequence, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{TS}}$ to learn $\mathcal{A}$'s output: $X$ during $\mathsf{KeyGen}$ and the ECDSA signature $(r, s)$ on $m$ during signing. After interacting with $\mathcal{A}$, $\mathcal{S}$ eventually outputs a transcript, and indicates to $\mathcal{F}_{\mathsf{TS}}$ whether to provide output to $\mathcal{H}$ for this command. (This is known as *security with abort*.) It then proceeds to the next command in the same manner. If $\mathcal{D}$ provides the command $(\mathsf{ssid}, \mathsf{Corrupt}, P_i)$, $\mathcal{S}$ provides a simulated view of the stored state of $P_i$, and continues simulating the view of $P_i$ until it arrives at command $(\mathsf{ssid}, \mathsf{De\text{-}corrupt})$. At that point, it provides no output until the next command of the form $(\mathsf{ssid}, \mathsf{Corrupt}, P_i)$.

**Real world execution:** In the real world execution of the protocols, $\mathcal{D}$ provides the sequence of commands to the adversary $\mathcal{A}$, and to honest party $\mathcal{H}$. The two of them execute the protocols corresponding to the commands, sending messages back and forth. At the end, $\mathcal{A}$ forwards its entire view of those executions to $\mathcal{D}$, and $\mathcal{H}$ forwards its outputs from the same executions. When the command is $(\mathsf{ssid}, \mathsf{Corrupt}, P_i)$, $\mathcal{A}$ takes control of $P_i$, gaining access to its stored state, and begins behaving as it likes on behalf of $P_i$ in further protocol executions, until it arrives at command $(\mathsf{ssid}, \mathsf{De\text{-}corrupt})$.

**Security:** We say that the threshold signatures scheme *securely realizes* $\mathcal{F}_{\mathsf{TS}}$ if $\mathcal{D}$ cannot distinguish between the ideal-world execution and the real-world execution with better than $\mathsf{negl}(\kappa)$ advantage.

**Security of Refresh:** We note that in the ideal-world, there are no key shares held by the two parties, or even by the ideal functionality. Rather, $\mathcal{F}_{\mathsf{TS}}$ generates a standard ECDSA key, provides the two parties with the public key, and stores the secret key for signing. Therefore, there is no

input/output behavior for $\mathsf{Refresh}$ defined in $\mathcal{F}_{\mathsf{TS}}$. Nevertheless, it is still important to prove that a) refresh does not leak anything about the secret key, and b) that it performs its intended role of re-randomizing the secret key. These properties are captured by a) requiring $\mathcal{S}$ to simulate the view of the adversary during $\mathsf{Refresh}$, and b) requiring $\mathcal{S}$ to simulate the state of party $P_b$ on command $(\mathsf{ssid}, \mathsf{Corrupt}, P_b)$. In the latter case, $\mathcal{S}$ would fail if the execution of $\mathsf{Refresh}$ did not result in a uniformly distributed key share.

**Hybrid world execution:** To simplify the presentations of the protocols and the proofs, we modify the real-world execution, giving the parties access to two ideal functionalities related to zero knowledge of discrete log: $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R}_{\mathsf{DL}}}$, and $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R}_{\mathsf{DL}}}$, as well as $\mathcal{F}_{\mathsf{coin}}$, a functionality for producing a random coin flip. By a classic result of Canetti [7], it suffices to prove that this *hybrid-world execution* is indistinguishable from the ideal-world execution. In practice, these functionalities can then later be replaced by any secure protocols that realize them, yielding a full description of a true real-world execution. Canetti's result says (generically) that if the hybrid-world protocol is indistinguishable from the ideal-world execution, then it follows immediately that the resulting real-world protocol is secure. In the ideal-world execution, there is only $\mathcal{F}_{\mathsf{TS}}$, and no other functionalities. It is the role of $\mathcal{S}$ to simulate the output of $\mathcal{F}_{\mathsf{zk}}$, $\mathcal{F}_{\mathsf{com-zk}}$ and $\mathcal{F}_{\mathsf{coin}}$.
We note that realizing these ideal functionalities requires a zero knowledge proof of knowledge: the simulator needs to be able to extract the witness in order to provide it to the functionality. Such constructions for discrete log are well known. Two other zero knowledge proofs that we use in our constructions, $\Pi_{\mathsf{GCD}}$ and $\Pi_{\mathsf{eq}}$, are not proofs of knowledge, and so we do not define ideal functionalities for these. Instead we use them in-line, and make use of the simulators that are known to exist. We define those proof below.

# C Zero Knowledge Proof of GCD

Let $\alpha$ be some lower bound on the largest prime divisor of $\mathsf{GCD}(N, \phi(N))$. Then the probability that a random element in $\mathbb{Z}_N^*$ has an $N$th root is at most $\frac{1}{\alpha}$ (Lemma A.8, [26]). The protocol proceeds with the verifier sending random challenge values, $y_1, \ldots, y_m$ to the prover. The prover demonstrates that each of these has an $N$th root, by finding $t_1, \ldots, t_m$ such that $t_i^N \equiv y_i \bmod N$. Taking $m = \frac{\tau}{\log \alpha}$ suffices for soundness $2^{-\tau}$, once the verifier has determined that $N$ has no factor smaller than $\alpha$. In practice, setting $\alpha \approx 10,000$ and using 10 repetitions (in the non-interactive setting) provides good efficiency, and suffices for computational security $\kappa = 128$.
While Goldberg et al. [26] introduced the use of $\alpha$ for improved efficiency, they in turn cite Gennaro et al. [25] for proofs of completeness, soundness and zero knowledge. We will use $\mathcal{S}_{\mathsf{GCD}}$ to refer to the simulator that they describe in Section 3.1, when demonstrating that the construction below is zero knowledge.
$\mathcal{R}$ queries the challenger to receive challenge $N_1^*, C_1^*, X^*$, and plays the part of $\mathcal{S}$, interacting with $\mathcal{A}$. $\mathcal{R}$ simulates key generation by running the honest protocol. It stores $X, x_1$ for use in what follows.
Until the $i$th event, when $P_2$ is malicious during refresh, and each time there is a new corruption of $P_2$, $\mathcal{R}$ uses its knowledge of $x_1$ (which might be updated during refresh procedures) to run the protocol honestly. In particular, it constructs $C = \mathsf{Enc}_N(x_1)$ honestly. Note that it still uses simulated proofs $\tilde{\pi}$ and $\widetilde{\Psi}$.
In the $i$th event, $\mathcal{R}$ uses its challenge to construct $(C_1^*, N_1^*, \tilde{\pi}, \widetilde{\Psi})$, using simulated proofs $\tilde{\pi} \leftarrow \mathcal{S}_{\mathsf{GCD}}(N^*)$ and $\widetilde{\Psi} \leftarrow \mathcal{S}_{\mathsf{eq}}(N_1^*, C_1^*, X^*)$; this is done whether the $i$th event is an execution of refresh

$\Pi_{\mathsf{GCD}}$

**Inputs:** Prover: $N, \phi(N)$.
Verifier: $N$.

**Prover:** Compute: $y_1, \ldots, y_m = H(N)$.
Compute: $d = N^{-1} \bmod \phi(N)$.
Compute: For $i \in \{1, \ldots, m\}$, $t_i = y_i^d \bmod N$.
Send: $t_1, \ldots, t_m$.

**Verifier:** Verify: $N$ is positive with sufficient bit length.
Compute: $y_1, \ldots, y_m = H(N)$.
For $i \in \{1, \ldots, m\}$, verify that $\mathsf{GCD}(y_i, N) = 1$ ; $y_i = t_i^N \bmod N$.

Figure 9: Non-interactive Zero Knowledge proof for the language $L_{\mathsf{GCD}}$, due to Goldberg et al. [26]. We present the protocol in the random oracle model, using the Fiat-Shamir transform.

with malicious $P_2$, or a new corruption of $P_2$ during refresh or sign. In the latter case, $\mathcal{R}$ uses $X^*$ when simulating $X_1$ in $\mathsf{state}_2$.
In the $i$th refresh command prior to the next de-corrupt command, $\mathcal{R}$ queries its challenger and receives $N_i^*, C_i^*$. It updates $X^* = X^* + (r \times G)$, where $r \leftarrow [q]$ is the simulated output of $\mathcal{F}_{\mathsf{coin}}$ created by $\mathcal{R}$. It simulates $P_1$'s message using $(N_i^*, C_i^*, \tilde{\pi}, \widetilde{\Psi})$. In any sign commands prior to the next de-corrupt command, $\mathcal{R}$ verifies the correctness of $C'$ sent by $P_2$ by using the current instance of the Paillier-EC-Refresh oracle: it extracts $k_2$ and $x_2$ from $P_2$'s calls to $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R_{DL}}}$, sets $\alpha = k_2^{-1} m \bmod q$ and $\beta = k_2^{-1} r x_2 \bmod q$, and submits query $(C', \alpha, \beta)$ to its oracle $\mathcal{O}_{C_{i^*}}$.
In the remaining events, $\mathcal{R}$ proceeds as $\mathcal{S}$ does (and as $\mathcal{R}$ did in Hybrid $\mathsf{H}_1$). If the challenge bit $b$ in the Paillier-EC-Refresh game is 1, then $\mathcal{R}$ produces the same distribution as in Hybrid $\mathsf{H}_2^{(i-1)}$. Otherwise, it produces that of Hybrid $\mathsf{H}_2^{(i)}$.