The Mathematical Engine of CGGMP21: A Look Under the Hood

The CGGMP21 protocol is a sophisticated symphony of modern cryptographic techniques designed to solve a complex problem: how to create a standard Elliptic Curve Digital Signature Algorithm (ECDSA) signature when the private key is split among multiple parties and can never be reconstructed. This is achieved by combining several powerful mathematical tools, primarily **Verifiable Secret Sharing**, **Homomorphic Encryption**, and **Zero-Knowledge Proofs**.

Here is a breakdown of the mathematical concepts and how they fit together in the CGGMP21 protocol.

1. The Foundation: Standard ECDSA

First, let's recall the standard ECDSA signature equation. A signature is a pair of values, (r,s), calculated as follows:

- **Select a random nonce:** A secret random number, k, is chosen.
- **Calculate a point on the curve:** $R=k \cdot G$, where G is the generator point of the elliptic curve.
- **Determine r:** r is the x-coordinate of the point R.
- **Calculate s:** $s=k-1(H(m)+r \cdot x)(modq)$

Where:

- x is the private key (a large integer).
- H(m) is the hash of the message to be signed.
- q is the order of the curve's generator point.

The core challenge for a Threshold Signature Scheme (TSS) like CGGMP21 is to compute the signature (r,s) without any single party ever knowing the full private key x or the full random nonce k.

2. Distributed Key Generation (DKG)

The first step is to create the split private key without ever assembling it. This is done through a Distributed Key Generation (DKG) protocol.

- Secret Sharing: CGGMP21 uses a technique called Verifiable Secret Sharing (VSS), often based on Feldman's VSS. Each of the n parties generates a secret random polynomial of degree t−1 (where t is the threshold). For instance, party i creates:
  $fi(z)=ai,0+ai,1z+ai,2z2+\cdots+ai,t-1zt-1$
  The term ai,0 is party i's initial share of the main private key.
- Commitments: To prevent cheating, each party broadcasts commitments to the

coefficients of their polynomial by publishing points on the elliptic curve:
$C_{i,j}=a_{i,j}\cdot G$
These commitments allow other parties to verify the shares they receive without revealing the secret coefficients.

- **Share Distribution:** Each party i then securely sends a value $f_i(j)$ to every other party j.
- Final Key Shares: Each party j can now compute their final share of the private key, x_j, by summing the shares they received from all other parties:
$x_j=\sum_{i=1}^{n} f_i(j)$
The global private key is $x=\sum_{i=1}^{n} a_{i,0}$, but it is never calculated. The corresponding global public key, $X=x\cdot G$, can be computed publicly by anyone by summing the initial commitments: $X=\sum_{i=1}^{n} C_{i,0}$.

At the end of this process, each party holds a share x_i of the private key, and the global public key X is known, but the global private key x remains a secret.

3. Homomorphic Encryption: The Secret Calculator

A crucial part of the ECDSA equation is the multiplication $r\cdot x$. Since both r and x will be secret-shared values, a special tool is needed to perform this multiplication without revealing the shares. This is where **additively homomorphic encryption**, typically the **Paillier cryptosystem**, comes into play.

The key property of Paillier encryption is:
$Enc(m_1)\cdot Enc(m_2)=Enc(m_1+m_2)$
This allows for addition of encrypted plaintexts by multiplying their ciphertexts. This property is the backbone of a sub-protocol called Multiplication-to-Addition (MtA). MtA allows parties to collaboratively compute shares of a product (e.g., $k_i\cdot x_j$) without revealing their individual secret shares (k_i or x_j).

4. The Distributed Signing Process

With the DKG and homomorphic encryption in place, the parties can now collaboratively sign.

1. **Distributed Nonce Generation:** Similar to the DKG for the private key x, the parties use a similar process to generate shares, k_i, of the secret nonce k.
2. **Compute r:**
    - Each party computes their public share of the nonce point: $R_i=k_i\cdot G$.
    - The global nonce point R is the sum of these public shares: $R=\sum R_i$.
    - All parties can now compute the value r, which is the x-coordinate of R.
3. **Compute s (The Core of the Protocol):**
    - Recall the equation: $s=k^{-1}(H(m)+r\cdot x)$. We can rewrite this as $s\cdot k=H(m)+r\cdot x$.
    - The goal is for each party i to compute a share σ_i that can be combined to form the final signature component s. Let's set $\sigma=k^{-1}\cdot x$. The equation becomes $s=k^{-1}H(m)+r\cdot\sigma$.
    - The parties hold shares k_i and x_i. Using the **MtA protocol** (powered by Paillier encryption), they interact to compute shares of the cross-products needed for the terms $r\cdot x$ and k^{-1}. This is the most complex part of the protocol, involving multiple

rounds of communication where parties exchange encrypted values and perform homomorphic operations.
    - At the end of the MtA and other interactions, each party i can compute a local signature share, si. This share is a linear combination of their private key share xi, their nonce share ki, the public message hash H(m), and values derived from the MtA protocol.
    - The final signature value s is simply the sum of all the shares: $s=\Sigma si (mod q)$.

5. Zero-Knowledge Proofs: Enforcing Honesty

Throughout the DKG and signing processes, how can a party be sure that others are following the rules? For example, during the MtA protocol, how do we know a party used their correct share inside the homomorphic encryption?

This is accomplished using **Zero-Knowledge Proofs (ZKPs)**. At various steps, each party generates a ZKP to prove that the values they are broadcasting were computed correctly, without revealing the secret inputs to those computations. For example, a party will provide a ZKP to prove:

- That their initial DKG commitments correspond to a valid secret polynomial.
- That their encrypted inputs to the MtA protocol were formed correctly.
- That they know the private key share corresponding to their public key share.

If a party provides an invalid proof, they are immediately identified as malicious or faulty. This ability to pinpoint the source of an error is the "Identifiable Abort" feature that makes CGGMP21 so robust.

In summary, CGGMP21 ingeniously weaves together secret sharing, homomorphic encryption, and zero-knowledge proofs to perform calculations on distributed secrets. This allows a group of parties to collectively wield the power of a single private key, creating standard and verifiable ECDSA signatures without ever exposing that key to a single point of failure.