**System Design: Node/Validator health check:**

**Objective :**

The primary aim of this system is to monitor the health of validator nodes and identify if any of them are potentially compromised. The health is determined based on a series of challenges that each validator node has to answer correctly.

**Components :**

1. Validator Node :
   1. Represents a participant in the network.
   2. Has a unique name, private key, and public key.
   3. Responds to various challenges that test its validity.
2. Challenges :
   1. Mathematical: Simple arithmetic challenge.
   2. String Manipulation: Reverse a string.
   3. Hash: Generate a SHA256 hash.
   4. Time: Report the current time.
   5. Cryptographic Signature: Sign a message with a private key.
3. System Drivers :
   1. Script to simulate all validator nodes responding to challenges.
   2. Logic to detect potentially compromised validators.

**Workflow :**

1. Initialization :
   1. Create validator nodes with their private and public keys.
   2. Initialize an empty voting data for each validator.
2. Challenge Generation :
   1. System generates a random challenge for each type for each validator node.
   2. Validators respond to these challenges based on their type.
3. Response Evaluation :
   1. Each validator's response to a challenges is recorded.
   2. Eve, for simulation purposes, is programmed to fail challenges in order to test the system's detection capabilities.
4. Compromise Detection :
   1. Validators that consistently fail or pass challenges at a suspiciously high rate are flagged as potentially compromised.
5. Report Generation :
   1. All challenge responses are printed out.
   2. Potentially compromised validators are identified and saved into a file.

**Key Features :**

1. Robustness : By employing multiple challenge types, the system makes it difficult for a compromised node to consistently pass the health check.
2. Extensibility : New challenge types can easily be introduced into the system for added security layers.
3. Security : The cryptographic signature challenge leverages RSA keys, ensuring responses cannot be forged without access to the validator's private key.
4. Transparency : All challenges and their expected vs. actual responses are printed, allowing for easy manual verification if needed.

**Limitations** :

1. Simulated Environment : The current setup mainly serves as a prototype or a simulation, which means in a real-world scenario more features and security checks will be needed.
2. Hardcoded Failure : The system currently forces the "Eve" validator to fail for simulation. In an actual system, validators would be operating without such interferences.

**Recommendations for Improvement :**

1. Audit Trail : Maintain a more detailed audit trail for each validator, logging not just the responses but also other meta-data such as response time, IP address, etc.
2. Additional Challenge Types : Include more sophisticated challenges, like network-based challenges, memory integrity checks, or hardware-level validations.

**Conclusion :**

The current design provides a foundational framework for checking the health of validator nodes. While it successfully identifies simulated compromise scenarios, for real-world applications, the design will benefit from enhancements and continuous iterations.