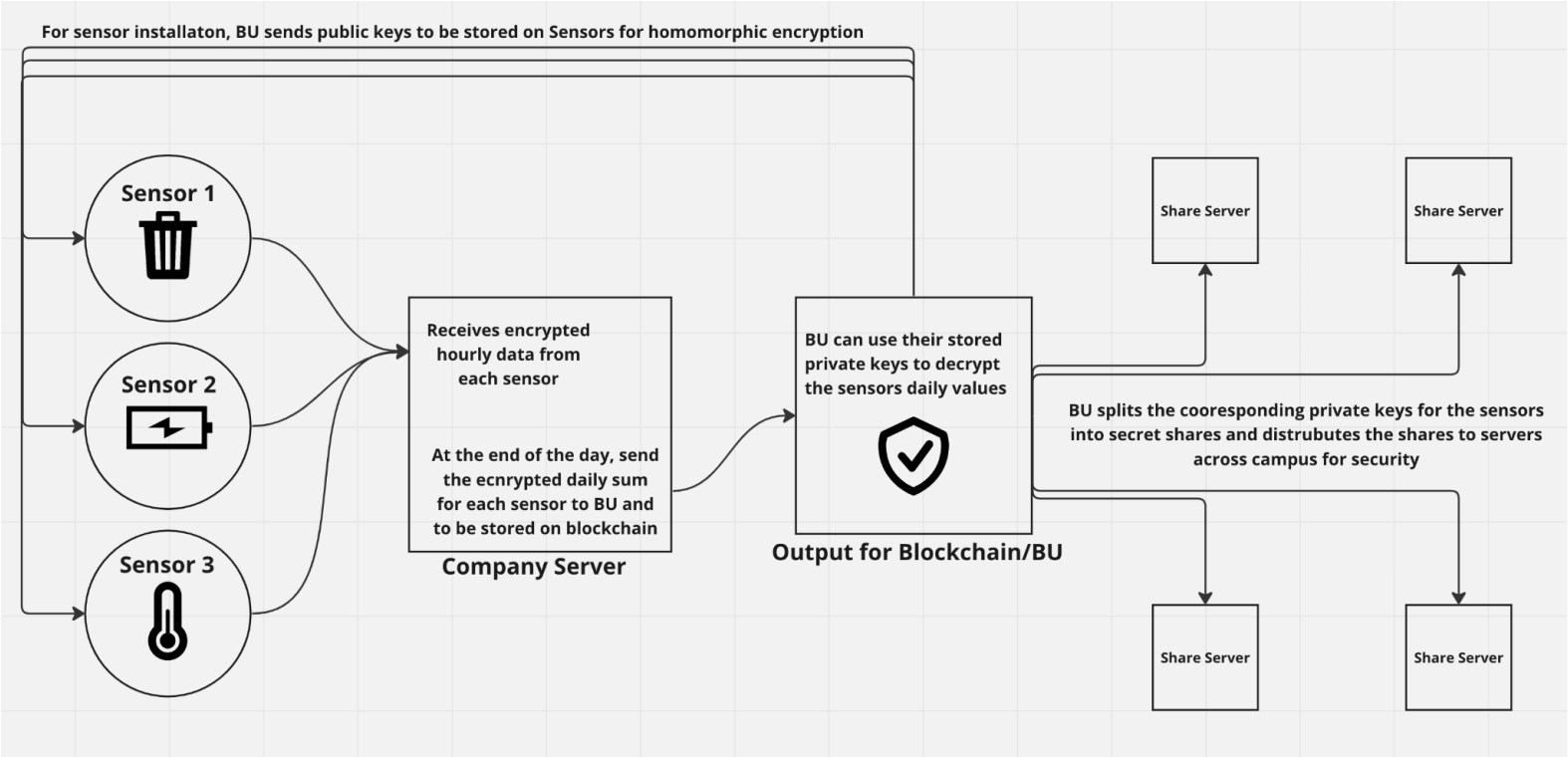
## Contributions:

**Max Karambelas:** Helped with brainstorming all aspects of the design. Wrote the report outlining our design process regarding our sensors and server communication. Wrote the code for our company server and the code for the interaction between the company server and the sensors. Edited the code for the company sensors.
**EJ Wong:** Helped with brainstorming all aspects of the design. Wrote the code outlining the logic of the sensors. Wrote the ZKP portion of the report and edited all parts. Edited the code for communication between the sensors and server.
**George Trammell:** Helped with brainstorming all aspects of the design. Wrote the ZKP portion of the report and edited all parts. Wrote the code for the ZKP and worked out all the logic behind how the ZKP was going to function and how it would integrate into our design.

## Sensor Manufacturer Diagram:




## Sensor Manufacturer Outline:

### Public Key and Secret Key Generation and Security.

To begin the implementation of the sensors, BU must first construct a public/private key pair for each sensor on the network. These keys will be generated using homomorphic encryption — more specifically, the Paillier cryptosystem — which

will be explained later when the sensor logic and company server logic is outlined. BU will then take each secret key and, using Shamir secret sharing, disperse them throughout servers on the BU campus. This will ensure that even if someone was to access one of the internal servers, they would only get pieces of the secret keys and not the full keys themselves. It would also ensure that only BU has access to the secret keys and therefore will be the only party able to decrypt sensor data. All keys will be securely stored on each sensor in the CDS building so that the sensors can encrypt their values to send to our companies servers. BU will also be advised to keep track of which sensor private keys correspond to their public keys, mainly so that when they decrypt sensor data they will know which sensor the data originally came from. By securely storing the public keys on the sensors themselves, we can protect against third parties using these public keys to encrypt data and pass it off as data sent from the sensors.

### Sensor Logic.

The sensors receive data every minute and collect this data as a running sum. This running sum will be saved every hour and secured using homomorphic encryption with the public key generated by BU that was installed on that sensor. This security will ensure that all data being sent from the sensors will be encrypted and therefore cannot be unlocked by anyone not in possession of all the secret shares of the corresponding secret key mentioned in the security protocol above. This encrypted hourly summed data is then sent over to our company's server for further calculation. The purpose of only having the sensors compute hourly data rather than daily totals is to limit the amount of computing power expended by the sensors. This would also allow these sensors to reset their data and start with an empty dataset every hour, minimizing the amount of data storage that sensor will need to hold. Furthermore, when we receive this hourly data from the sensors we won't know the data itself but also we won't be able to tell what sensor or type of sensor it came from. We also cannot see which public key was assigned to which physical sensor — only BU has access to this data.

### Company Server Logic.

Our company server will take in these hourly encrypted totals from each sensor and keep a running total for each sensor for the duration of the day. At the end of each 24-hour day, our server will have an encrypted daily total for each sensor in the building and will send these totals to BU so they can have the daily total of every sensor in the building. Our use of homomorphic encryption allows for our company to take these hourly totals from the sensors and compute the sum of these values to obtain daily totals without ever knowing the hourly or daily totals themselves. This will protect the data even from us, as without the private keys that BU generated for the sensors, there is no way for us to decrypt or trace any of the data the sensors send to us.

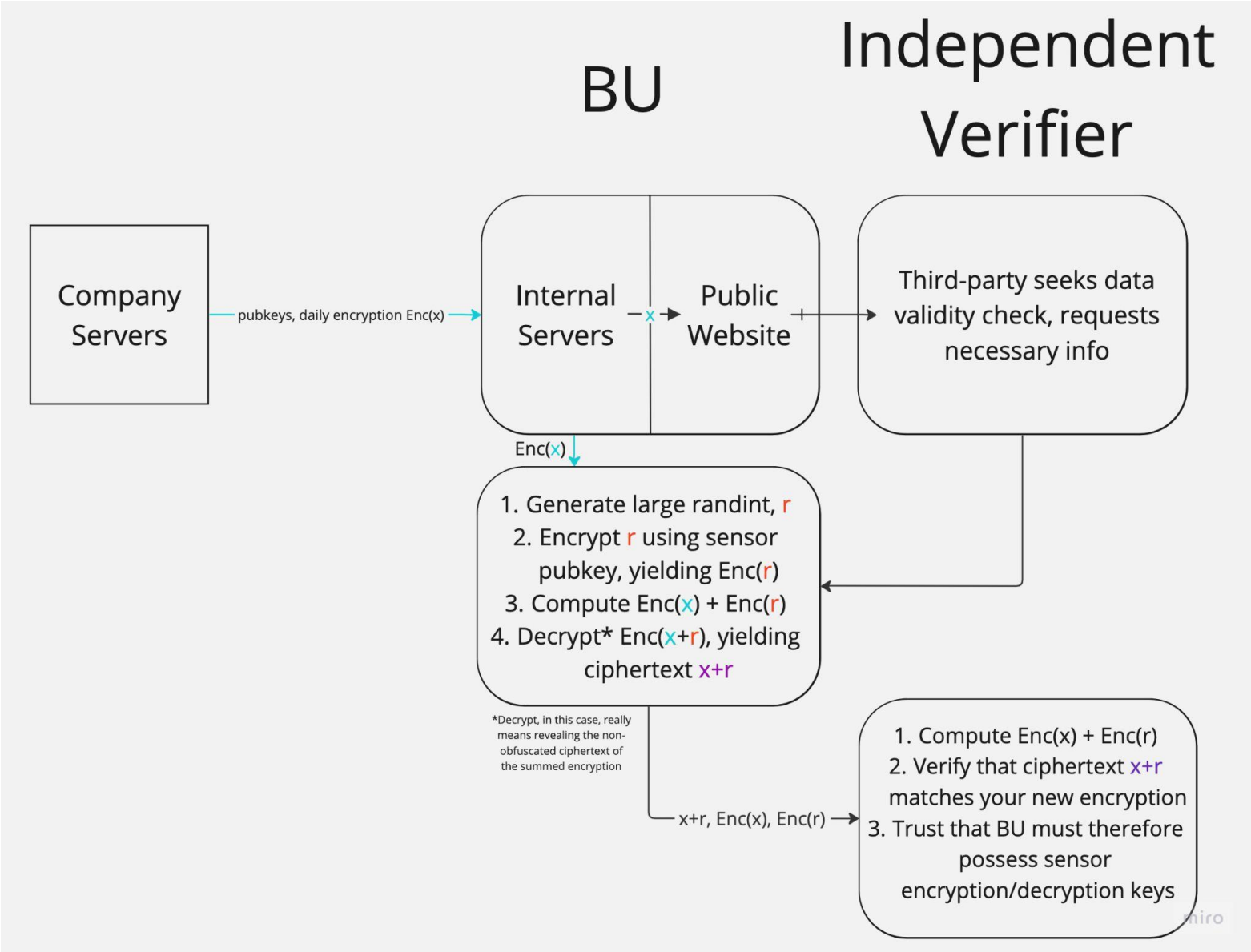
## Final Output.

Once the day ends, BU will receive an encrypted daily total for each of the sensors in the building. They won't know the order in which they receive the sums and which sensors they correspond to initially because we as a company don't know which sensors the data is coming from either. But, BU can simply collect their initial Shamir secret shares to reconstruct their private key values and then run a simple loop over the daily sums, trying each key on each sensor until they are all unlocked. Once each encryption is unlocked, BU can then determine the sensor it came from by seeing what secret key was used to unlock it. This is why upon the generation of these secret keys in the beginning, we told BU to remember which sensor the corresponding public keys went to. With this method, not just the data itself but even its origin is unknown throughout this process until the very end when BU unlocks the sensors' summed data. Finally, BU can post this data to a university website for public or monitored access and prepare to counter any suspicion with the proof of validity below.

## Verifying Data Authenticity with a Zero Knowledge Proof.

*This explanation corresponds to the diagram below.*

To prove to the rest of the world that the data on our website truly came from our sensors, we will use a special homomorphic case of a Zero Knowledge Proof. First, an independent third party must request proof that our data is valid by using a simple script made publicly available by BU. BU will then generate a random integer *r* for each sensor and encrypt them with the sensors' corresponding public keys. This encrypted *r* will be subsequently used as an element in the homomorphic equation used on the similarly encrypted data that BU obtains from the sensors (which we will call *x*). It is important that *r* is encrypted with the same public key as its sensor's data as it both allows for homomorphic operations and proves that BU contains the matching sensor public key needed to encrypt more data. After generating a new homomorphic encryption based on the summation of the encrypted sensor data and *r*, BU then decrypts this result to reveal non-obfuscated ciphertext x+r. BU would finally send this ciphertext to the verifier along with the sensor's encrypted data and BU's newly encrypted *r*. Although sharing non-obfuscated ciphertext with a third party is generally unsafe, the previous homomorphic addition of an encrypted *r* value creates the layer of security necessary to prevent third parties from gaining any insight into the data. Finally, the verifier can conclude the script which independently computes the homomorphic encryption value and checks it against BU's. If the assertion is successful, the third party now knows that BU must have the ability to both encrypt and decrypt sensor data.




## Citations:

Paillier Encryption Methods Source for Python:
https://python-paillier.readthedocs.io/en/stable/phe.html
Paillier Encryption Discussion:
Paillier, Pascal. (2005). Paillier Encryption and Signature Schemes. Encyclopedia of Cryptography and Security. 10.1007/0-387-23483-7_293.