Project Report

Naga Srihith Penjarla Nandakishore S Menon
Mayank Kabra

Idea

This project aims to securely perform DNS query resolution in IoT devices using Secure Multi-party Computation protocol. We need 3 IoT devices in the same network (LAN) to run this protocol. The idea is to give each of the three devices a share of the key used for encryption/decryption of DNS queries/responses. As no single device owns the key entirely, it prevents a single-point-of-compromise.

This project uses the ***Paillier*** cryptosystem for IoT devices to communicate with the DNS server. It is a homomorphic public-key cryptography algorithm. The DNS server has a public-private key pair. The public part is known to all the clients. An IoT client who needs to query the resolver generates a symmetric key (AES-128) and encrypts this key with the resolver's public key, and sends it to the resolver. The resolver now learns the client's symmetric key by decrypting it using its private key. Thus, the resolver and the client can decrypt/encrypt the DNS queries/responses using the symmetric key.

The symmetric key that the client will send to the resolver is distributed among the three IoT devices. For this project, we use a 3-Party Computation protocol named ASTRA. Each party generates a part of the key, and the final key is the sum of the keys generated by the parties. Once each party generates their share of the key, they secret share it according to the semantics of the ASTRA protocol.

Whenever a device has a query, it secret shares the query with the other devices, and then using the ASTRA's shared circuit evaluation method, they encrypt the query. A similar process is done at the time of decrypting the response got from the server.

Deliverables

Following is a list of deliverables of this project:

- DNS Resolver
- DNS Client for IoT devices

Technologies

- C/C++
- Socket Programming

Implementation Details

DNS Resolver:

- The code for the DNS recursive resolver has been written in C/C++ programming language. The resolver listens on port 53 for incoming DNS queries from DNS clients.

- Encryption - The queries/responses sent/received are encrypted using AES-128-bit encryption. The AES-128-bit key is initially received from the client using homomorphic public-key cryptography. We have used the ***Paillier*** cryptosystem for this purpose.

DNS Client for IoT devices:

Symmetric Encryption

- Each of the three devices in our multi-party arrangement (here, 3-party) randomly produces a key K_i. The final key K is created by ADDing every K_i. $K = K_1 + K_2 + K_3$, which is further encrypted using the resolver's public key.

- The shares of the key are generated using a 3-party protocol, ASTRA and the symmetric-key protocol AES (Advanced Encryption Standard). The key K is 128-bits as AES takes 128 bits as input and outputs 128 bits of encrypted ciphertext. AES relies on the substitution-permutation network principle, which is performed using a series of linked operations involving replacing and shuffling the input data.

- The symmetric key generation takes place in a sequence of steps:

    1. **SubBytes:** Each byte is substituted with another byte in this phase. It utilises a lookup table commonly known as the S-box.
    2. **ShiftRows:** In this step, each row is shifted a particular number of times.
    3. **MixColumns:** This process essentially involves multiplying matrices. Each column is multiplied by a particular matrix, which changes the order of each byte in the column.

4. **<u>Add Round Key:</u>** The resultant output of the previous stage is XOR-ed with the corresponding round key.

Symmetric Decryption

- The decryption proceeds in the opposite direction to it, which, when performed, reverts the changes.

- All the encryption and decryption steps are performed in a shared manner, following the sharing semantics of ASTRA.

Asymmetric Encryption

- For the client group to perform DNS queries, the session key needs to be generated, encrypted and communicated with the DNS Resolver for each session.

- The encryption of the session key is typically done using an asymmetric encryption protocol where the DNS resolver's public key is used to encrypt the session key generated by the client. This encrypted key is sent to the resolver, which decrypts it and uses it to decrypt each of the queries during that session.

- In our implementation, we are using the Paillier cryptosystem to perform the session key encryption. The Paillier cryptosystem is a additive homomorphic asymmetric encryption algorithm used for public key cryptography. Let's say keys k_1, k_2 are encrypted to get $P(k_1)$ and $P(k_2)$ using Paillier cryptosystem, then

$P(k_1 + k_2) = P(k_1) + P(k_2)$. We are leveraging the additive homomorphic nature of the Paillier cryptosystem to generate the session key and communicate it with the resolver.

- Key generation: A 128 bit key, K_i is generated by each of the parties within the client system and the session key would be the sum of each of these keys ($K_0 + K_1 + K_2$). Let p_0 be the party initiating the query. Each party p_i will generate their corresponding K_i and compute $P(K_i)$ using the public key of the resolver. Then parties p_1 and p_2 will send their encrypted keys to p_0. p_0 will sum their encrypted keys with its own encrypted key and compute $P(K_0 + K_1 + K_2)$ which is the encryption of the session key. This result is then sent to the DNS resolver, which can decrypt it using its private key to obtain the session key.

- In order to perform the Paillier encryption, we have used the libhcs library. The public key consists of a public key p_k and a random initialization hcs_{random}. These are made publicly available by the resolver. The libhcs implementation of Paillier cryptosystem required the random initialization for encryption using the public key. However the same may **not** be necessary for other implementations. The shares K_i possessed by each of the client parties are the shares that will be used for encrypting the queries using MPC.

Technical Details

The functions which play an essential role in this implementation are discussed below.

Symmetric: The entire encryption and decryption have been done in 2 phases, i.e. an offline phase and an online phase. The offline phase, also known as the preprocessing phase, generates the random shares independent of the input value. Once the parties give their inputs, the online phase begins (as this is an input-dependent phase). The preprocessing is done to save time.

- **Encryption**: The encryption is majorly done using the following functions:

1. offline_phase(): Runs the offline part of the encryption as per the ASTRA protocol, where the shared randomness data is pre-computed.

2. online_phase(): Runs the online part of the encryption as per the ASTRA protocol, where private inputs of the parties are used along with the pre-computed randomness.

3. offline_secret_share(): Generates common randomness as per the ASTRA protocol for each party.

4. online_secret_share(): The querying party secret shares its query with the other parties using the previously computed random data.

5. SBox(): Evaluates the AES s-box circuit (Boyar-Peralta) in a secret shared fashion. Each gate called within the function performs operations specified by ASTRA. It is the only non-linear circuit in the whole AES.

6. shiftrow(): Performs the AES shiftrow operation in a secret shared fashion. As it contains all linear operations, parties don't need to interact and can do the computation locally.

7. mixcolumns(): Performs the AES mix-column operation in a secret shared fashion. It is also a local evaluation.

8. key_expansion(): Used for generating keys for each round. It is non-linear as it makes use of the SBox.

9. AND(): Performs the AND operation as per the semantics of the ASTRA's offline AND protocol.

10. AND_online(): Performs the AND operation as per the semantics of the ASTRA's online AND protocol.

11. reconstruct(): Reconstructs the shared value. Each party sends a part of its share to the querying party following a cyclic order.

- **Decryption**:

1. inverse_SBox(): Evaluates the AES inverse s-box circuit in a secret shared fashion. Each gate called within the function performs operations specified by ASTRA. It is a non-linear circuit.

2. inverse_shiftrow(): Performs the AES inverse shiftrow operation in a secret shared fashion. As it contains all linear operations, parties don't need to interact and the computation can be done locally.

3. inverse_mixcolumns(): Performs the AES inverse mix-column operation in a secret shared fashion. It is also a local evaluation.

4. inverse_key_expansion(): Used for generating keys for each round. It is non-linear as it makes use of the inverse SBox.

5. AND(): Performs the AND operation as per the semantics of the ASTRA's offline AND protocol.

6. AND_online(): Performs the AND operation as per the semantics of the ASTRA's online AND protocol.

7. reconstruct(): Reconstructs the shared value. Each party sends a part of its share to the querying party following a cyclic order.

Asymmetric:

The asymmetric operations are majorly done using the following functions:

1. get_pub_key(): Get the public key of the DNS resolver

2. gen_key(): Each party generates its share of the session key.

3. ecnrypt_key(): Each party encrypt their share of the key using the public key of the DNS resolver.

4. receive_keys(): The querying party receives the other parties' encrypted shares of the session keys.
5. add_keys(): Add all the encrypted session keys. As the Paillier cryptosystem is homomorphic, adding the encryption of the shares of the keys will result in the encryption of the sum of the shares.

6. send_key(): Shares the encrypted session key to all the parties and send it to the DNS resolver.

Resolver:

1. lookup(): This function works to gather authoritative answers to the client's query from the nameserver.

2. recursive_lookup(): This function recursively calls the lookup function to perform the name resolution.

3. decrypt_key(): Using the private key, decrypt the session key sent by the DNS client.

Challenges Faced

1. Performing asymmetric encryption using MPC - MPC protocols securely evaluates functions represented in the form of arithmetic/boolean circuits. But, for the asymmetric encryption, we could not find any state-of-the-art MPC protocol. We decided to use the RSA cryptosystem but could not find any efficient

implementations of MPC protocol evaluating the RSA algorithm in a privacy-preserving fashion. That is why we decided to use homomorphic public-key cryptography.

2. Pailier cryptosystem: The library used for performing encryption and decryption, as per Paillier cryptosystem, required a random seed for generating the public key. The encryption function also requires this random key of this library. Since the key generation is done at the Resolver's end and the client does the encryption, the seed had to made publicly available.

Active Issues

1. DNS resolver - The resolver is not able to resolve all the queries. There are some queries for which the resolver returns only the *Alias* name. We are trying to debug this issue using packet analysis tools.