USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Qelect: Lattice-based Single Secret Leader Election Made Practical

Yunhao Wang and Fan Zhang, *Yale University*

https://www.usenix.org/conference/usenixsecurity25/presentation/wang-yunhao

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Qelect: Lattice-based Single Secret Leader Election Made Practical

Yunhao Wang
Yale University

Fan Zhang
Yale University

A Artifact Appendix

A.1 Abstract

Our artifact is a C++ library implementing the main single secret leader election protocol Qelect in [2]. Our main results aimed to be produced from this artifact are 1) the local computation time with party size in $\{32, 64, 128, 256, 512, 1024, 2048\}$, and 2) the communication time of 128 parties under both LAN and WAN settings. Our main claim in [2] (which is the same as in the submission version) uses the party size $G = 128$ as in section 7.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact should not pose any risk to the evaluators regarding security, or data privacy. None does our artifact has any ethical concerns. The testing data, key pairs and communications are all simulations with no real impact. Despite all these, we still recommend the evaluators start a fresh AWS EC2 instance without testing our code using their own machines.

A.2.2 How to access

Our code is public on Zenodo, which is also the version in our final submission.

A.2.3 Hardware dependencies

Our main runtime results should be reproducible on AWS EC2 c6i.2xlarge instances with 8 vCPU, 16GB RAM and Intel Xeon Scalable processors. Note that we do not make an explicit assumption on the network latency, but estimate the communication time based on the EC2 network directly. Corresponding latency figures are also documented in section 7 of our final submission (same as in the full version [2]).

A.2.4 Software dependencies

On a standard AWS EC2 c6i.2xlarge instance, we run the benchmarks with boot disk configured with Ubuntu 24.04 LTS operating system.

We also rely on the following software and libraries:

- C++ build environment
- CMake build infrastructure
- Docker environment to run our simulation
- SEAL library 4.1 (or latest version) and all its dependencies
- PALISADE library release v1.11.9 and all its dependencies

A detailed installation script is provided in the README.md file in our artifact. Again, all secrets and data are simulated, and thus no third-party models/datasets are used.

A.2.5 Benchmarks

We benchmarked our main SSLE protocol Qelect presented in section 6 in [2] (and as in our final submission version). The party size we provided to be tested for evaluators in the docker is $G \in \{32, 64, 128, 256, 512, 1024, 2048\}$, which covers our main claim w.r.t. party size $G = 128$ as in section 7.

A.3 Set-up

A.3.1 Installation

The installations and tests could be broken into two parts. One for *local computation* and the other for *communication simulation* between multiple parties. To recover the runtime of our major claim w.r.t. the party size $G = 128$, we first show how to launch such 128 AWS instances with proper configurations. Notice that later we would also simulate the local computation for $G \in \{32, 64, 128, 256, 512, 1024, 2048\}$, while the communication time will be estimated only for $G = 128$ under both LAN and WAN settings.

Instance configuration

1. In AWS console, under the region AWS us-east-2 (Ohio), create key pair named ssle_us_east, which will automatically get its pem file (rename to ssle.pem) downloaded to local.

2. In AWS console, under other three AWS regions: us-west-1 (California), eu-west-1 (Ireland), and ap-southeast-1 (Singapore), import the previously downloaded ssle.pem file to the corresponding ssle_us_west, ssle_eu_west, ssle_ap_southeast (so that the key pairs under all four regions are actually the same). This step is to guarantee that the evaluators could be able to control all instances simultaneously, in order to simulate the broadcast in between.

3. In AWS console, under four AWS regions: AWS us-east-2 (Ohio), us-west-1 (California), eu-west1 (Ireland), and ap-southeast-1 (Singapore), create a launch template with the following configuration:

   - Application and OS Images: ubuntu 24.04, x86_64
   - Instance type: c6i.2xlarge
   - Key pair: select the corresponding ssle_[region] key pair just created
   - Network setting: create security group and corresponding VPC that allows for any IPv4 address in both the inbound and outbound rules.

4. In AWS console, launch 128 instances under AWS us-east-2 (Ohio) (this will be used to simulate the communication under LAN); launch 32 instances under all other three regions respectively. In generally, with G, uniformly distribute the instances across all four regions to simulate the communication under WAN setting.

Local computation setup Ssh to one of the instances under AWS us-east-2 (Ohio) Note that the following installation only needs to be performed on a single AWS EC2 instance.

```
# install docker
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo curl -fsSL https://download.docker.com \
/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo
"$UBUNTU_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list
> /dev/null

sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli
containerd.io docker-buildx-plugin
docker-compose-plugin
# execute the MPE
sudo apt install -y unzip
wget https://zenodo.org/records/14854598/files/\
    wyunhao/Qelect-v3.2.zip
unzip Qelect-v3.2.zip
cd <directory_name_unzipped>
sudo docker build --no-cache -t ssle_project .
```

Communication setup To recover our estimated communication time under both LAN and WAN settings, we also need to equip all launched instances with proper files. Looking ahead, we simulate the communication time by letting them publish messages prepared beforehand. The following steps are for WAN setting. One could follow exactly the same procedure but only under AWS us-east-2 (Ohio) region for LAN setting.

1. In AWS console, under four AWS regions, execute the command in CloudShell (notice that this is a single-line command):

   ```
   aws ec2 describe-instances  --filters
   "Name=instance-state-name, Values=running"
   --instance-ids $instance_id  --query
   'Reservations[*].Instances[*].PublicIpAddress'
   --output text
   ```

2. Copy all 128 IPv4 public addresses returned from AWS CloudShell and put them in a single "ip.txt" file, one IPv4 address a line. Note that in LAN setting, we would have 128 IPv4 addresses directly for AWS us-east-2 (Ohio) region. Thus, when testing against WAN setting, pick 32 out of them.

3. Upload the "ip.txt" file from local to all instances via the following single-line command:

   ```
   pscp -p 100 -h ip.txt -x "-i  ssle.pem -o
   StrictHostKeyChecking=no" -l ubuntu ip.txt .
   ```

4. Dump random bytes to simulate the ciphertexts[1] used for broadcasting to all:

   ```
   dd if=/dev/urandom of=1M.txt bs=1M count=1
   ```

   And then also upload this simulated data to all instances:

   ```
   pscp -p 100 -h ip.txt -x "-i  ssle.pem -o
   StrictHostKeyChecking=no" -l ubuntu 1M.txt .
   ```

5. Upload the ssle.pem file from local to all instances via the following single-line command:

   ```
   pscp -p 100 -h ip.txt -x "-i  ssle.pem -o
   StrictHostKeyChecking=no" -l ubuntu ssle.pem .
   ```

[1] For $G = 128$, we have the communication size to be around 983KB. We use 1M data for simplicity.

6. Ssh to one of the instances and execute the following:

```
sudo apt-get update
sudo apt-get install -y build-essential
sudo apt -y install pssh
```

A.3.2 Basic Test

After installation, it should be easy to test the local computation, the evaluators could run the command:

```
After param generation.
Initial noise budget: 613
Lefted noise budget: 42
Total broadcast communication size ...: 67 MB.
--------------------------------
Total number of parties: 32
Preprocessed time      : 20446144 us.
Total time             : 2429615 us.
```

It shows that the local computation for one of the 32 parties takes around 2.4 seconds. The full log of our docker execution should include runtime for party sizes in $\{32, 64, 128, 256, 512, 1024, 2048\}$.

A.4 Evaluation workflow

The major results we want to reproduce here is the local computation and communication time for party size $G = 128$. To show evaluators the runtime trend, we consider a range of party sizes aforementioned.

A.4.1 Major Claims

(C1): The local runtime of $G \geq 2048$ is less than 10 seconds.

(C2): The communication time for $G = 128$ under LAN setting is around 25 seconds, and around 42 seconds under WAN setting. Since our communications happen in parallel, while the major prior work [1]'s communications run sequentially, we claim that our protocol is at least two orders of magnitude faster than theirs.

A.4.2 Experiments

(E1): *[Local Runtime] [<1 human-minute]*:

Preparation: Ssh to the instance with docker installed and execute the command

Execution: `sudo docker run ssle_project`

Results: The last log block should look like the following:

```
After param generation.
Initial noise budget: 613
Lefted noise budget: 47
Total broadcast communication size ... 4295 MB.
--------------------------------
Total number of parties: 2048
Preprocessed time      : 191495790 us.
Total time             : 6745102 us.
```

(E2): *[LAN Communication] [<1 human-minute]*:

Preparation: Follow the same steps as in Appendix A.3.1 when preparing for the communication tests, but use all 128 instances under the same region: AWS us-east-2 (Ohio). Ssh to one of the instances.

Execution:

```
MY_IP=$(curl https://ipinfo.io/ip)
time parallel-scp -p 100 -h ip.txt -x "-i \
 /home/ubuntu/ssle.pem -o StrictHostKeyChecking=no" \
 -l ubuntu 1M.txt "$MY_IP"_com.txt
```

Results: The final runtime log should be as follows. The final communication time is doubled since we have two rounds of broadcast. The evaluators could repeat the execution step multiple times to take the average.

```
real        0m2.192s
user        0m13.059s
sys         0m0.520s
```

(E3): *[WAN Communication] [<1 human-minute*

Preparation: Follow the same steps as in Appendix A.3.1 when preparing for the communication tests, with 32 instances under each of the four regions.

Execution:

```
MY_IP=$(curl https://ipinfo.io/ip)
time parallel-scp -p 100 -h ip.txt -x "-i \
 /home/ubuntu/ssle.pem -o StrictHostKeyChecking=no" \
 -l ubuntu 1M.txt "$MY_IP"_com.txt
```

Results: The final runtime log should be as follows. Same as before, we have two rounds in total and could take average of multiple trials for estimation.

```
real        0m8.772s
user        0m20.370s
sys         0m1.806s
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.

References

[1] D. Boneh, A. Partap, and L. Rotem. Post-Quantum Single Secret Leader Election (SSLE) From Publicly Rerandomizable Commitments. Cryptology ePrint Archive, Paper 2023/1241, 2023.

[2] Y. Wang and F. Zhang. Qelect: Lattice-based single secret leader election made practical. Cryptology ePrint Archive, Paper 2025/122, 2025.