Project Specification, Implementation Specification & Security Analysis

Anonymous

Purdue University

Problem Statement

We have 3 parties P1, P2, and P3 where

- P1, P2 , and P3 have a sharing of the decryption (private) key sk for ElGamal encryption. (The corresponding encryption (public) key is publicly known)
- Each Pi also holds an El Gamal encryption of a message mi Zp(none of the pi 's knows mi).
- A client C wants to learn (m1 m2) + m3 mod p, and is willing to pay 10000 ALGO for that.

Goal is to implement a protocol (combining MPC and an Algorand smart contract) which will allow Client C to learn what he wants but only if he pays.
Assumption: At most one of P1, P2 , and P3 is passively corrupted, and in addition the adversary might make one of the parties publicly fail (i.e., crash where the crash is observed by everyone). But the adversary may not attack (messages on) the network!

Language Used: Python. MPC is coded with low-level functions (in terms of only multiplication and addition gadget.). Smart contract part is incomplete. Hence, attack is requested on spec or Python implementation.

1 Protocol Specification

Specified concept is described in Fig.1 (screenshot from class slide). Here Ideal MPC is assumed. We have 3 parties and 1 client here. Party P1, P2, P3 has their ciphertext of messages m1, m2 and m3 which are namely c1, c2, c3. They also have secret key shares namely x1,x2 and x3. As, they have publicly key. Hence g is available to all of them and they can calculate g^{xi}

1.1 Communication

Communication protocol is specified in Fig.2. Inputs and outputs are specified below:
Input to trusted party: Each party Pi provides ci(El-gamal ciphertext of mi) and g^{xi} to circuit or trusted party.
Calculation by trusted party: Trusted party calculates Shamir sharing of Final function $m1m2 + m3 \mod p$ and returns 1 sharing to each party as well as evaluation point αi.
Reconstruction of $m1m2 + m3 \mod p$: Reconstruction will be done by client upon receiving each share. Reconstruction will be done by Lagrange Interpolation algorithm. Parties will not provide their share unless they receive 3333.33 ALGO.

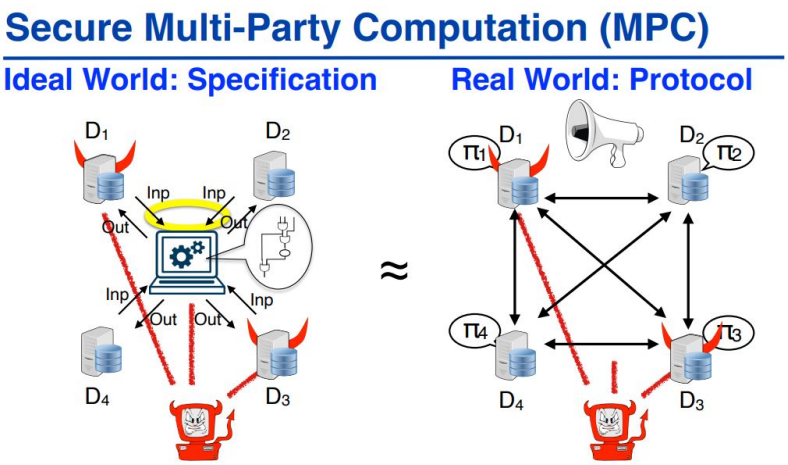

Figure 1: Ideal MPC specification used for this project

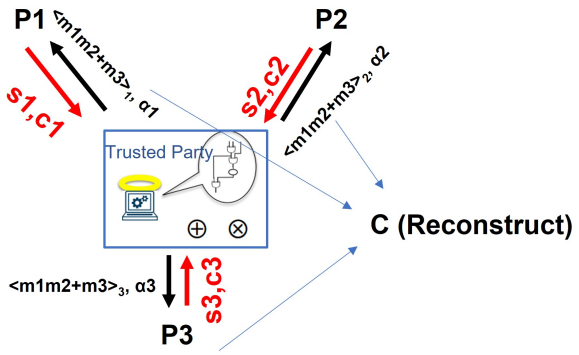

Figure 2: Full specification and protocol. Trusted party takes inputs of share of the secret key and ciphertexts and return Shamir share and its evaluation point. Client C with capability of reconstructing with Lagrange interpolation can reconstruct the secret.

1.2 Circuit of Trusted Party:

Circuit of trusted party is presented in Fig.3(a). Each party provides their secret which goes to a multiplication gadget and create the secret. With the help of the secret, messages m1, m2 and m3 is decrypted within trusted party. After this multiplication gadget is used to calculate $m1m2$ before adding it to $m3$ with an addition gadget. Note that, for El-gamal decryption is also a multiplication gadget as shown below.

Decrypted message, $m = s^{-1}c = s^{p-2}c$ (shown in Fig.3(b)), using Fermet's Little theorem, where c is ciphertext, s is secret and p is prime modulo. Note that all the operations are mod p. Hence, once decryption needs $p-2$ multiplication gadget.

Final step uses to share results of $m1m2 + m3 \mod p$ additively between all the parties (Fig. 4). This makes sure by passively corrupting one party, output will not be revealed.as we are taking degree-2 polynomial. Polynomial used for this particular case is $f(x) = s + ax + bx^2$ where a=3 and b=5. si is provided to each each party along with evaluation point.

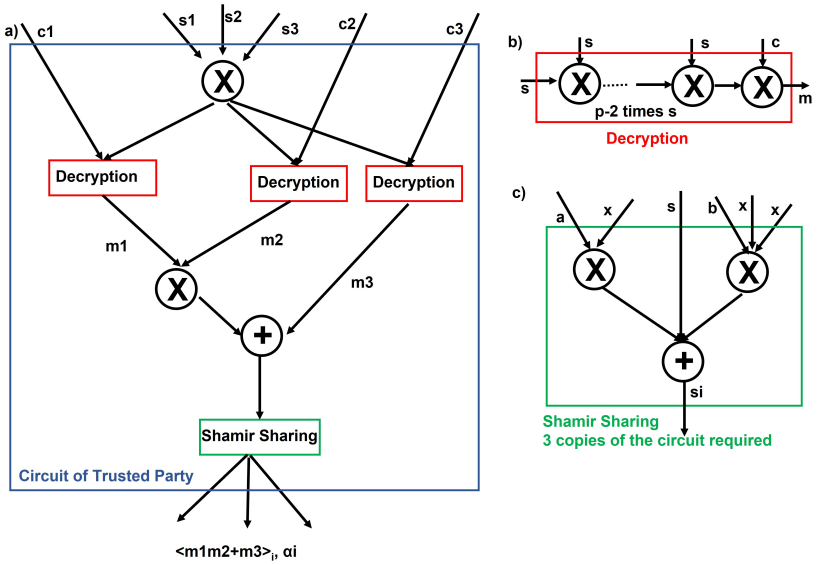

Figure 3: a) High level circuit diagram of trusted party. b) Decryption circuit c) Shamir sharing circuit. Note that in low level all the circuits are made of either multiplication and addition gadget. Code is written in circuit-level too.

1.3 Reconstruction by Client:

Upon paying equal amount, client receives the Shamir share of final results and evaluation point. By which, he can calculate Lagrange interpolation. Lagrange interpolation is generally given by

$$f(x) = \sum_{i=1} \ell_i(x) s_i \quad \text{where } \ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

However, we do not need full polynomial here. Rather just $f(0)$ should be enough. Hence, final reconstruction equation will be

$$s = f(0) = \sum_{i=1} \ell_i(0) s_i \quad \text{where } \ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{-\alpha_j}{\alpha_i - \alpha_j}$$

2 Code & Security Analysis

2.1 Code

Code will be found in the link: https://github.com/<....>/CS_555_blue_team_code.git

2.2 Security Analysis

2.2.1 Correctness

Final function reconstructed by the client is:
Output = Reconstruct (Sharing $((s1s2s3)^{-1}c1(s1s2s3)^{-1}c2 + (s1s2s3)^{-1}c3)$)
= Reconstruct (Sharing $(s^{-1}c1s^{-1}c2 + s^{-1}c3)$)
= Reconstruct (Sharing $(m1m2 + m3)$) = $m1m2 + m3$

if reconstruction and sharing is correct which is theoretically true. We can run standalone python code and verify it following the instructions. Note that all the operations are mod p

2.2.2 Privacy

Here privacy is secured theoretically. Secure channel is assumed in the question. Also, we use degree-2 shamir Polynomial. Hence, we had 3 sharing. Just by corrupting one party, adversary / dishonest client learns nothing. Here, as per question passive corruption is assumed. This circuit will of course fail against active corruption due to property of Shamir sharing.
Client (reconstruction circuit) needs to receive each and every share and evaluation point to reconstruct which makes the network secure even when 1 party is corrupted. Hence, under the assumption each party gives his share only when he receives the ALGO, privacy will be there.
In case of public failure, other parties will be aware and as secret key can not be restored, output calculation will not be possible and known to everyone including client.

2.2.3 Disclaimer:

I have tried implementing the ALGO transfer logic using Algorand Sandbox. However, due to lack of familiarity of syntax, I am unable to finish in time. Checked in pyteal code include a vulnerable implementation which just returns 1 share even if payment is not made. This is a known issue. Hence, discussion on possible attack on theory, spec and standalone Python implemetation will be appreciated.