Noname manuscript No.
(will be inserted by the editor)

Ideas

Donghang Lu

the date of receipt and acceptance should be inserted later

1 Introduction

Our goal is to design a transaction mixing protocol, which takes many transactions as input with the form of secret sharing and output them as a whole anonymity set. And validate nodes are not able to find out the source of any transaction, therefore providing privacy and unlinkability for these transactions.

And my protocol is a (n, k, t) protocol, there are n validate nodes who receive secret shares from multiple clients, and up to t of them may be corrupted by an advrsary, and any group of $t + 1$ or more validate nodes will reveal single secret, and our protocol will output k secrets each time so that these k secrets form an anonymity set and no one can trace their source.

2 Assumptions and System Model

In this section, we discuss the assumptions and the system model for our protocols, giving special attention to their practicality over the Internet.

2.1 Communication Model

Our protocol is based on asynchronous setting. We do not depend on any time bounds and limits.

Formally, we consider an asynchronous network of $n \geq 3t+1$ nodes $P_1, \ldots, P_n$ of which the adversary may corrupt up to t nodes during its existence .

Address(es) of author(s) should be given

2.2 Complexity and Cryptographic Assumptions

Our adversary is computationally bounded with a security parameter κ. A function $\epsilon(\cdot)$ is called *negligible* if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < 1/\kappa^c$ for all $\kappa > \kappa_0$.

And since we will use $PolyCommit_{ped}$ in eAVSS-SC, some cryptographic assumptions are needed. They are DLog assumption, t-polyDH Assumption and t-SDH Assumption.

2.3 Sub-protocols

Two sub-protocols are needed in our design.

First one is eAVSS-SC protocol and we use it to achieve secret sharing functions. The second one is VBA protocol which is used to make agreement on all honest valiadate nodes.

3 Protocol Design

From a high level point of view, our protocol works in three stages.

First, there are many clients sending their secret share s_j of s_0 by sending message like $(ID_d, SEND, \zeta, C, H_c(x), \mathbf{W_j}, \phi_j(x), \hat{\phi_j}(x))$ to every validate node P_j. And for each VSS request, P_j will initialize an eAVSS-SC protocol $ID|avss.i$ to deal with it (i in $ID|avss.i$ is the id of client).

Second, the validate node P_j waits for k sharings $ID|avss.i$ reaching $READY$ phase, which means their ζ is consistent and at least one node P_j has already received $(echo, \zeta)$ from at least $(n - t)$ nodes for each of these k eAVSS-SC instance.

Next, P_j proposes the set of k sharings which reach $READY$ phase for validated Byzantine agreement. The proposal is a set $\mathcal{L} = \{(i, M_i)\}$, indicating every ready sharing and containing the list M_i of signatures on *echo* messages. The predicate of VBA needs to verify that a proposal contains k valid lists of signatures from instances of protocol eAVSS-SC.

Finally, after the nodes decide in the VBA protocol for a set $\mathcal{L}$ which includes k eAVSS-SC instances, every nodes waits for these sharings to complete. For a node P_j , if it has received no information about an instance $ID|avss.m$ in final set $\mathcal{L}$ before, in $READY$ phase, it will receive at least (t+1) messages containing $(ready, \zeta, share, \phi_i(j), \hat{\phi_j}(x), w_j^i, C_i, \hat{h_c}(i), w_i^C)$ and passing $VerifyEval()$, P_j can use this info to run interpolation and get its share $\phi_0(j)$ and $\hat{\phi_0}(j)$.

So at last, each honest validate node P_j will have exactly k secret shares, the final share of P_j will be the summation of all these k shares.

The Reconstruction phase of our protocol is exactly the same as eAVSS-SC . And the result will be the summation of these k secrets $s_1 + s_2 + s_3 + ... + s_k$.

And we can simply get $s_1^2 + s_2^2 + s_3^2 + ... + s_k^2$ by doubling message size so that each message will actually contain two VSS instances, one for s_i and one for s_i^2. With the same method we can get $s_1^3 + s_2^3 + ... + s_k^3$ until $s_1^k + s_2^k + ... + s_k^k$. With these values we can reveal all the secrets.

Proof of knowledge is needed here to prove the k VSS instances in each message are corresponding to s_i,s_i^2, ... and s_i^k.

4 Complexity Analyze

Since the message complexity and communication complexity of eAVSS-SC is $O(n^2)$ and $O(n^2)$. and there are k eAVSS-SC instances running in our protocol. So the message complexity and communication complexity due to VSS is $O(kn^2)$ and $O(kn^2)$. And the message complexity and communication complexity of VBA is $O(n^2)$ and $O(n^3)$. So I am confused a little bit about why do we use VBA since VBA has communication complexity of $O(n^3)$, which just makes all the efforts of reducing Communication Complexity by using eAVSS-SC wasted.