J. Shanghai Jiaotong Univ. (Sci.), 2009, 14(1): 112-116
DOI: 10.1007/s12204-009-0112-5

Batch Secret Sharing for Secure Multi-party Computation in Asynchronous Network

HUANG Zheng[1*] (黄　征), *GONG Zheng*[2] (龚　征), *LI Qiang*[1] (李　强)
(1. School of Information Security and Engineering, Shanghai Jiaotong University, Shanghai 200030, China;
2. Distributed and Embedded Security Group, Faculty of EEMCS, University of Twente,
7500 AE Enschede, the Netherlands)

Abstract: This paper proposes an efficient batch secret sharing protocol among n players resilient to $t < n/4$ players in asynchronous network. The construction of our protocol is along the line of Hirt's protocol which works in synchronous model. Compared with the method of using secret share protocol m times to share m secrets, our protocol is quite efficient. The protocol can be used to improve the efficiency of secure multi-party computation (MPC) greatly in asynchronous network.

Key words: secret share, secure multi-party computation, asynchronous network
CLC number: TP 309　　**Document code:** A

Introduction

Secure multi-party computation (MPC) protocol allows a set of n players to compute an arbitrary agreed function of their private inputs in a secure way, even if an adversary may corrupt up to some arbitrary players. Secret sharing protocol is the basement for many MPC protocols. The efficiency of MPC protocols depends mainly on the efficiency of secret sharing protocol.

Players in MPC protocol are connected by a network. The network could be synchronous or asynchronous. In a synchronous network, there is a global clock. Message sent in the synchronous network will be guaranteed to be received in the next clock tick. While in an asynchronous network, there is no global clock. Message sent in the asynchronous network could be arbitrarily delayed. Compared with synchronous network, the asynchronous network is more like the Internet and the ad hoc networks, where message sent could also be arbitrarily delayed. Therefore, studying the protocols in the asynchronous network is more practical.

1 Related Works

Secret sharing protocol in synchronous network has been widely studied. However, there is little study in secret sharing and MPC in the asynchronous network. Canetti[1] studied MPC in asynchronous network. Ben-Or *et al*[2] showed that perfect asynchronous secure multiparty computation is possible in the information security setting if and only if $t < n/4$. Although theoretically impressive, these results lack in the area of practical feasibility, especially in the protocol shown in Ref. [2]. The complicated exchanges of message and zero-knowledge proofs might render them impractical[3]. Srinathan *et al*[4] provided an more efficient secure multiparty computation protocol in asynchronous model resilient to $t < n/4$ corrupt players. In order to share the secrets, it needs communications of $O(mn^3 \lg |\mathcal{F}| + mn^3 \lg n)$ bits and broadcasts $O(mn^2)$ bits.

In this paper, we propose a batch secret sharing protocol in asynchronous model which is resilient to $t < n/4$ corrupt players. The adversary in our protocol is active adversary with unbound computation power. This paper shows that share secret in batch mode will be more efficient than that of using secret sharing protocol several times. The construction of our protocol is along the line of Hirt's protocol[5] which works in synchronous model. Our protocol needs to communicate $O(n^4 \lg |\mathcal{F}| + mn^2 \lg |\mathcal{F}|)$ bits and broadcast $O(n^2 \lg |\mathcal{F}|)$ bits, where m is the number of the secrets been shared.

2 Protocol Model

The model used in our protocol could be described as follows. The participants of the protocol are the set

Received date: 2008-10-30
Foundation item: the National Natural Science Foundation of China (No. 60803146)
∗**E-mail:** huang-zheng@sjtu.edu.cn

$\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ of n players, and each player is associated with a non zero value $\alpha_i \in \mathcal{F}$. The players are connected by bilateral asynchronous secure channels. Broadcast channels are not assumed to be available. All the computations in the sequel are done in $\mathcal{F}$. In our model, we consider asynchronous network where every two parties are connected via a reliable and private communication channel. Messages sent on a channel can be arbitrarily delayed, however, each message sent is eventually received. Furthermore, the order in which messages are received in a channel may be different from the order in which they were sent.

3 Primitives

3.1 Agreement on a Common Subset (ACS)

In an asynchronous computation, the players in $\mathcal{P}$ often need to decide on a common subset of players that satisfy some property. The common set is of size at least $n - t$, where $n = |\mathcal{P}|$. We need ACS protocol to achieve this purpose[1].

Let Q be a predication that assigns a binary value to each player P_i, denoted $Q(i)$, based on whether P_i has satisfied the specified property. Note that $Q(i)$ is dynamic which means that not all the players need to be assigned at the same time. When a player P_i is assigned his value, denoted by $Q(i)$, it is guaranteed that all the players will eventually know this value. Let $\mathcal{P}$ be the set of players and t be the maximum number of corrupt players in $\mathcal{P}$.

Theorem 1 (protocol ACS$[Q, \mathcal{P}, t]$) Let the players agree on a common subset of players, denoted by $\mathcal{C}$ such that $|\mathcal{C}| \geqslant (|\mathcal{P}| - t)$. Moreover, $Q(j) = 1$ for every $P_j \in \mathcal{P}$. The ACS protocol has to do $O(|\mathcal{P}|)$ Byzantine agreement.

3.2 Asynchronous Verifiable Secret Sharing (AVSS)

Like verifiable secret sharing (VSS) in synchronous network, AVSS scheme consists of two sub-protocols: a sharing sub-protocol AV-Share, in which a player acts as dealer to share a secret among the other players, and a reconstruction sub-protocol, in which the players reconstruct the secret from their shares. For the definition and the implementing of AVSS scheme, we also adopt directly the scheme presented in Ref. [2] which is resilient to $n/4$ corrupt players.

Let $\mathcal{P}$ denote the set of all players, P denote the dealer, t be the maximum number of corrupt players in $\mathcal{P}$ and s be the secret to be shared.

Theorem 2 (AV-Share$[\mathcal{P}, t, P, s]$ protocol) Let the dealer P share the secret s with the following holds.

Termination If P is honest, then every honest player will eventually complete the AV-share protocol. If some honest player has completed AV-Share protocol, then all the honest players will eventually complete the AV-Share protocol.

Correctness Once a honest party has completed AV-Share protocol then there exits a unique value, r, such that: ① all the honest players output r; ② if P is honest,then $r = s$.

Secrecy If P is honest and no honest player has begun reconstruction protocol, then the bad parties have no information about the shared secret s.

4 Protocol Construction

The goal of the protocol is to generate m t-shared random triples (a, b, c) with $c = ab$ in such a way that the adversary obtains no information about a, b, c except that c is the product of a and b.

The triples are generated in a block of $l = \lceil m/n \rceil$ triples. The block is generated in a non-robust manner and there is a verification step at the end of the generation. Fault may be found in the verification step. In case of fault found, a set $\mathcal{D}$ of two or three players is identified in all the honest players. At least one player in $\mathcal{D}$ is the corrupt player. The players in $\mathcal{D}$ will be eliminated from further computation (player elimination). The triples of the block with fault are discarded. Player elimination ensures that at most t blocks fail, and hence in order to generate m random triples, at most $(n + t)$ blocks must be processed.

There are two steps in the verification. The first verification step is the verification of the degrees of all sharing. The second verification step is the verification that all players share the correct product shares.

Both verification steps use n random triples for blinding purpose. These triples are used to protect the privacy of the l triples and their privacy is annihilated in the verification step. Therefore, in each block, $l + n$ triples are generated. l triples will be used as the output and n triples will be used in the verification steps.

During the generation of blocks, some players may be eliminated. Let $\mathcal{P}'$ denote the actual set of players, n' denote the actual number of players with $n' = |\mathcal{P}'|$, and t' denote the maximum number of corrupt players in $\mathcal{P}'$. Without loss of generality, we assume that $\mathcal{P}' = \{P_1, P_2, \cdots, P_{n'}\}$. In the beginning, $\mathcal{P}' = \mathcal{P}$, $n' = n$, and $t' = t$. The inequality $2t' < n' - t' - t$ holds in the beginning. In player elimination, n' will be decreased by 3 or 2, and t' by 1. Clearly, the inequality $2t' < n' - t' - t$ still holds. This inequality is needed for robust computation.

The protocol is processed as follows:

(1) Initial the preparation phase by setting $\mathcal{P}' = \mathcal{P}$, $n' = n$, and $t' = t$.

(2) Repeat until n blocks are succeeded.

Generate $l + n$ triples (in parallel) in a non-robust manner.

Do the first Verification.

If no fault is detected in the first verification step, do the second Verification.

If both verification steps are successful, then the generation of the block is successful. If either verification step fails, then all triples of the block are discarded.

4.1 Generating One t-shared Triple (a, b, c)

The generation of t-shared triple is executed in a batch mode, that means $l + n'$ triples are generated in the same run of the protocol.

The generation of one t-shared triple (a, b, c) in non-robust manner is proceeded as follows:

Step 1 The players jointly generate t'-sharing of random values a and b.

(1) Select two random t'-degree polynomials $\tilde{f}_i(x)$ and $\tilde{g}_i(x)$ and hand the shares $\tilde{a}_{ij} = \tilde{f}_i(\alpha_j)$ and $\tilde{b}_{ij} = \tilde{g}_i(\alpha_j)$ to player P_j for $j = 1, 2, \cdots, n'$.

(2) Broadcast one bit to indicate that P_i has finished the sharing.

(3) Execute protocol $\mathcal{C} = \mathrm{ACS}[Q, \mathcal{P}', t']$ with the Boolean predicate: $Q(j) = 1$ if P_j has finished the broadcast and the share that P_j sends has been received. Let $\mathcal{C}1$ be the common set outputted by ACS protocol.

(4) The polynomial of sharing a is $\tilde{f}(x) = \sum\limits_{j\in\mathcal{C}1} \tilde{f}_j(x)$ (thus $a = \tilde{f}(0)$), and the polynomial for sharing b is $\tilde{g}(x) = \sum\limits_{j\in\mathcal{C}1} \tilde{g}_j(x)$. Calculate his share of a and b as $\tilde{a}_i = \sum\limits_{j\in\mathcal{C}1} \tilde{a}_{ji}$, and $\tilde{b}_i = \sum\limits_{j\in\mathcal{C}1} \tilde{b}_{ji}$.

Step 2 The players jointly compute a t'-sharing of $c = ab$.

(1) Calculate $\tilde{e}_i = \tilde{a}_i\tilde{b}_i$ and select random a t'-degree polynomial $\tilde{h}_i(x)$ with $\tilde{h}_i(0) = \tilde{e}_i$. Send the shares $\tilde{e}_{ij} = \tilde{h}_i(\alpha_j)$ to player P_j for $P_j \in \mathcal{P}'$.

(2) Broadcast one bit to indicate that P_i has finished the sharing.

(3) Execute protocol $\mathcal{C} = \mathrm{ACS}[Q, \mathcal{P}', t']$ with the Boolean predicate: $Q(j) = 1$ if P_j has finished the broadcast and the share that P_j sends has been received. Let $\mathcal{C}2$ be the common set outputted by ACS protocol.

(4) Compute his share $\tilde{c}_i$ of c as $\tilde{c}_i = \sum\limits_{j\in\mathcal{C}2} \varpi_i \tilde{e}_{ji}$, where $\varpi_i = \prod\limits_{j\in\mathcal{C}2, j\neq i} \dfrac{\alpha_j}{\alpha_j - \alpha_i}$ is the Lagrange interpolation factor.

Step 3 The players jointly increase the degree of the sharing of a, b and c to t (this step is performed only if $t' < t$).

(1) Select three random $(t-1)$-degree polynomials $\bar{f}_i(x)$, $\bar{g}_i(x)$ and $\bar{h}_i(x)$. Hand the shares $\bar{a}_{ij} = \bar{f}_i(\alpha_j)$, $\bar{b}_{ij} = \bar{g}_i(\alpha_j)$, and $\bar{c}_{ij} = \bar{h}_i(\alpha_j)$ to player P_j for $j = 1, 2, \cdots, n'$.

(2) Broadcast one bit to indicate that P_i has finished the sharing.

(3) Execute protocol $\mathcal{C} = \mathrm{ACS}[Q, \mathcal{P}', t']$ with the Boolean predicate: $Q(j) = 1$ if P_j has finished the broadcast and the share that P_j sends has been received. Let $\mathcal{C}3$ be the common set outputted by ACS protocol.

(4) Compute his share as $a_i = \tilde{a}_i + \alpha_i \sum\limits_{j\in\mathcal{C}3} \bar{a}_{ji}$, $b_i = \tilde{b}_i + \alpha_i \sum\limits_{j\in\mathcal{C}3} \bar{b}_{ji}$, and $c_j = \tilde{c}_i + \alpha_i \sum\limits_{j\in\mathcal{C}3} \bar{c}_{ji}$.

4.2 Verification of the Degrees of All Sharing in a Batch Manner

The goal of the verification protocol is to verify the degree of the sharing of the first $l + n$ triples in a single step, using another n triples. The basic idea of this protocol is to verify the degree of a random liner combination of the polynomials. More precisely, every player distributes a random challenge vector of length l with elements in $\mathcal{F}$. Then, every player constructs the random polynomial towards every challenge player, who checks if the constructed polynomial is of appropriate degree. In order to preserve the privacy of the involved polynomials, the other n' triples in one block are used. A set $\mathcal{D} \subseteq \mathcal{P}'$ of three players is identified and eliminated from $\mathcal{P}'$ when some fault is found.

Code for party $P_i \in \mathcal{P}'$:

(1) Select a random vector $[r_1, r_2, \cdots, r_l]$ with elements in $\mathcal{F}$ and send it to each player $P_i \in \mathcal{P}'$.

(2) Upon receiving a random vector from P_v, compute the following corresponding linear combinations

$$\tilde{a}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \tilde{a}_{ij}^{(k)} + \tilde{a}_{ij}^{(l+v)},$$

$$\tilde{b}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \tilde{b}_{ij}^{(k)} + \tilde{b}_{ij}^{(l+v)},$$

$$\tilde{c}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \tilde{c}_{ij}^{(k)} + \tilde{c}_{ij}^{(l+v)},$$

$$\tilde{e}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \tilde{e}_{ij}^{(k)} + \tilde{e}_{ij}^{(l+v)},$$

$$\bar{a}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \bar{a}_{ij}^{(k)} + \bar{a}_{ij}^{(l+v)},$$

$$\bar{b}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \bar{b}_{ij}^{(k)} + \bar{b}_{ij}^{(l+v)},$$

$$\bar{c}_{ij}^{\Sigma} = \sum_{k=1}^{l} r_k \bar{c}_{ij}^{(k)} + \bar{c}_{ij}^{(l+v)},$$

and hands these combinations to P_v.

(3) Wait until received combinations from $n'-t'$ players. Without lost of generality, assume that the received combinations are of the set $\mathcal{R} = \{P_1, P_2, \cdots, P_{\hat{n}}\}$. P_i verifies whether for each $j = 1, 2, \cdots, n'$, the received combinations $\tilde{a}_{j1}^{\Sigma}, \tilde{a}_{j2}^{\Sigma}, \cdots, \tilde{a}_{j\hat{n}}^{\Sigma}$ lie on a polynomial of degree at most t'. The same verification is performed for the shares $\tilde{b}_{j1}^{\Sigma}, \tilde{b}_{j2}^{\Sigma}, \cdots, \tilde{b}_{j\hat{n}}^{\Sigma}$ and $\tilde{c}_{j1}^{\Sigma}, \tilde{c}_{j2}^{\Sigma}, \cdots, \tilde{c}_{j\hat{n}}^{\Sigma}$. Next, P_i verifies whether the received combinations $\bar{a}_{j1}^{\Sigma}, \bar{a}_{j2}^{\Sigma}, \cdots, \bar{a}_{j\hat{n}}^{\Sigma}$ lie on a polynomial of degree at most $t-1$. The same verification is performed for the combinations $\bar{b}_{j1}^{\Sigma}$, $\bar{b}_{j2}^{\Sigma}, \cdots$, $\bar{b}_{j\hat{n}}^{\Sigma}$ and $\bar{c}_{j1}^{\Sigma}, \bar{c}_{j2}^{\Sigma}, \cdots, \bar{c}_{j\hat{n}}^{\Sigma}$. Furthermore, P_i verifies whether the received combinations $\tilde{e}_{j1}^{\Sigma}, \tilde{e}_{j2}^{\Sigma}, \cdots, \tilde{e}_{j\hat{n}}^{\Sigma}$ lie on a polynomial of degree at most t'.

(4) If all the verifications passed, take no action. If not, try to localize the fault and identify the set $\mathcal{D}$. To do this, P_i selects one of the polynomials of too high degree and broadcast the location of the fault. Without loss of generality, we assume that the fault was observed in the combination $\tilde{a}_{j1}^{\Sigma}, \tilde{a}_{j2}^{\Sigma}, \cdots, \tilde{a}_{j\hat{n}}^{\Sigma}$. P_i calculates the maximum set $\mathcal{H}$ such that for each $v \in \mathcal{H}$, $\tilde{a}_{jv}^{\Sigma}$ lies on a polynomial of degree at most t'. Let $\vec{f}(x)$ denote the polynomial that is determined by the combinations hold by players in set $\mathcal{H}$. If $|\mathcal{H}| \geqslant n' - 2t'$, P_i selects a player P_k as the player whose combination does not lie on $\vec{f}(x)$ and set $\mathcal{D}_i := \{P_i, P_j, P_k\}$. If $|\mathcal{H}| \leqslant n' - 2t'$, P_i selects a player P_k in P' randomly and sets $\mathcal{D}_i := \{P_j, P_i, P_k\}$. P_i broadcasts the complaint $\mathcal{D}_i$.

(5) Execute protocol ACS$[Q, \mathcal{P}', t']$ with the Boolean predicate: $Q(j) = 1$ if P_j has finished the broadcast. Players that have complete the broadcast, set $\mathcal{C} = \text{ACS}(Q, \mathcal{P}', t')$. P_i selects P_j in the $\mathcal{C}$ who has the smallest index number and sets the final $\mathcal{D} := \mathcal{D}_j$.

(6) Discard the block and set $\mathcal{P}' := \mathcal{P}' \backslash \mathcal{D}$, $n' = n' - 3$, $t' = t' - 1$.

4.3 Verification of All Players Sharing the Correct Product Shares

Now, we have to verify that in each triple $k = 1, 2, \cdots, l$, every player P_i shares the correct product share $\tilde{e}_i^{(k)} = \tilde{a}_i^{(k)} \tilde{b}_i^{(k)}$. Since in the first verification, the degrees of all sharing are of degree t'. It is sufficient to verify that the shares $\tilde{e}_1^{(k)}, \tilde{e}_2^{(k)}, \cdots, \tilde{e}_{n'}^{(k)}$ lie on a polynomial of degree at most $2t'$. Note that at least $n'-t'-t' > 2t'$ shares of the honest players uniquely define this polynomial. The key idea of this verification is that every player P_i interpolates random combinations of $\tilde{e}_i^{(k)}$ and checks if the combinations lie on a polynomial of degree at most $2t'$. If a fault is detected, a set $\mathcal{D}$ of two players will be identified and eliminated from the player set $\mathcal{P}'$.

Code for party $P_i \in \mathcal{P}'$:

(1) Interpolate $\tilde{e}_1^{\Sigma}, \tilde{e}_2^{\Sigma}, \cdots, \tilde{e}_{n'}^{\Sigma}$ by using the values computed in the previous verification protocol $\tilde{e}_{j1}^{\Sigma}$, $\tilde{e}_{j2}^{\Sigma}, \cdots, \tilde{e}_{j\hat{n}}^{\Sigma}$, for $j = 1, 2, \cdots, n'$.

(2) Check if $\tilde{e}_1^{\Sigma}, \tilde{e}_2^{\Sigma}, \cdots, \tilde{e}_{n'}^{\Sigma}$ lie on a polynomial of degree at most $2t'$. If all the verifications passed, take no action. Else try to localize the fault and identify the set $\mathcal{D}$. To do this, P_i calculates the maximum set $\mathcal{H}$ such that for each $i \in \mathcal{H}$, $\tilde{e}_j^{\Sigma}$ lies on a polynomial of degree at most $2t'$. Let $\vec{f}(x)$ denote the polynomial that is determined by the combinations hold by players in set $\mathcal{H}$. P_i selects a player P_k as the player whose combination does not lie on $\vec{f}(x)$ and sets $\mathcal{D}_i := \{P_i, P_k\}$. P_i broadcasts the complaint $\mathcal{D}_i$.

(3) Execute protocol ACS$[Q, \mathcal{P}', t']$ with the Boolean predicate: $Q(j) = 1$ if P_j has finished the broadcast. Players that have complete the broadcast set $\mathcal{C} = \text{ACS}[Q, \mathcal{P}', t']$. P_i selects P_j in the $\mathcal{C}$ who has the smallest index number and sets the final $\mathcal{D} := \mathcal{D}_j$.

(4) Discard the block and set $\mathcal{P}' := \mathcal{P}' \backslash \mathcal{D}$, $n' = n' - 2$, $t' = t' - 1$.

4.4 Security Analysis

The correctness also follows from simple algebra that if all players are honest, then the verification will always pass. However, if one player in the Core-Set $\mathcal{C}$ does not share the correct product share, the interpolation of combination shares will not lie on a polynomial of degree at most $2t'$. If a fault is detected, P_i may be corrupt player who has declared the fault intentionally. Otherwise, P_i is honest, then P_i could interpolate at least $n' - t'$ values of $\tilde{e}_j^{\Sigma}$, among which $n' - 2t'$ are from honest players. The $n' - 2t'$ interpolated values could determine a polynomial of degree at most $2t'$ uniquely. Thus, the player P_k whose share does not lie on the polynomial determined by the $n' - 2t'$ honest players is corrupt. In all the above cases, at least one player in the set $\mathcal{D} = \{P_i, P_k\}$ is corrupt.

5 Complexity Analysis and Conclusion

Here we give a brief complexity analysis of our protocol. We use MC to denote the complexity of the message that sent via secure channel and use BAC denote the complexity of the message that sent by using Broadcast protocol.

Generation of one block The message complexity of generation m triples is that MC $= O((l +$

$n)n^3 \lg|\mathcal{F}|)$ and BAC $= O(n^2)$. In the worst case in which t players are found to be corrupt, the message complexity of generation m triples is also the same: MC $= O((l+n)n^3 \lg|\mathcal{F}|) = O(n^4 \lg|\mathcal{F}| + mn^2 \lg|\mathcal{F}|)$ and BAC $= O(n^2)$.

First verification step This verification step will be performed $O(n)$ times, so the whole message complexity of this verification step in the preparation phase is that: MC $= O(n^3 \lg|\mathcal{F}|)$ and BAC $= O(n^2 \lg|\mathcal{F}|)$.

Second verification step This verification step will be performed $O(n)$ times, so the whole message complexity of this verification step in the preparation phase is that: BAC $= O(n^2 \lg|\mathcal{F}|)$.

The total message of the protocol is that MC $= O(n^4 \lg|\mathcal{F}| + mn^2 \lg|\mathcal{F}|)$ and BAC $= O(n^2 \lg|\mathcal{F}|)$.

The message complexity of the general purpose MPC protocol is the bottle neck to its implementation. We propose an efficient batch secret sharing protocol in asynchronous network that could improve the efficiency of MPC greatly.

References

[1] Canetti R. Studies in secure multiparty computation and applications [D]. Rehovot, Israel: Weizmann Institute of Science, 1995.

[2] Ben-or M, Kelmer B, Rabin T. Asynchromous secure computation with optimal resilience [C]//*Proceedings of 13th ACM PODC*. New York: ACM, 1994: 183-192.

[3] Ashwin K M V N, Srinathan K, Pandu R C. Asynchronous perfectly secure computation tolerating generalized adversaries [C]//*Proceedings of ACISP 2002*. Heidelberg: Springer-Verlag, 2002: 497-511.

[4] Srinathan K, Rangan C. Efficient asynchronous secure multiparty distributed computation [C]//*Proceedings of Progress in Cryptology INDOCRYPT 2000*. Heidelberg: Springer-Verlag, 2000: 117-130.

[5] Hirt M, Mauren U. Robustness for free in unconditional multiparty computation [C]//*Advances in Cryptology-CRYPTO '01*. Heidelberg: Springer-Verlag, 2001: 101-118.