

## 1. Give a bcbBroadcast algorithm

With  $O(N)$  messages

Without signatures, but using MACs!

Assume  $N > 5f$ .

What is the complexity?

这个解决方案是对之前的签名回声广播算法的改进，其中使用消息认证码（MAC）代替数字签名。

### 使用MAC代替数字签名

- 在原算法中，使用  $\sigma := \text{sign}(\text{self}, [\text{ECHO}, m])$  生成数字签名。在改进方案中， $\sigma$  变为包含  $n$  个 MAC 的向量（称为认证器），其中  $\sigma[pk] := \text{authenticate}(\text{self}, pk, [\text{ECHO}, m])$ 。
- 原算法中使用  $\text{verifysig}(pj, [\text{ECHO}, m], \sigma)$  来验证签名。在改进方案中，使用  $\text{verifyauth}(\text{self}, pj, [\text{ECHO}, m], \sigma[\text{self}])$  来验证 MAC。

### 潜在问题与解决方案

- MAC认证器与签名的等效性：**MAC认证器的验证可能在某些正确进程  $q$  上失败，而在其他正确进程  $q'$  上成功。这可能发生在：
  - $pk$ （或发送者）是拜占庭式的。
  - MAC  $\sigma[q']$  由  $pk$  正确计算。
  - 但 MAC  $\sigma[q]$  被  $pk$ （或发送者）篡改。
- 当  $pk \neq$  发送者并且是拜占庭式的情况下：**这可能违反 bcbBroadcast 的有效性（Validity）。

### 引入 $5f+1$ 规则以恢复有效性

- 发送 FINAL 消息的条件调整：**在接收到  $n-f = 4f+1$  个有效的 ECHO 消息后发送 FINAL 消息，而不是之前的  $(n+f)/2$ 。
- 发送者验证  $n-f$  个 MAC：**并转发  $n-f$  个认证器。至少  $n-2f$  个认证器来自正确的进程。
- 保证：** $n-2f > (n+f)/2$ ，需要  $n > 5f$ 。

#### MAC-Vector Echo Broadcast

# 拜占庭一致性广播算法，使用消息认证码（MAC）

Implements: ByzantineConsistentBroadcast, instance bcb, with sender  $s$ .

Uses: AuthPerfectPointToPointLinks, instance  $a1$ .

# 初始化

upon event  $\langle \text{bcb}, \text{Init} \rangle$  do

```
    sentecho := FALSE; # 是否已发送echo消息
    sentfinal := FALSE; # 是否已发送final消息
    delivered := FALSE; # 是否已决定接收消息
    echos :=  $[\perp]N$ ; # 记录收到的echo消息
     $\Sigma$  :=  $[\perp]N$ ; # 记录认证码
```

# 广播事件处理

```

upon event < bcb, Broadcast | m > do # 广播者s广播消息m
    forall q ∈ n do # 对所有进程
        trigger < a1, Send | q, [SEND, m] >; # 发送SEND消息

# 处理SEND消息
upon event < a1, Deliver | p, [SEND, m] > such that p = s and sentecho = FALSE do
    sentecho := TRUE;
    forall pk ∈ n do # 对所有进程
        σ[pk] := authenticate(self, pk, [ECHO, m]); # 生成对应的MAC
    trigger < a1, Send | s, [ECHO, m, σ] >; # 发送带有MAC的ECHO消息

# 处理ECHO消息
upon event < a1, Deliver | p, [ECHO, m, σ] > do
    if echos[p] = ⊥ ∧ verifyauth(self, p, [ECHO, m], σ[self]) then # 验证MAC
        echos[p] := m; # 记录echo消息
        Σ[p] := σ; # 记录MAC

# 发送FINAL消息
upon exists m ≠ ⊥ such that #{p ∈ n | echos[p] = m} > N-f and sentfinal = FALSE do
    sentfinal := TRUE;
    forall q ∈ n do
        trigger < a1, Send | q, [FINAL, m, Σ] >; # 发送FINAL消息

# 处理FINAL消息
upon event < a1, Deliver | p, [FINAL, m, Σ] > do
    if #{q ∈ n | Σ[q] ≠ ⊥ ∧ verifyauth(self, q, [ECHO, m], Σ[q][self])} > (N+f)/2
    and delivered = FALSE do
        delivered := TRUE;
        trigger < bcb, Deliver | s, m >; # 决定接收消息

```

### 1. 时间复杂度 (Time complexity) :

- 此算法涉及三个主要阶段：发送 SEND 消息、发送 ECHO 消息和发送 FINAL 消息。
- 每个阶段都需要一次消息传递延迟。
- 因此，总的时间复杂度为三次消息传递延迟，即 3。

### 2. 消息复杂度 (Message complexity) :

- 在每个阶段，算法中的每个进程都会向所有其他进程发送消息。
- 假设共有  $N$  个进程，每个进程都发送消息给其他  $N-1$  个进程，但考虑到实际操作中一些优化（例如，不需要向自身发送消息），消息的总数接近  $N$  个。
- 因此，总的消息复杂度是  $O(N)$ 。

### 2. Give a brbBroadcast algorithm

Using digital signatures

What is the complexity?

Totality or uniform totality?

Implements: ByzantineReliableBroadcast, instance brb, with sender s.  
 Uses: AuthPerfectPointToPointLinks, instance a1.

```

# 初始化算法
upon event < brb, Init > do
    sentecho := FALSE;           # 是否已发送回声标志
    sentready := FALSE;          # 是否已发送准备标志
    delivered := FALSE;          # 是否已传递标志
    echos := [ $\perp$ ] $N$ ;           # 存储收到的回声
     $\Sigma$  := [ $\perp$ ] $N$ ;             # 存储签名

# 发送者广播消息
upon event < brb, Broadcast | m > do
    forall q  $\in$   $\pi$  do
        trigger < a1, Send | q, [SEND, m] >; # 向所有进程发送消息

# 当接收到发送者的消息时
upon event < a1, Deliver | p, [SEND, m] such that p = s and sentecho = FALSE do
    sentecho := TRUE;                # 设置已发送回声标志为真
     $\sigma$  := sign(self, [ECHO, m]);    # 对回声消息进行签名
    forall q  $\in$   $\pi$  do
        trigger < a1, Send | q, [ECHO, m,  $\sigma$ ] >; # 广播带有签名的回声消息

# 当接收到回声消息时
upon event < a1, Deliver | p, [ECHO, m,  $\sigma$ ] > do
    if echos[p] =  $\perp$   $\wedge$  verifysig(p, [ECHO, m,  $\sigma$ ]) then # 验证签名
        echos[p] := m;                                     # 存储验证通过的回声消息
         $\Sigma$ [p] :=  $\sigma$ ;                                   # 存储签名

# 当超过半数的进程发送了回声消息时
upon exists m  $\neq$   $\perp$  such that  $\#\{p \in \pi \mid echos[p] = m\} > (N+f)/2$  and delivered = FALSE do
    delivered := TRUE;                # 设置已传递标志为真
    trigger < brb, Deliver | s, m >;   # 传递消息
    forall q  $\in$   $\pi$  do
        trigger < a1, Send | q, [READY, m,  $\Sigma$ ] >; # 向所有进程发送准备消息

# 当接收到准备消息时
upon event < a1, Deliver | p, [READY, m,  $\Sigma'$ ] > do
    forall q  $\in$   $\pi$  do
        if echos[q] =  $\perp$   $\wedge$   $\Sigma'[q] \neq \perp \wedge$  verifysig(p, [ECHO, m,  $\Sigma'[q]$ ]) then
            echos[q] := m;                # 存储验证通过的回声消息
             $\Sigma$ [q] :=  $\Sigma'[q]$ ;          # 存储签名

```

## 1. 时间复杂度:

- **最佳情况:** 算法在最佳情况下需要2次消息延迟。这包括:
  - 第一次延迟: 发送者 `src` 广播初始 `SEND` 消息到所有其他进程, 然后接收进程回应 `ECHO` 消息。
  - 第二次延迟: 接收进程接收到足够的 `ECHO` 消息后, 广播 `READY` 消息, 随后所有正确的进程可以交付消息。

## 2. 消息复杂度:

- 消息复杂度为  $O(N^2)$ 。这是因为:

- 每个进程都需要向所有其他  $N-1$  个进程发送消息，这发生在几个阶段中：
  - 初始 SEND 消息的广播。
  - ECHO 消息的响应。
  - 最后， READY 消息的广播。
- 因此，每个阶段的消息数是  $N(N-1)$ ，考虑到这些阶段，总的消息数在数量级上是  $N^2$ 。