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`σ := sign(self, [ECHO, m])`生成数字签名。在改进方案中，`σ`变为包含n个MAC的向量（称为认证器），其中`σ[pk] := authenticate(self, pk, [ECHO, m])`。
- 原算法中使用`verifysig(pj, [ECHO, m], σ)`来验证签名。在改进方案中，使用`verifyauth(self, pj, [ECHO, m], σ[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 σ[q']由pk正确计算。
  - 但MAC σ[q]被pk（或发送者）篡改。
- **当pk ≠ 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 al.

# 初始化
upon event < bcb, Init > 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 := [⊥]N;  # 记录收到的echo消息
    Σ := [⊥]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 ∈ Π do  # 对所有进程
        trigger < al, Send | q, [SEND, m] >;  # 发送SEND消息

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

# 处理ECHO消息
upon event < al, 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 ∈ Π | echos[p] = m} > N-f and sentfinal = FALSE do
    sentfinal := TRUE;
    forall q ∈ Π do
        trigger < al, Send | q, [FINAL, m, Σ] >;  # 发送FINAL消息

# 处理FINAL消息
upon event < al, Deliver | p, [FINAL, m, Σ] > do
    if #{q ∈ Π | 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)`。
3. 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 al.
```

```
# 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 := [⊥]N;                # 存储收到的回声
    Σ := [⊥]N;                    # 存储签名

# 发送者广播消息
upon event < brb, Broadcast | m > do
    forall q ∈ Π do
        trigger < al, 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 < al, 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
    σ := sign(self, [ECHO, m]);               # 对回声消息进行签名
    forall q ∈ Π do
        trigger < al, Send | q, [ECHO, m, σ] >;  # 广播带有签名的回声消息

# 当接收到回声消息时
upon event < al, Deliver | p, [ECHO, m, σ] > do
    if echos[p] = ⊥ ∧ verifysig(p, [ECHO, m, σ]) then  # 验证签名
        echos[p] := m;                          # 存储验证通过的回声消息
        Σ[p] := σ;                              # 存储签名

# 当超过半数的进程发送了回声消息时
upon exists m ≠ ⊥ such that #{p ∈ Π | 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 ∈ Π do
        trigger < al, Send | q, [READY, m, Σ] >;  # 向所有进程发送准备消息

# 当接收到准备消息时
upon event < al, Deliver | p, [READY, m, Σ'] > do
    forall q ∈ Π do
        if echos[q] = ⊥ ∧ Σ'[q] ≠ ⊥ ∧ verifysig(p, [ECHO, m, Σ'[q]]) then
            echos[q] := m;                      # 存储验证通过的回声消息
            Σ[q] := Σ'[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`。

Totality 或 Uniform Totality:

- 这个算法满足**Totality**属性，即如果任何一个正确的进程传递了一条消息，每个正确的进程最终都会传递这条消息。