Signed Messages

- Traitors ability to lie makes Byzantine General Problem so difficult.
- If we restrict this ability, then the problem becomes easier
- Use authentication, i.e. allow generals to send unforgeable signed messages.

Signed Messages

- Assumptions about Signed Messages

A1: every message that is sent is delivered correctly
A2: the receiver of a message knows who send it
A3: the absence of a message can be detected
A4: a loyal general’s signature cannot be forged, and any alteration of the contents of his signed messages can be detected. Anyone can verify the authenticity of a general’s signature.

Note: no assumptions are made about a traitor general, i.e. a traitor can forge the signature of another traitor.
Signed Messages

◆ Signed message algorithm assumes a choice function
  – if a set $V$ has one single element $v$, then $\text{choice}(V) = v$
  – $\text{choice}(\emptyset) = R$, where $\emptyset$ is the empty set
    » RETREAT is default
  – $\text{choice}(A,R) = R$
    » RETREAT is default
  – set $V$ is not a multiset (recall definition of a multiset)
  – thus set $V$ can have at most 2 elements, e.g. $V = \{A,R\}$.

Signed Messages

◆ Signing notation
  – let $v:i$ be the value $v$ signed by general $i$
  – let $v:i:j$ be the message $v:i$ counter-signed by general $j$
◆ each general $i$ maintains his own set $V_i$ containing all orders he received
◆ Note: do not confuse the set $V_i$ of orders the general received with the set of all messages he received. Many different messages may have the same order.
BGP: Signed Message Solution

SM(m) -- from Lam82

Initially $V_i = \Phi$

1) The commander signs and sends his value to every lieutenant
2) For each $i$
   A) If lieutenant $i$ receives a message of the form $v:0$ from the commander and he has not yet received any order, then
      i) he lets $V_i$ equal \{v\}
      ii) he sends the message $v:0:i$ to every other lieutenant
   B) If lieutenant $i$ receives a message of the form $v:0:j_1:...:j_k$ and $v$ is not in the set $V_i$, then
      i) he adds $v$ to $V_i$
      ii) if $k < m$, then he sends the message $v:0:j_1:...:j_k:i$ to every lieutenant other than $j_1,...,j_k$

Algorithm SM(m)

- The SM(m) algorithm for signed messages works for
  \[ N \geq m + 2 \]
  - i.e. want non faulty commander and at least one non faulty lieutenant
- How does one know when one does not receive any more messages?
  - by missing message assumption A3, we can tell when all messages have been received
  - this can be implemented by using synchronized rounds
- Now traitor can be detected!
  - e.g. 2 correctly signed values $\Rightarrow$ general is traitor
Algorithm $SM(m)$

- example, general is traitor

```
<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack:0</td>
</tr>
<tr>
<td>retreat:0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lieutenant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack:0:1</td>
</tr>
<tr>
<td>retreat:0:2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lieutenant 2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack:0</td>
</tr>
<tr>
<td>attack:0:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lieutenant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack:0:1</td>
</tr>
<tr>
<td>retreat:0:2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lieutenant 2</th>
</tr>
</thead>
</table>
```

Algorithm $SM(m)$

- example, lieutenant 2 is traitor

```
<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack:0</td>
</tr>
<tr>
<td>attack:0:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lieutenant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack:0:1</td>
</tr>
<tr>
<td>retreat:0:2</td>
</tr>
</tbody>
</table>

| lieutenant 2 |
```
Algorithm SM(m)

◆ example:
  - SM(0)
    » general sends v:0 to all lieutenants
    » processor i receives v:0 \( V_i = \{v\} \)
  - SM(1)
    » each lieut. countersigns and rebroadcasts v:0
    » processor i receives (v:0:1, v:0:2,..., v:0:(N-1))

Algorithm SM(m)

  - case 1: commander loyal, lieutenant j = traitor
    » all values except v:0:j are v

    \[ v \in V_i \quad \forall \text{ loyal lieut. } i \]

    » processor j cannot tamper

    \[ V_i = \{v\} \quad \forall \text{ loyal lieut. } i \]

  - case 2: commander = traitor, => all lieut. loyal
    » all lieutenants correctly forward what they received
      - agreement: yes
      - validity: N/A
Algorithm SM(m)

- e.g.:
  - SM(2)
    - each lieut. countersigns and rebroadcasts all messages from the previous round
    - processor i has/receives
      - \( v:0 \)
      - \( v:0:1, v:0:2, \ldots, v:0:(N-1) \)
    - \( \overline{v:0:1:1}, v:0:1:2, v:0:1:3, \ldots, v:0:1:N-1 \)
    - \( \overline{v:0:2:1}, \overline{\overline{v:0:2:2}}, v:0:2:3, \ldots, v:0:2:N-1 \)
    - \( \ldots \)
    - \( \overline{\overline{v:0:N-1:1}}, v:0:N-1:2, v:0:N-1:3, \ldots, v:0:N-1:N-1 \)

original message
after 1st rebroadcast
after 2nd rebroadcast

Algorithm SM(m)

- case 1: commander loyal, 2 lieutenants are traitors
  - want each loyal lieut to get \( V=\{v\} \)
  - round 0 => all loyal lieuts get \( v \) from commander
  - other rounds:
    - traitor cannot tamper
    - \( => \) all messages are \( v \) or \( \Phi \)
- case 2: commander traitor + 1 lieut. traitor
  - round 0: all loyal lieuts receive \( v:0 \)
  - round 1:
    - traitors send one value or \( \Phi \)
  - round 2:
    - another exchange (in case traitor caused split in last round)
    - traitor still cannot introduce new value
  \( => \) agreement: yes
  validity: N/A
Algorithm SM(m)

◆ Cost of signed message
  – encoding one bit in a code-word so faulty processor cannot “stumble” on it.
  – e.g.
    » unreliability of the system $F_S = 10^{-10}/h$
    » unreliability of single processor $F_P = 10^{-4}/h$
    » want: Probability of randomly generated valid code word

$$P = \frac{10^{-10}}{10^{-4}} = 10^{-6} \approx 2^{-20}$$

  » given $2^i$ valid codewords, want $(20 + i)$ bits/signature
  » e.g. Attack/Retrieve
    => $2^i$
    => 21 bit signature

Agreement

◆ Important notes:
  – there is no way to guarantee that different processors will get the same value from a possibly faulty input device, except having the processors communicate among themselves to solve the Byz.Gen. Problem.
  – faulty input device may provide meaningless input values
    » all that Byz.Gen. solution can do is guarantee that all processors use the same input value.
    » if input is important, then use redundant input devices
    » redundant inputs cannot achieve reliability. It is still necessary to insure that all non-faulty processors use the redundant data to produce the same output.
Agreement

- Implementing BGP is no problem
- The problem is implementing a message passing system that yields respective assumptions, i.e.:
  A1: every message that is sent is delivered correctly
  A2: the receiver of a message knows who send it
  A3: the absence of a message can be detected
  A4: a loyal general’s signature cannot be forged, and any alteration of the contents of his signed messages can be detected. Anyone can verify the authenticity of a general’s signature