Distributed Snapshots

Assume:

- active monitoring: \( p_0 \) requests state info
- FIFO channels
- Each process records its local state and the states of all its incoming channels
Distributed Snapshot Protocol

[Chandy and Lamport]

1. Process $p_0$ starts the protocol by sending itself a take snapshot message.

2. When $p_i$ receives first take snapshot from $p_j$, it records its local state, $\sigma_i$, and sends take snapshot along all its outgoing channels. For each $k$ such that there is a $p_k$-to-$p_i$ channel, $p_i$ sets $\chi_{ki} = \lambda$.

3. Every protocol message that $p_i$ receives is then recorded in the appropriate $\chi_{ki}$, until $p_i$ receives a take snapshot on the channel, at which point $\chi_{ki}$ remains stable.

4. When all $\chi_{ki}$'s are stable, $p_i$ sends $\sigma_i$ and the $\chi_{ki}$'s to $p_0$. 
Correctness of Protocol

- Termination is guaranteed
- Remains to show global state constructed is consistent
- Show a stronger claim: If algorithm was initiated at $\Sigma^i$, terminated at $\Sigma^f$, and reports $\Sigma^s$, then

\[ \Sigma^i \rightsquigarrow \Sigma^s \rightsquigarrow \Sigma^f \]

where $\rightsquigarrow$ denotes “reachable.” Thus, the reported state could have been reached in some run that leads from $\Sigma^i$ to $\Sigma^f$
Correctness Proof

- for every $i$, let $e_i^*$ denotes the event when $p_i$ receives its first take snapshot
- Every $p_i$-event that causally precedes $e_i^*$ is pre-recording event, and every $p_i$-event that causally follows $e_i^*$ is post-recording event
- Assume “real” run is $r$
- Swap consecutive global states of $r$, maintaining causality, until all pre-recording events precede all post-recording events, and consistent run $r'$ is obtained such that $\Sigma^i \leadsto_{r'} \Sigma^s \leadsto_{r'} \Sigma^f$
Consider adjacent $\langle e, e' \rangle$ of $r$ such that $e$ is a post-recording event and $e'$ is a pre-recording event. If $e \not\to e'$ they can be swapped. Assume $e \to e'$, either:

1. Both $e$ and $e'$ belong to the same process. Then, since they are post-/pre- recording events, $e' \to e$ contradicting $e \to e'$, OR

2. $e$ is a $\text{snd}_i$ event and $e'$ is its corresponding $\text{rcv}_j$. But, since $e$ is a post-recording event, $p_i$ had sent $p_j$ a take snapshot before $e$, and, since the channel is FIFO, the message would have been received before $e'$. Thus, $e'$ is also a post-recording event

THUS, $e \not\leftrightarrow e' \implies$ they can be swapped!
Stable Predicates

- Stable Predicates: Once true remain true
- Examples: deadlock, termination, and loss of tokens
- If distributed snapshot protocol results in states $\Sigma^i \rightarrow \Sigma^s \rightarrow \Sigma^f$ as above, then for a stable $\Phi$:
  \[
  \Sigma^s \models \Phi \implies \Sigma^f \models \Phi \\
  \Sigma^s \models \neg \Phi \implies \Sigma^i \models \neg \Phi
  \]
- Thus, DSP good vehicle for evaluating stable predicates
- What about other predicates? (what good is knowing that $\Phi$ may have held?)
Encryption

Def. Algorithms that protect confidentiality of data.

- $\{m\}_k$ denotes the message $m$ encrypted with the key $k$
- $\{m\}_{k^{-1}}$ denotes the message $m$ decrypted with the key $k^{-1}$

- **Symmetric Encryption** algorithms use the same key for encryption and decryption (the key has to be kept secret!)

- **Asymmetric (Public Key) Encryption** algorithms use different keys for encryption and decryption; the encryption key is usually made public while the decryption key remains private
“Cryptography is rarely ever the solution to a security protocol. . . Cryptography can enhance computer security; it is not a substitute for computer security”

- Where are keys generated?
- How are they generated?
- Where are they stored?
- Where are they used?
- How are they revoked/replaced?
More on Keys

- **Session keys** are keys that are used for one communication session and then discarded;
- Session keys are not necessarily stored
- Often (more expensive) public key cryptography is used to distribute session keys
- Often, a trusted server Trent is assumed. T represents a secure database that is public and write-protected (from others). T can be used as a public-key directory, or as a generator for “good” session keys
- Each participant A may have a secure *long-term key* with T, denoted by $K_A$
What are Security Protocols?

- **Security Protocols** are:
  - short conventional sequences of messages
  - use cryptography

- The **goals** of security protocols include:
  - creating **secure** channels
  - agreeing on a new **shared secret**
  - **authentication**

- They are **frequently wrong**
Example: Needham Schroeder SK

- One of the earliest protocols
- Forms the basis of the *Kerberos* authentication protocol
- Uses purely symmetric encryption algorithms
- Enables A and B to set up a secure channel using T
- A and B share a private, *long-term key* with T
NSSK: The protocol

A · B · N_a

\{ B · N_a · k · \{ k · A \} K_b \} K_a

A

B

T

\{ k · A \} K_b

\{ N_b \} k

\{ N_b - 1 \} k

NYU G22.2631-001 – p.12
In English...

- A informs T she wants to communicate with B and supplies a fresh nonce; Intruder may learn that A wishes to communicate with B but cannot alter message effectively.

- T sends A an encrypted message that includes her request, the new key $k$, and an encrypted message, $T_b$, to B that includes the new session key $k$.

- A sends B the encrypted message $T_b$.

- B sends A a fresh nonce encrypted with $k$.

- A decrements B’s nonce, encrypts it with $k$, and sends it to B (So that the B is assured A generated message).
A “more realistic" NSSK

Where $T_b = \{k \cdot A\}_{K_b}$. 
NSPK (1978)

Intended Run:
NSPK: Undesirable Run

\[ \{ N_a \cdot A \} \]_{K_P} \quad \rightarrow \quad \{ N_a \cdot A \} \]_{K_B} \\
A \quad \downarrow \quad P \quad \downarrow \quad B

\{ N_a \cdot N_b \} \]_{K_A} \\
\downarrow \quad \downarrow

\{ N_b \} \]_{K_P} \quad \rightarrow \quad \{ N_b \} \]_{K_B} \\
P \quad \downarrow \quad P \quad \downarrow

Due to Gavin Lowe (1995)
Diagnosis of a Failure

- **Who was duped?**
  - **not A**— She meant to share \( N_a \cdot N_b \) with \( P \)
  - **B**— Thinks he shares \( N_a \cdot N_b \) only with \( A \)

- **Secrecy failed**: \( P \) knows values

- **Authentication failed**: \( A \) had no run with \( B \)

- **A** offered \( P \) a service:
  - gave \( P \) nonce \( N_a \)
  - promised to translate \( \{ N_a \cdot N \}_K_A \) into \( \{ N \}_K_P \)

- But this service was **unintended**!!
  (At least not when \( N \) is computed by a legitimate party and \( P \) doesn’t even know it)
**Conclusions**

- **Security Protocols** are:
  - short conventional sequences of messages
  - use cryptography

- The **goals** of security protocols include:
  - creating **secure** channels
  - agreeing on a new **shared secret**
  - authentication

- They are **frequently wrong**
Why are Security Protocols Hard?

- Attacker chooses pattern of communication
- Attacker may also be a player
  - May hold keys
  - Will misuse them
- Attackers *manipulate* honest players
- Honest players *play by the rules*
- But they may be forced to serve as oracles
- Protocol may allow for *unindented services*