Key Refreshing in Wireless Sensor Networks

Simon R. Blackburn[1] Keith M. Martin[1] Maura B. Paterson[1]
Douglas R. Stinson[2]

[1]Information Security Group
Royal Holloway, University of London

[2]David R. Cheriton School of Computer Science
University of Waterloo

ICITS 2008

Outline

Forward Security Through Key Refreshing

Key Refreshing Schemes

Resynchronisation Schemes

Forward Security in Sensor Networks

- Forward security: An adversary who captures a key at time t should not be able to decrypt messages sent with earlier versions of that key.
- Forward security can be obtained through key refreshing.
- Klonowski, Kutyłowski, Ren, Rybarczyk (2007); Mauw, van Vessen, Bos (2006) studied key refreshing for networks in which each key is shared by a pair of nodes.
- Keys used in sensor networks may be shared by more than two nodes.

A Standard Technique for Key Refreshing

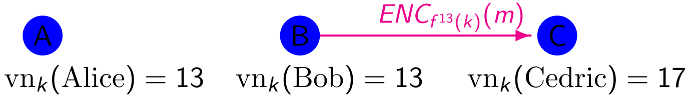
$\mathcal{K}$ keyspace
$f : \mathcal{K} \to \mathcal{K}$ one-way function
$k \in \mathcal{K}$ key

$$k \mapsto f(k)$$

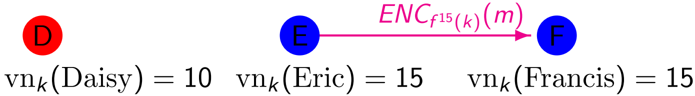
- Define $\mathrm{vn_k(X) = i}$ if user X possesses the key $f^i(k)$.
- Alice and Bob can update a shared key k every time a message is sent, so that $\mathrm{vn_k(Alice) = vn_k(Bob)}$ at all times.
- What happens when other users possess the same key?

Difficulties in Refreshing Widely Shared Keys

- undecipherable messages



- degradation of forward security



Schemes to Synchronise Key Refreshing

Appropriate techniques will depend on the network environment.

- Synchronous key refreshing
  - event-driven
  - flooded
- Asynchronous
  - periodic resynchronisation
  - resynchronisation by a flood
  - resynchronisation via a leader election

Networks with Synchronised Clocks

Scheme (Event-driven refreshing)

Nodes refresh their keys in response to some event that can be observed by the whole network

- Clock synchronisation comes at a cost, but may be required by the application. *e.g.* intruder detection, volcano monitoring
- This form of refreshing incurs no overheads.

Networks with Frequent Flooding

Scheme (Flooded refreshing)

1. *Before initiating a flood, a node updates its keys.*
2. *A node that receives a flooded message must update the appropriate key in order to decrypt it; it similarly updates all its keys before forwarding the message.*
3. *A node keeps a given version of each key until it broadcasts a message using a higher version number.*

- As long as neighbours do not broadcast simultaneously this prevents undecipherable messages.
- Most appropriate for networks in which flooding is frequent. *e.g.* a disaster recovery scenario in which real-time updates are flooded to sensors attached to medical personnel

Asynchronous Key Refreshing

Scheme (Message-driven refreshing)

1. *Alice and Bob exchange* $\mathrm{vn}_k(\mathrm{Alice})$ *and* $\mathrm{vn}_k(\mathrm{Bob})$.
2. $$\mathrm{newvn} = 1 + \max\{\mathrm{vn}_k(\mathrm{Alice}), \mathrm{vn}_k(\mathrm{Bob})\}$$
3. *Alice and Bob apply f to k repeatedly until* $\mathrm{vn}_k(\mathrm{Alice}) = \mathrm{vn}_k(\mathrm{Bob}) = \mathrm{newvn}$.

- Works well if all nodes are more-or-less equally active.
- To avoid degradation of forward security, it is necessary to resynchronise the version numbers held throughout the network.

Infrequent Network-Wide Events

Scheme

Nodes update keys to a specified version number when the event is detected.

- *e.g.* nodes could update to version number $100j$ after the j^{th} occurrence of the event.
- Suitable when the amount of traffic between occurrences of the event does not vary greatly and can be reasonably estimated.
- *e.g.* an intruder detection system may be armed/disarmed by a flooded message triggered by the locking/unlocking of a door.
- Requires no communication overheads.

Infrequent Local Events

Scheme

The first node to reach a specified version number triggers a flood prompting all nodes to update their keys to that number.

- Suitable when there are no network-wide events and the network can only support occasional flooding.
- This is the case for networks measuring events that occur locally, and in which there is a low amount of (mostly local) communication between nodes.

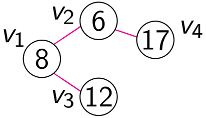
Regular Disconnection

Scheme

Nodes periodically execute a leader election protocol to determine the highest version number, then update their keys correspondingly.

- Could be used in a network that is temporarily disconnected, to resynchronise the version numbers held in the various components once connectivity is reestablished.
- It can be achieved in time $O(D)$ with message complexity $O(DE)$ (where D is the network diameter and E the number of edges,) using a variant of an algorithm due to Peleg.

Leader Election



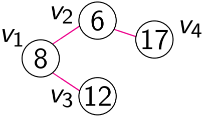
nodes send tuples (s, y, d, v)

- s=node id
- y=id of current most distant node
- d=max current distance
- v=highest known version number

- nodes broadcast tuple in response to first broadcast they receive
- once they receive responses from all their neighbours they broadcast updated tuple
- initiating node sends terminating condition when it receives tuples with identical d values in two consecutive pulses

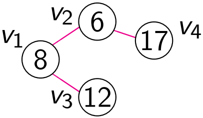
Leader Election

	1	2	3	4
send	$(1, 1, 0, 8)$			



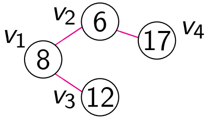
Leader Election

	1	2	3	4
send	$(1,1,0,8)$			
receive		$(1,1,0,8)$	$(1,1,0,8)$	
send		$(2,2,1,8)$	$(3,3,1,12)$	



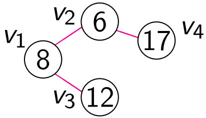
Leader Election

	1	2	3	4
send	$(1,1,0,8)$			
receive		$(1,1,0,8)$	$(1,1,0,8)$	
send		$(2,2,1,8)$	$(3,3,1,12)$	
receive	$(2,2,1,8)$ $(3,3,1,12)$			$(2,2,1,8)$
send	$(1,2,1,12)$			$(4,4,2,17)$

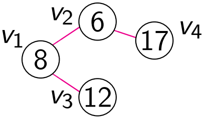


Leader Election

	1	2	3	4
send	$(1,1,0,8)$			
receive		$(1,1,0,8)$	$(1,1,0,8)$	
send		$(2,2,1,8)$	$(3,3,1,12)$	
receive	$(2,2,1,8)$			$(2,2,1,8)$
	$(3,3,1,12)$			
send	$(1,2,1,12)$			$(4,4,2,17)$
receive		$(1,2,1,12)$	$(1,2,1,12)$	
		$(4,4,2,17)$		
send		$(2,4,2,17)$	$(3,3,1,12)$	

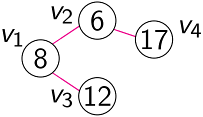


Leader Election



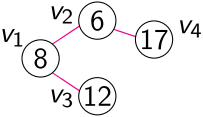
	1	2	3	4
send	$(1,1,0,8)$			
receive		$(1,1,0,8)$	$(1,1,0,8)$	
send		$(2,2,1,8)$	$(3,3,1,12)$	
receive	$(2,2,1,8)$			$(2,2,1,8)$
	$(3,3,1,12)$			
send	$(1,2,1,12)$			$(4,4,2,17)$
receive		$(1,2,1,12)$	$(1,2,1,12)$	
		$(4,4,2,17)$		
send		$(2,4,2,17)$	$(3,3,1,12)$	
receive	$(2,4,2,17)$			$(2,4,2,17)$
	$(3,2,1,12)$			
send	$(1,4,2,17)$			$(4,4,2,17)$

Leader Election



	1	2	3	4
send	$(1,1,0,8)$			
receive		$(1,1,0,8)$	$(1,1,0,8)$	
send		$(2,2,1,8)$	$(3,3,1,12)$	
receive	$(2,2,1,8)$ $(3,3,1,12)$			$(2,2,1,8)$
send	$(1,2,1,12)$			$(4,4,2,17)$
receive		$(1,2,1,12)$ $(4,4,2,17)$	$(1,2,1,12)$	
send		$(2,4,2,17)$	$(3,3,1,12)$	
receive	$(2,4,2,17)$ $(3,2,1,12)$			$(2,4,2,17)$
send	$(1,4,2,17)$			$(4,4,2,17)$
receive		$(1,4,2,17)$ $(4,4,2,17)$	$(1,4,2,17)$	
send		$(2,4,2,17)$	$(3,4,2,17)$	

Leader Election



	1	2	3	4
send	$(1,1,0,8)$			
receive		$(1,1,0,8)$	$(1,1,0,8)$	
send		$(2,2,1,8)$	$(3,3,1,12)$	
receive	$(2,2,1,8)$ $(3,3,1,12)$			$(2,2,1,8)$
send	$(1,2,1,12)$			$(4,4,2,17)$
receive		$(1,2,1,12)$ $(4,4,2,17)$	$(1,2,1,12)$	
send		$(2,4,2,17)$	$(3,3,1,12)$	
receive	$(2,4,2,17)$ $(3,2,1,12)$			$(2,4,2,17)$
send	$(1,4,2,17)$			$(4,4,2,17)$
receive		$(1,4,2,17)$ $(4,4,2,17)$	$(1,4,2,17)$	
send		$(2,4,2,17)$	$(3,4,2,17)$	
receive	$(2,4,2,17$ $(3,4,2,17)$			$(2,4,2,17)$
send	$(0,4,2,17)$			$(4,4,2,17)$

Summary of Techniques

Scheme	Required Network Properties	Suitable Application Environments
Key Refreshing		
Mauw *et al.*	nodes communicate directly with the base station	
Klonowski *et al.*	keys are shared by pairs of nodes	
1. Event-driven	frequent occurrence of a network-wide event	synchronised clocks
2. Flooded	frequent flooding of messages	frequent flooding
3. Message-driven	-	any

Summary of Techniques

Scheme	Required Network Properties	Suitable Application Environments
Resynchronisation		
Periodic	occasional network-wide event	infrequent network-wide events
Flooded	capable of supporting occasional flooded messages	infrequent local events
Leader Election	-	regular disconnection

Thank you!