Self-Healing Key Distribution*

Dirk Balfanz, Drew Dean, Matt Franklin, Mike Malkin,
Sara Miner and Jessica Staddon

GSEC, December 10, 2001

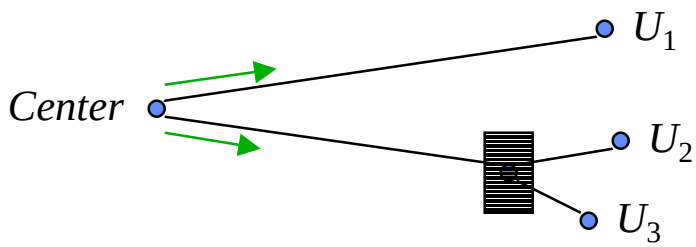
* This work was partially supported by DARPA grant N66001-00-1-8921

Xerox Palo Alto Research Center
parc

Outline

- The problem
- Previous solutions
- The self-healing approach
  - Two new mechanisms
    - Self-healing from secret sharing
    - Distribution of distinct keys with revocation
  - The combined scheme
  - Extensions and Comments
    - A long-lived approach

The Session Key Distribution Problem

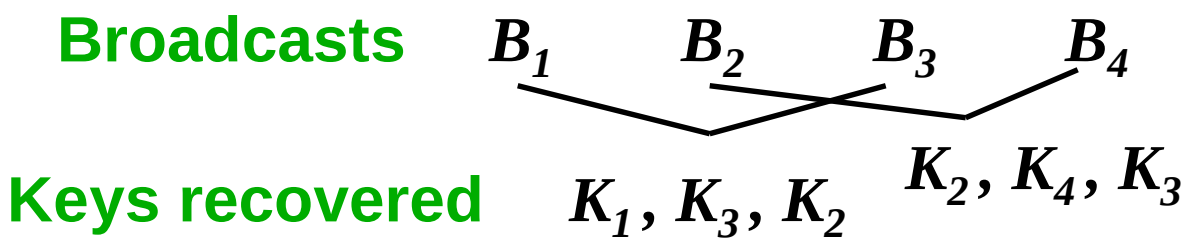


- Center broadcasts keying information at beginning of each of m sessions, active users can construct current session key
- Messages exchanged during a session are encrypted with that session's key
- *Challenge*: In the presence of packet loss, can users recover session keys in a *noninteractive* manner?
  - No additional network load
  - No risk of group member exposure

Comparison with previous solutions

- Recovery from loss of packets relies solely on packets sent later
  - [Wong-Lam, 2000] Redundantly encode information about past keys in later “repair” packets
  - [Perrig-Song-Tygar,2001] Trade-off overhead with computational load, “repair” packets become smaller “hints”
  - Care must be taken to ensure new users can’t recover old keys
  - Stateful, hierarchical key management is used
    - Cheating is possible through untraceable keys
- Self-healing approach ties key recovery to membership status
  - Must be a member before and after a session in order to recover that session’s key
  - Broadcasts only depend on current set of revoked users (no need to save state)
  - Cheating is only possible by giving away personal (i.e. traceable) keys unless coalitions are large

The self-healing approach



"sandwiching" property

- Ability to recover is closely tied to membership status
  - Minimum delay possible in dynamic group
- Self-healing procedure is computationally efficient
  - Polynomial interpolation
- Resilient: approach is resistant to collusion

Self-healing via secret sharing

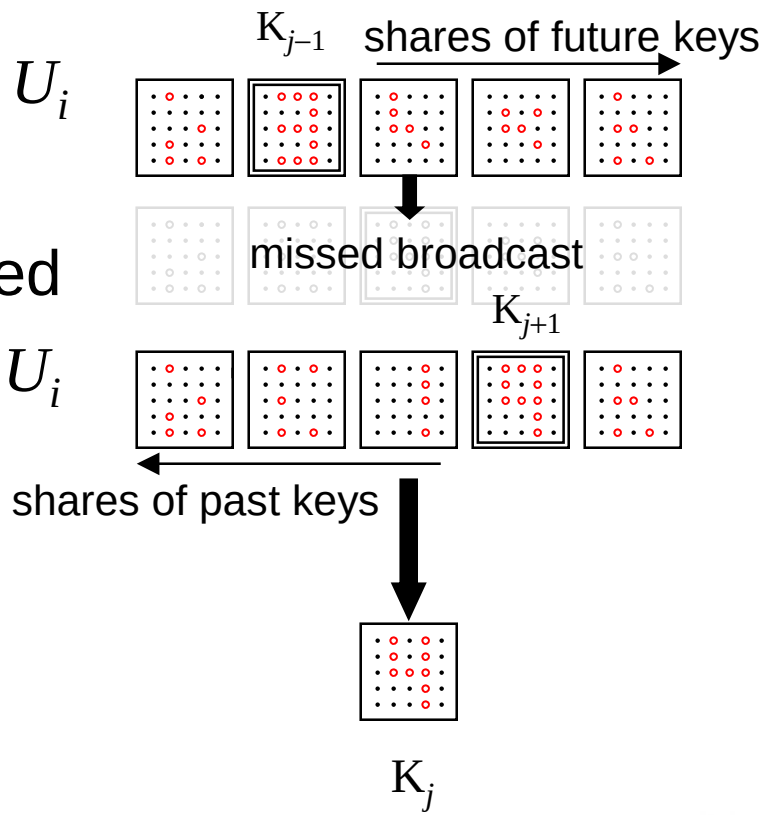
- From each broadcast, users recover current session key and shares of past and future session keys
  - Shares are a 2-out-of-2 sharing of session keys [Shamir `79]
- The shares users recover must be different to ensure resiliency

Self-healing via secret sharing (cont'd)

Shares recovered by U_i from B_{j-1}

B_j is not received

Shares recovered by U_i from B_{j+1}



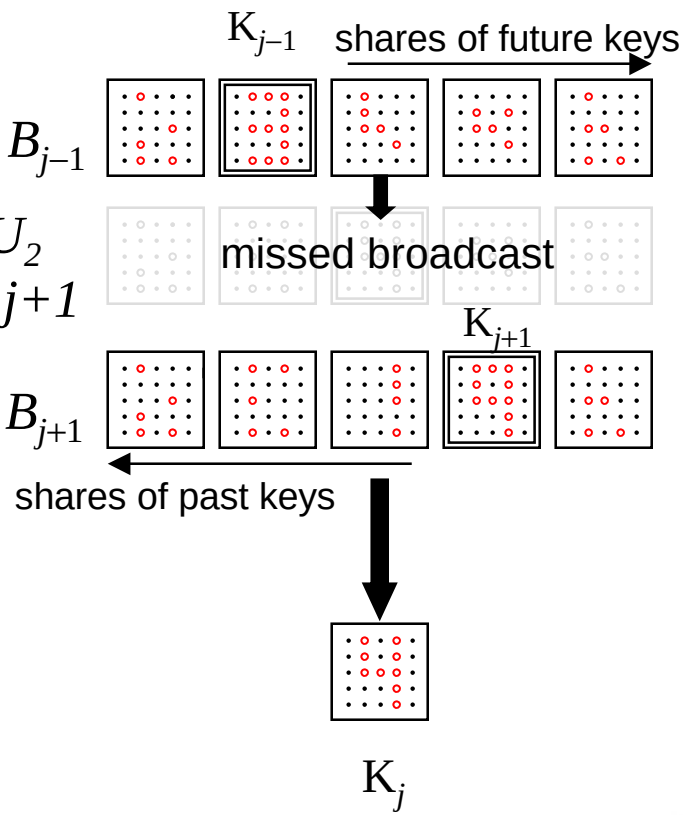
Self-healing via secret sharing (cont'd)

Users' shares must be different to ensure enough resiliency

Shares recovered by U_1 from B_{j-1}

In session j, U_1 is revoked, U_2 joins at beginning of session $j+1$

Shares recovered by U_2 from B_{j+1}



Self-healing via secret sharing (cont'd)

- **Set-up** Center chooses at random $2m$ polynomials of degree t: $h_1,\ldots,h_m,p_1,\ldots,p_m$,and $K_1,\ldots,K_m$. For $s = 1,\ldots,m$, let $q_s(x)=K_s-p_s(x)$. U_i stores the personal key, $\{h_1(i),\ldots,h_m(i)\}$

- **Broadcast** For $j = 1,\ldots,m$, broadcast B_j:

$$\{h_1(x)+p_1(x),\ldots,h_{j-1}(x)+p_{j-1}(x),\ h_j(x)+K_j,h_{j+1}(x)+q_{j+1}(x),\ldots,h_m(x)+q_m(x)\}$$

- **Self-healing** By U_i:

Shares of future keys →

From B_{j-1}: $p_1(i),\ldots,p_{j-2}(i)$, K_{j-1}, $q_j(i), q_{j+1}(i), q_{j+2}(i),\ldots,q_m(i)$

← Shares of past keys

$+$

From B_{j+1}: $p_1(i),\ldots,p_{j-2}(i), p_{j-1}(i), p_j(i)$, K_{j+1}, $q_{j+2}(i),\ldots,q_m(i)$

$=$

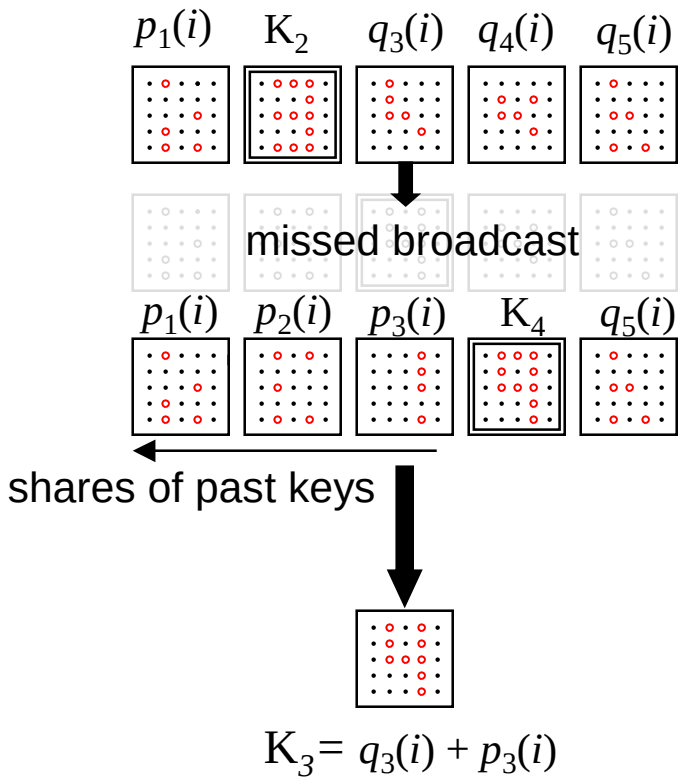
K_j

Self-healing via s sharing (cont'd)

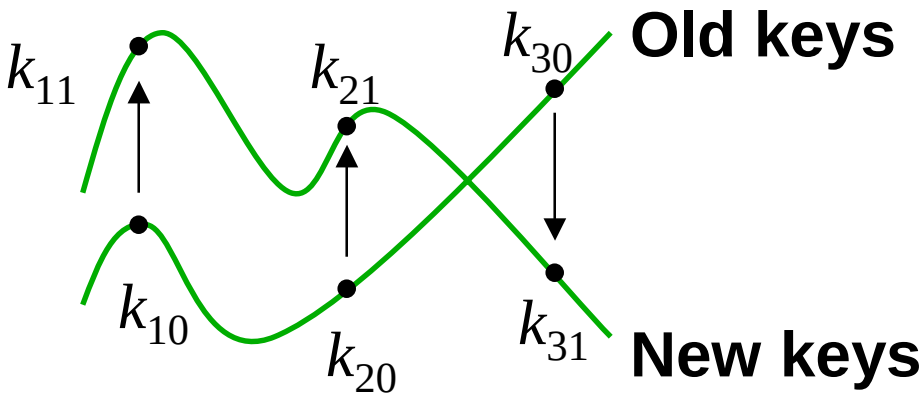
Referring back to the intuitive diagram…

U_i recovers from B_2

U_i recovers from B_4



Distributing distinct keys with revocation*



- User U_i stores k_{i0}
  - If the keys $\{k_{i0}\}_i$ are t-wise independent, we can represent them as points on a polynomial of degree t
- *Goal*: Through a short broadcast, distribute a point on a *new* polynomial to each user
  - Achieve the maximum possible resiliency

*This is an extension of a Naor-Pinkas [NP `00] method for distributing the *same* key with revocation.

Distributing distinct keys with revocation (cont'd)

- First, distributing a common key, T, [NP`00]:
  - Set-up: Let $s(x)$ be a polynomial of degree t, U_i stores $s(i)$. Let W denote set of t revoked users.
  - Broadcast: $\{s(0)+T\} \cup \{(w, s(w)) : w \in W\}$
  - Key recovery: U_i knows $t+1$ points on $s(x)$ so can reconstruct $s(0)$ and solve for T
- Why doesn't this work for distinct keys, $\{T_i\}_i$?
  - Define $f(x)$ such that $f(i) = T_i$, for all i
  - Broadcast $\{f(x) + s(0)\} \cup \{s(w) : w \in W\}$
  - Recovery U_i learns $f(x)$, $\boldsymbol{U_i}$ **learns all new keys**

Distributing distinct keys with revocation (cont'd)

- Set-up: Center chooses a random bivariate polynomial: $s(x,y)=a_{00} + a_{10}x + a_{01}y + \ldots + a_{tt}x^t y^t$
  - U_i stores personal key $s(i,i)$
  - New keys are points on poly $f(x)$
- Broadcast: $\{ f(x) + s(N,x) \} \cup \{ (w, s(w,x)) : w \in W \}$
  - W, set of t indices of the revoked users
  - N, an integer larger than any user index
- Key Recovery:
  - U_i has $t+1$ points on the poly $s(x,i)$
  - $f(x)+s(N,x)|_{x=i} - s(N,i) = f(i)$

Self-healing key distribution

- Self-healing via distribution of shares of session keys
  - Distinct shares ensure resiliency
- Key distribution mechanism for share distribution
  - Each user recovers a different set of shares

Self-healing key distribution (cont'd)

- Set-up For $j = 1,\ldots,m$, and $r = 1,\ldots,m$
  - Center chooses $s_{rj}(x,y) = a_{00}+\ldots+a_{t\text{-}1,t\text{-}1}x^t y^t$
  - U_i stores $\{s_{1j}(i,i),\ldots,s_{mj}(i,i)\}_j$
  - Center chooses $\{p_j(x)\}_j$, $\{K_j\}_j$, this determines $\{q_j(x)\}_j$
- Broadcast
  - Let W be the set of t revoked users, for $j = 1,\ldots,m$:

$$B_j=\{p_1(x) + s_{1,j}(N,x),\ldots, p_{j-1}(x) +s_{j-1,j}(N,x), K_j+s_{j,j}(N,x), q_{j+1}(x)+ s_{j+1,j}(N,x),\ldots, q_m(x)+s_{mj}(N,x)\} \cup\{s_{rj}(w,x) : w\in W, r = 1,..,m\}$$

- Key and shares recovery
  - Active user, U_i, can reconstruct $\{s_{rj}(x,i)\}_r$, evaluate at $x = N$, and recover session key and unique shares

Self-healing key distribution (cont'd)

Why self-healing?

$\{s_{rj}(N,i): r = 1,..,m\}$ ⟵ $\{s_{rj}(x,i): r = 1,..,m\}$

$+$ ⟵ $\{s_{rj}(i,i): r = 1,..,m\}$

$$B_j: \left\{ \begin{array}{c} p_1(x) + s_{1,j}(N,x),\ldots, p_{j-1}(x) + s_{j-1,j}(N,x) \\ K_j + s_{j,j}(N,x) \\ q_{j+1}(x) + s_{j+1,j}(N,x),\ldots, q_m(x) + s_{mj}(N,x) \end{array} \right\} + \cup\{s_{rj}(w,x): w \in W,\ r = 1,..,m\}$$

$$p_1(i),\ldots,p_{j-1}(i),\ K_j,\ q_{j+1}(i),\ldots,q_m(i)$$

Shares of past keys ⟵ | ⟶ Shares of future keys

Self-healing key distribution (cont'd)

Why t-resilient?

Resiliency comes from the degrees of the polynomials

Consider $U_1,\ldots,U_t$

- $U_1,\ldots,U_t$ can't break the revocation mechanism
  - At most learn $s_{rj}(x,1),\ldots,s_{rj}(x,t)$
  - For $v > t$, have no information on $s_{rj}(N,v)$
  - Shares are masked differently in each session
    - Can't recover useful masking function through shares
- $U_1,\ldots,U_t$ can't abuse self-healing
  - Need $t+1$ points on $p_j(x)$ or $q_j(x)$ to determine another user's share

Comments

- Communication overhead depends on m and t, **not** n
  - Communication overhead is $O(mt^2+mt)$ times session key size
  - User storage is $O(m^2)$ times session key size
  - Have variant with communication overhead $O(t^2+mt)$ times session key size and a moderate amount of additional work by group members
- Optimality results
  - Communication overhead is at least $\max\{t^2, mt\}$ times session key size for unconditional security
    - The key distribution portion requires $O(t^2)$ and self-healing requires $O(mt)$

Extending the lifetime of this approach

What do we do after the m sessions are over?

- Can't simply restart the scheme
  - Users know $\{s_{rj}(x,i) : r = 1,..,m\}$
  - Users know $\{s_{rj}(w,w) : w \in W,\ r = 1,..,m\}$

A solution: With modular exponentiations can *control* what users learn

- Technique originated with Feldman [F`87] and is used in [NP`00]
  - All operations occur in the exponent
  - Users start with a base set of m^2 keys and they evolve over time

Extending the lifetime of this approach (cont'd)

- $f_v(x) = a_0+a_1x+\ldots a_tx^t$, let $g^{f_v(x)} = (g^{a_0}, g^{a_1},\ldots,g^{a_t})$
- The basic idea:
  - U_i stores $s(i,i)$. Goal is distribution of points on g^{f_v}
  - For the vth session
    - $s_v(x,y) = r_v \cdot s(x,y)$
    - Broadcast $\{g^{r_v}, g^{f_v(x)+s_v(N,x)}\} \cup \{g^{s_v(w,x)} : w \in W\}$
    - Recovery U_i learns $g^{s_v(x,i)}$ but doesn't know r_v
      - U_i's new key is $g^{f_v(i)}$
      - g^{r_v} and $g^{s(x,i)}$ doesn't imply knowledge of $g^{s_v(x,i)}$

Extending the lifetime of this approach (cont'd)

A long-lived self-healing scheme

- Set-up For $j = 1,\ldots,m$

  Center chooses $h_{ij}(x,y) = a_{00}+\ldots+a_{t-1,t-1}x^t y^t$, U_i stores $\{h_{1j}(i,i),\ldots,h_{mj}(i,i)\}_j$

  Center chooses $\{p_j(x)\}_j$, $\{K_j\}_j$, this determines $\{q_j(x)\}_j$

  Note: The session keys will now be $g^{K_1},\ldots,g^{K_m}$

- Broadcast in the vth set of m sessions

  Let W be the set of t revoked user, for $j = 1,\ldots,m$:

  $B_j=\{g^{r_v}, g^{p_1(x)+s_{1,j}(N,x)},\ldots, g^{p_{j-1}(x)+s_{j-1,j}(N,x)}, g^{K_j+s_{j,j}(N,x)}, g^{q_{j+1}(x)+s_{j+1,j}(N,x)},\ldots, g^{q_m(x)+s_{mj}(N,x)}\}$
  $\cup\{g^{s_{ij}(w,x)} : w\in W,\ i = 1,..,m\}$

  $s_{xj}(x,y) = r_v \cdot h_{xj}(x,y)$, for $x = 1,\ldots,m$ (r_v is randomly chosen)

- Key and shares recovery

  Active user, U_i, can construct $g^{s_{xj}(i,i)}$

  From B_j, learns $\{g^{p_1(i)},\ldots, g^{p_{j-1}(i)}, g^{K_j}, g^{q_{j+1}(i)},\ldots, g^{q_m(i)}\}$

  $g^{p_j(i)}\, g^{q_j(i)} = g^{p_j(i)+q_j(i)} = g^{K_j}$

Final Comments

- Can use similar ideas to realize “sliding window” self-healing
  - Can always recover from loss if receive two “sandwiching” packets no more than m sessions apart
- Other applications: Distributing *content*
  - Provides reliable, noninteractive distribution of real-time content