Cryptographically-Enforced Hierarchical Access Control with Multiple Keys

Jason Crampton
Information Security Group
Royal Holloway, University of London

NordSec 2007
Hierarchical access control

Given

- a partially ordered set of security labels \((L, \leq)\)
- a function \(\lambda\) mapping users and protected objects to \(L\)

we require that a user \(u\) can only read \(o\) if \(\lambda(u) \geq \lambda(o)\)
Cryptographic hierarchical access control

Useful for third-party data publishing

- Data is made available by someone other than data owner
Cryptographic hierarchical access control

Useful for third-party data publishing

- Data is made available by someone other than data owner

Trivial solution

- Encrypt $o$ with $k(o)$
- Send \{ $k(y) : y \leq \lambda(u)$ \} to $u$
Cryptographic hierarchical access control

Useful for third-party data publishing

- Data is made available by someone other than data owner

Trivial solution

- Encrypt $o$ with $k(o)$
- Send $\{k(y) : y \leq \lambda(u)\}$ to $u$

Preferably

- Encrypt $o$ with $k(o)$
- Send $k(\lambda(u))$ to $u$
Cryptographic hierarchical access control

Useful for third-party data publishing

- Data is made available by someone other than data owner

Trivial solution

- Encrypt $o$ with $k(o)$
- Send $\{k(y) : y \leq \lambda(u)\}$ to $u$

Preferably

- Encrypt $o$ with $k(o)$
- Send $k(\lambda(u))$ to $u$
- Publish additional (encrypted) information enabling $u$ to derive $k(y)$ for all $y \leq k(\lambda(u))$
Iterative key assignment schemes

The data owner

- chooses $k(y)$ at random from the key space
Iterative key assignment schemes

The data owner

- chooses $k(y)$ at random from the key space
- publishes $\{E_{k(x)}(k(y)) : y \preceq x, x, y \in L\}$
  - $y \preceq x$ denotes that $y$ is an immediate child of $x$ in $L$
  - $E_k(m)$ denotes the encryption of message $m$ with key $k$
Iterative key assignment schemes

The data owner

- chooses $k(y)$ at random from the key space
- publishes $\{E_{k(x)}(k(y)) : y \prec x, x, y \in L\}$
  - $y \prec x$ denotes that $y$ is an immediate child of $x$ in $L$
  - $E_k(m)$ denotes the encryption of message $m$ with key $k$
- Sends $k(\lambda(u))$ to user $u$
Iterative key assignment schemes

The data owner

- chooses \( k(y) \) at random from the key space
- publishes \( \{E_k(x)(k(y)) : y \preceq x, x, y \in L\} \)
  - \( y \preceq x \) denotes that \( y \) is an immediate child of \( x \) in \( L \)
  - \( E_k(m) \) denotes the encryption of message \( m \) with key \( k \)
- Sends \( k(\lambda(u)) \) to user \( u \)

The user obtains

- \( k(y), y \preceq \lambda(u) \), by decrypting the appropriate datum of public information
Iterative key assignment schemes

The data owner

- chooses $k(y)$ at random from the key space
- publishes $\{E_{k(x)}(k(y)) : y \preceq x, x, y \in L\}$
  - $y \preceq x$ denotes that $y$ is an immediate child of $x$ in $L$
  - $E_{k}(m)$ denotes the encryption of message $m$ with key $k$
- Sends $k(\lambda(u))$ to user $u$

The user obtains

- $k(y), y \preceq \lambda(u)$, by decrypting the appropriate datum of public information
- $k(y), y < \lambda(u)$, by iteratively decrypting keys on some path between $\lambda(u)$ and $y$
Problem statement and motivation

Design a hierarchical key assignment scheme that supports multiple keys for each security label

- Minimize public storage
- Minimize number of keys distributed to users
Problem statement and motivation

Design a hierarchical key assignment scheme that supports multiple keys for each security label

- Minimize public storage
- Minimize number of keys distributed to users

Such schemes are useful for

- supporting lazy re-encryption
- enforcing hierarchical access control policies
Design decisions

- *Is the number of updates determined in advance?*
  If yes, we say the scheme is **bounded** (and **unbounded** otherwise)

- *Can the key for label x be updated independently of the one for y?*
  If yes, we say the scheme is **asynchronous** (and **synchronous** otherwise)
Preliminaries

A bounded asynchronous scheme

An unbounded asynchronous scheme

Concluding remarks

Questions
Illustration

Initial security lattice – one key per label
Illustration

$k(d)$ is updated – two keys for $d$
Illustration

$k(b)$ is updated – two keys for $b$, three for $d$
Basic approach

There are two “orthogonal” dimensions to the problem

- The set of security labels $L$
- Temporal – a chain of keys associated with each element of $L$
Basic approach

There are two “orthogonal” dimensions to the problem

- The set of security labels $L$
- Temporal – a chain of keys associated with each element of $L$

One solution is to

- construct an iterative key assignment scheme for $L$
- define a “key chain” for the temporal dimension
Hash chain solution

For a bounded scheme there are at most $m$ keys for each label
Hash chain solution

For a bounded scheme there are at most $m$ keys for each label

- Define an iterative key assignment scheme for $L$
- For each $x \in L$
  
  - choose a key $k_m(x)$ from $\{0, 1\}^l$
  
  - define $k_{i-1}(x) = h(k_i(x))$, where $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ is a suitable (public) hash function
Hash chain solution

For a bounded scheme there are at most $m$ keys for each label

- Define an iterative key assignment scheme for $L$
- For each $x \in L$
  - choose a key $k_m(x)$ from $\{0, 1\}^l$
  - define $k_{i-1}(x) = h(k_i(x))$, where $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ is a suitable (public) hash function
- When the key for label $x$ needs to be changed
  - select the next key for each label $y \leq x$
  - update public information for $L$’s scheme
Hash chain solution

For a bounded scheme there are at most $m$ keys for each label

- Define an iterative key assignment scheme for $L$
- For each $x \in L$
  - choose a key $k_m(x)$ from $\{0, 1\}^l$
  - define $k_{i-1}(x) = h(k_i(x))$, where $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ is a suitable (public) hash function
- When the key for label $x$ needs to be changed
  - select the next key for each label $y \leq x$
  - update public information for $L$’s scheme
- A user with security label $x$ can
  - compute the current key for $y < x$ from the public information for $L$
  - iteratively compute hashes of $y$’s key until the key for the desired time period is obtained
Hash chain solution

For a bounded scheme there are at most $m$ keys for each label

- Define an iterative key assignment scheme for $L$
- For each $x \in L$
  - choose a key $k_m(x)$ from $\{0, 1\}^l$
  - define $k_{i-1}(x) = h(k_i(x))$, where $h : \{0, 1\}^* \to \{0, 1\}^l$ is a suitable (public) hash function
- When the key for label $x$ needs to be changed
  - select the next key for each label $y \leq x$
  - update public information for $L$’s scheme
- A user with security label $x$ can
  - compute the current key for $y < x$ from the public information for $L$
  - iteratively compute hashes of $y$’s key until the key for the desired time period is obtained
- Future keys cannot be feasibly computed (if the hash function is any good!)
Preliminaries

A bounded asynchronous scheme

An unbounded asynchronous scheme

Concluding remarks

Questions
The scheme

We place no upper limit on the number of key updates

- Define an iterative key assignment scheme for $L$
The scheme

We place no upper limit on the number of key updates

- Define an iterative key assignment scheme for $L$
- Compute $(n, e, d)$ using RSA key generator and publish $(n, e)$
We place no upper limit on the number of key updates

- Define an iterative key assignment scheme for $L$
- Compute $(n, e, d)$ using RSA key generator and publish $(n, e)$
- Select a key for each $x \in L$
The scheme

We place no upper limit on the number of key updates

- Define an iterative key assignment scheme for $L$
- Compute $(n, e, d)$ using RSA key generator and publish $(n, e)$
- Select a key for each $x \in L$
- When the key for label $x$ needs to be changed
  - for each $y \leq x$ define the new key to be $(k(y))^d \mod n$
  - compute and replace appropriate public information
Key derivation

Note that

$$(k_i(y))^e = ((k_{i-1}(y)^d)^e = k_{i-1}(y)$$
Key derivation

Note that

\[ (k_i(y))^e = ((k_{i-1}(y)^d)^e = k_{i-1}(y) \]

In other words, the most recent key can be computed from the current one.
Key derivation

Note that

\[(k_i(y))^e = ((k_{i-1}(y)^d)^e = k_{i-1}(y)\]

In other words, the most recent key can be computed from the current one.

A user with security label \(x\) can

- compute the current key for \(y < x\) from the public information
- iteratively compute the key for the desired time period
Key derivation

Note that

\[(k_i(y))^e = ((k_{i-1}(y))^d)^e = k_{i-1}(y)\]

In other words, the most recent key can be computed from the current one.

A user with security label \(x\) can

- compute the current key for \(y < x\) from the public information
- iteratively compute the key for the desired time period

Future keys cannot be feasibly computed (without breaking RSA)
Preliminaries

A bounded asynchronous scheme

An unbounded asynchronous scheme

Concluding remarks

Questions
Using and evaluating schemes

Note that

- any asynchronous scheme can be used as a synchronous scheme
- any unbounded scheme can be used as a bounded scheme
- unbounded asynchronous schemes are particularly suitable for lazy revocation
- bounded synchronous schemes are particularly suitable for temporal access control policies
Using and evaluating schemes

Note that

- any asynchronous scheme can be used as a synchronous scheme
- any unbounded scheme can be used as a bounded scheme
- unbounded asynchronous schemes are particularly suitable for lazy revocation
- bounded synchronous schemes are particularly suitable for temporal access control policies

Ideally a scheme should

- have direct key derivation and low storage costs
- be unbounded and asynchronous
- require a single key for each user
## Summary of schemes

The schemes presented in this talk

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Update</th>
<th>Storage</th>
<th>U</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Public</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td>$Prev$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ind</td>
<td>Dir</td>
<td>Yes</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>Dir</td>
<td>n/a</td>
<td>No</td>
<td>$</td>
<td>L</td>
</tr>
<tr>
<td>Dir</td>
<td>Ind</td>
<td>No</td>
<td>$</td>
<td>L</td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>$q</td>
<td>E</td>
</tr>
</tbody>
</table>
Summary of schemes

A good scheme for temporal access control policies

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Update</th>
<th>Storage Public</th>
<th>Storage Private</th>
<th>U</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Prev</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ind</td>
<td>Dir</td>
<td>Yes</td>
<td></td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td></td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Dir</td>
<td>n/a</td>
<td>No</td>
<td></td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>Dir</td>
<td>Ind</td>
<td>No</td>
<td></td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td></td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>q</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>
## Summary of schemes

### A good scheme for lazy revocation

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Update</th>
<th>Storage</th>
<th>U</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Public</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>Private</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ind</td>
<td>Dir</td>
<td>Yes</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>Dir</td>
<td>n/a</td>
<td>No</td>
<td>$</td>
<td>L</td>
</tr>
<tr>
<td>Dir</td>
<td>Ind</td>
<td>No</td>
<td>$</td>
<td>L</td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>Ind</td>
<td>Ind</td>
<td>Yes</td>
<td>$q</td>
<td>E</td>
</tr>
</tbody>
</table>
Contributions

- First schemes to consider the design of multi-key assignment schemes
- Demonstrate applicability to lazy revocation and temporal access control
- Use both iterative key assignment schemes and Akl-Taylor schemes (see proceedings)
Future work

Formal security analysis

- Multi-key assignment schemes are constructed from components
- Atallah et al & Ateniese et al have undertaken security analyses for key assignment schemes
- Can it be shown that a multi-key scheme is as secure as (the weakest of) its components?
Preliminaries

A bounded asynchronous scheme

An unbounded asynchronous scheme

Concluding remarks

Questions