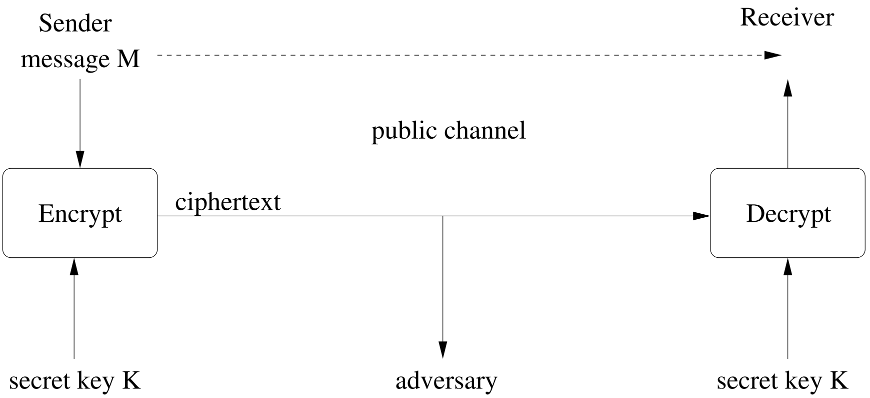
On Symmetric Key Broadcast Encryption

Sanjay Bhattacherjee and Palash Sarkar

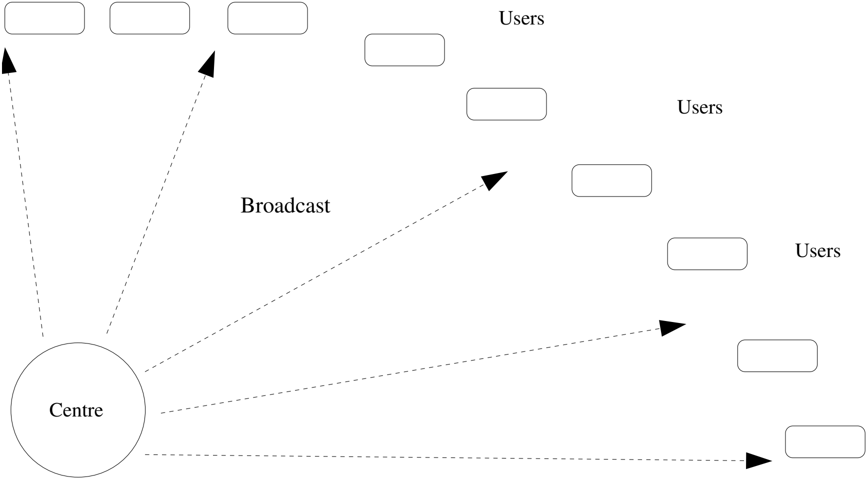
Indian Statistical Institute, Kolkata

Elliptic Curve Cryptography
(This is not)
2014

Conventional Symmetric Key Encryption



Symmetric Key Broadcast Encryption



Symmetric Key BE Functionality

- The centre pre-distributes secret information to the users.
- A broadcast takes place in a session.
- For each session:
  - Some users are privileged and the rest are revoked.
  - The actual message is encrypted once using a session key.
  - The session key undergoes a number of separate encryptions. This determines the header.
  - Only the privileged users are able to decrypt. A coalition of all the revoked users get no information about the message.

Parameters of Interest

- Size of the header.
- Size of the secret information required to be stored by the users.
- Time required by the centre to encrypt.
- Time required by a user to decrypt.

Hdr sz and enc time are proportional to # enc of the session key.

Requirement: Reduce header size, user storage and decryption time.

Applications of BE

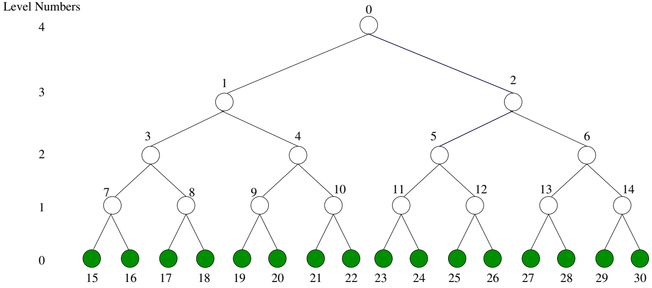
- **AACS** standard: content protection in optical discs: *Disney, Intel, Microsoft, Panasonic, Warner Bros., IBM, Toshiba and Sony.*
- **Pay-TV**: BSkyB in UK and Ireland has a subscriber base of over 10 million;
- Cable Television Networks (Regulation) Amendment Act, 2011 (India).
- File Sharing in Encrypted File Systems.
- Encrypted Email to Mailing Lists.
- Military Broadcasts: Global Broadcast Service (US), Joint Broadcast System (Europe).
- ...

Subset Cover Schemes

- Identify a collection $\mathcal{S}$ consisting of subsets of users.
- Assign keys to each subset in $\mathcal{S}$.
- To each user, assign secret information such that it is able to generate secret keys for each subset in $\mathcal{S}$ to which it belongs; and no more.
- During a broadcast, form a partition $\{S_1, \ldots, S_h\}$ of the set of privileged users with $S_i \in \mathcal{S}$.
- The session key is encrypted using the keys for $S_1, \ldots, S_h$.
- Each privileged user can decrypt; no coalition of revoked users gains any information about the session key (or the message).

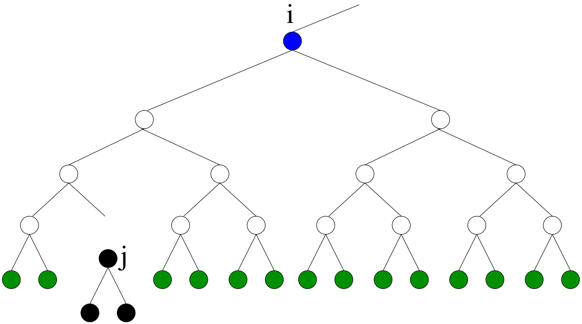
Subset Difference Scheme

Naor-Naor-Lotspiech (2001): patented, AACS standard.
Assumes an underlying full binary tree



Subsets in the collection $\mathcal{S}$

$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$: has all users that are in $\mathcal{T}_i$ but not in $\mathcal{T}_j$



Collection $\mathcal{S}$: has all subsets $S_{i,j}$ such that $j(\neq i)$ is in the subtree $\mathcal{T}_i$.

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed) || G_M(seed) || G_R(seed)$

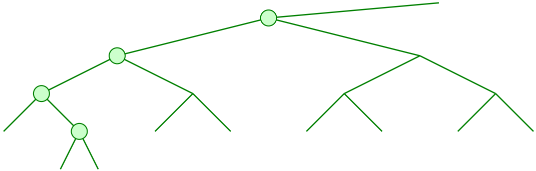


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$

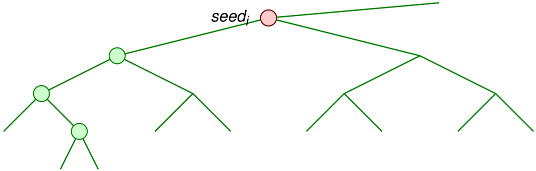


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$

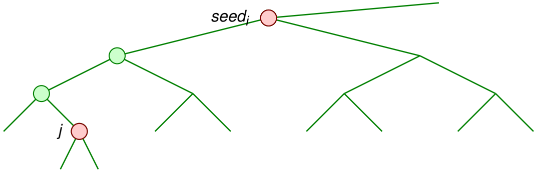


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$
$G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$

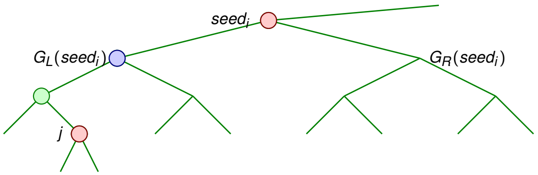


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$

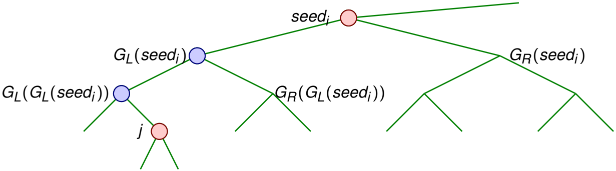


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$

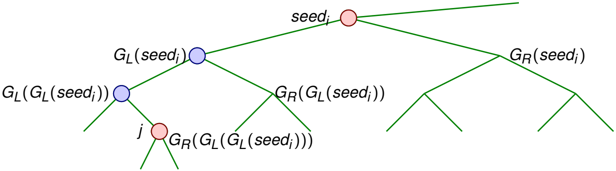


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Key Assignment

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$

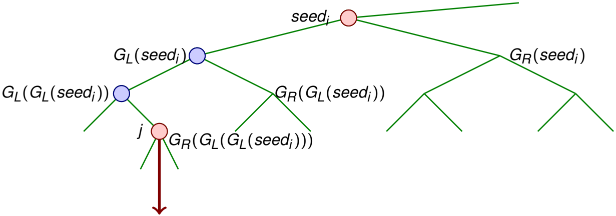


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Pseudo-random generator (PRG): $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$

$G(seed) = G_L(seed) || G_M(seed) || G_R(seed)$

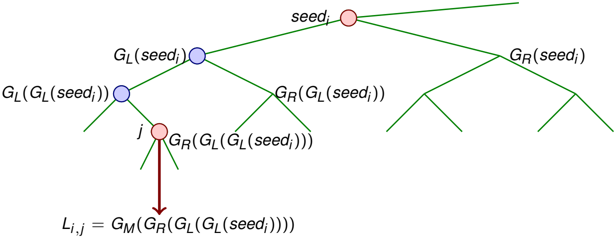


Figure : Key of $S_{i,j}$: $L_{i,j} = G_M(G_R(G_L(G_L(seed_i))))$

Assigning seeds to users

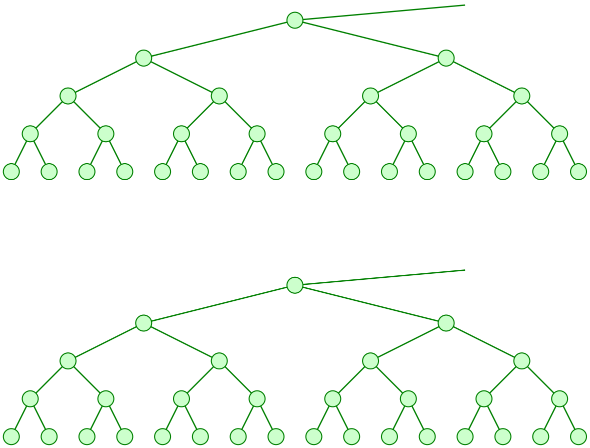
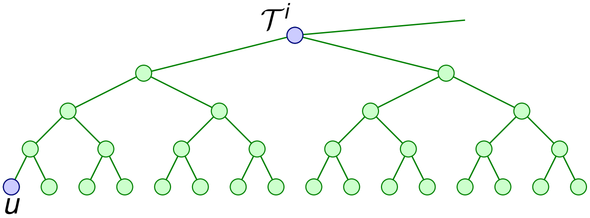


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users



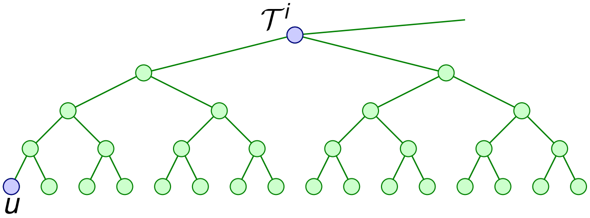


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

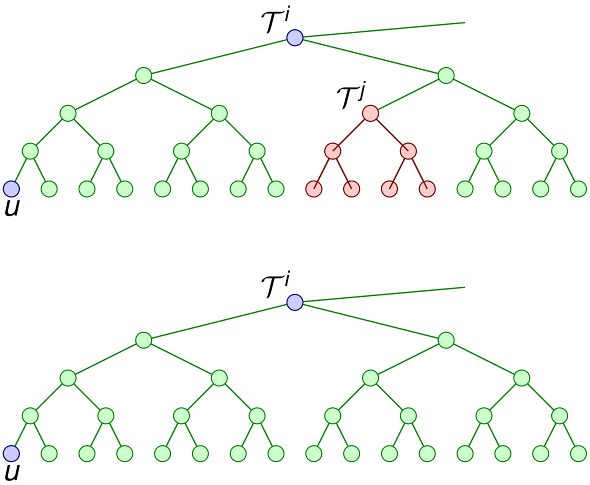


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

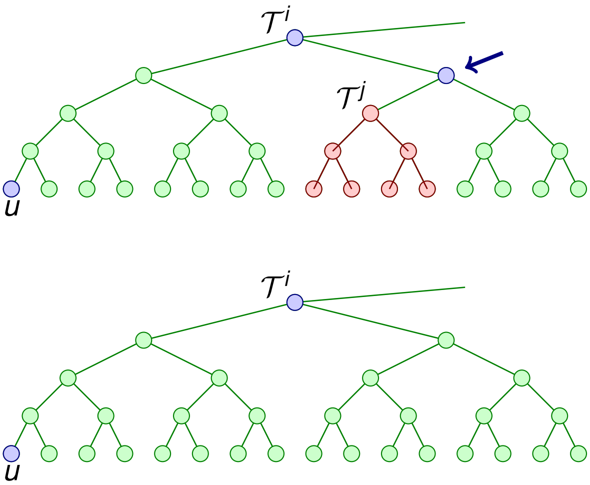


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

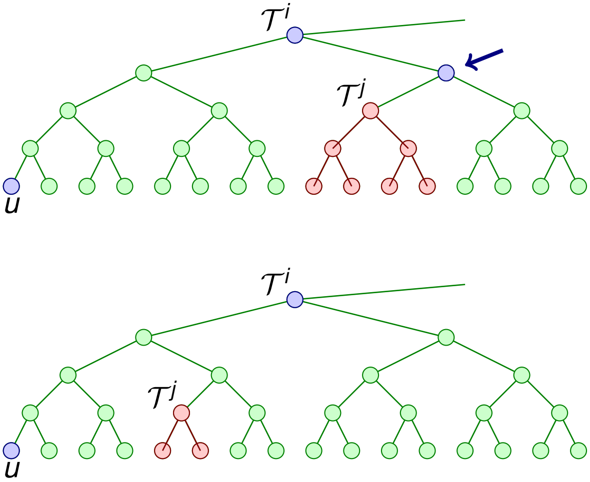


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

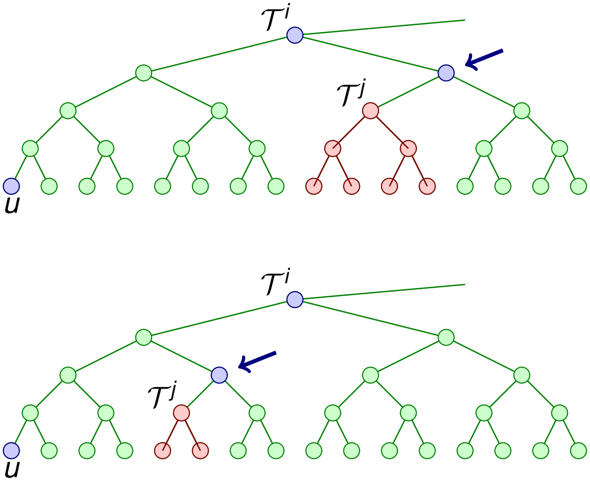


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

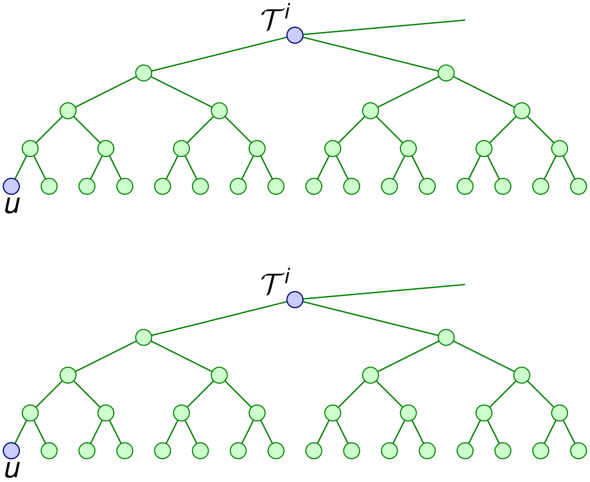


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

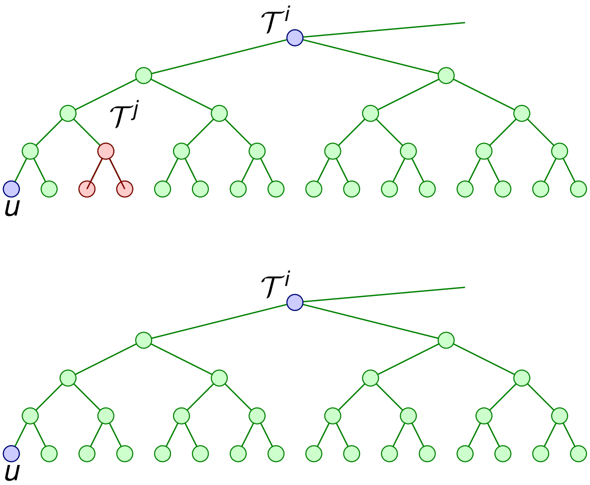


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

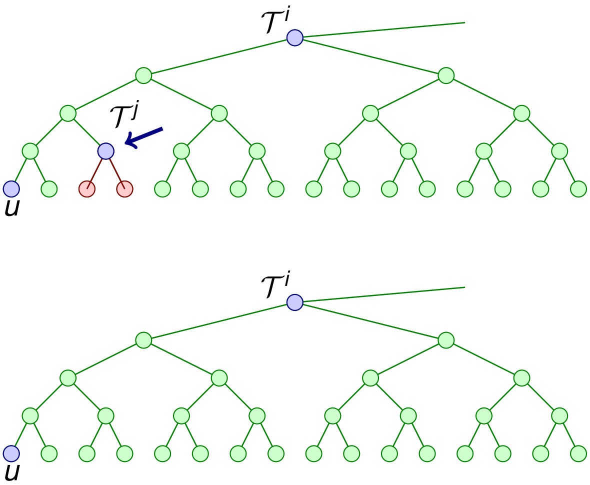


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

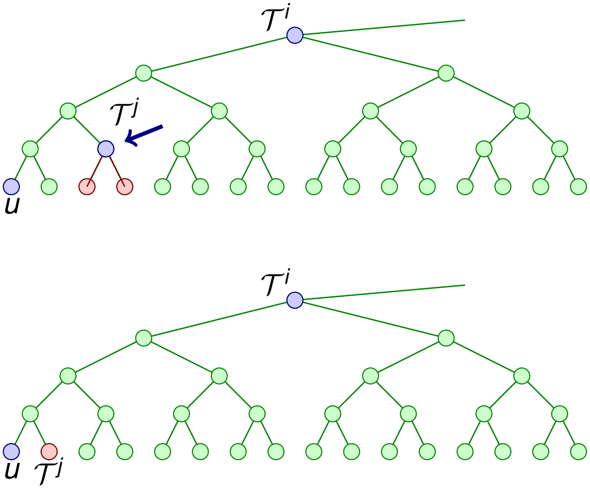


Figure : From one derived seed, keys of many subsets can be generated

Assigning seeds to users

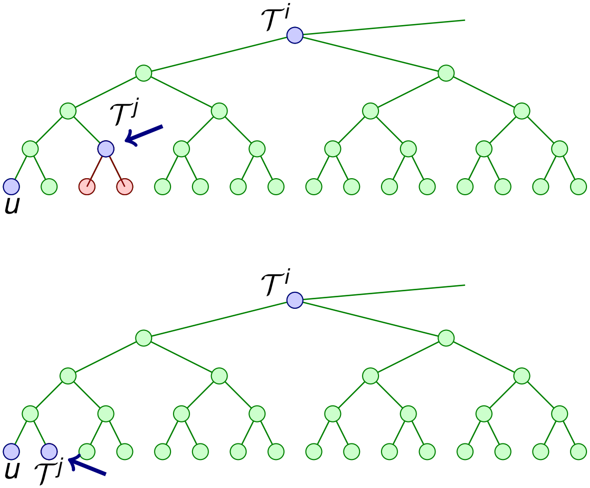


Figure : From one derived seed, keys of many subsets can be generated

User Storage

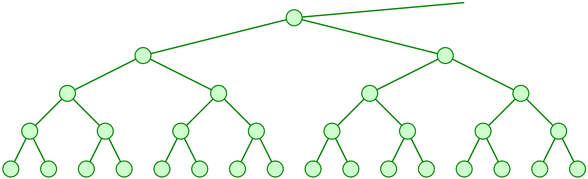


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes “falling-off” from the path between i and u, derived from $seed_i$.

User Storage

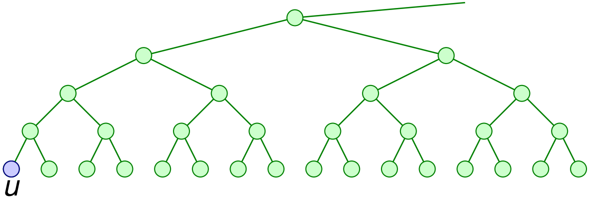


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes "falling-off" from the path between i and u, derived from $seed_i$.

User Storage

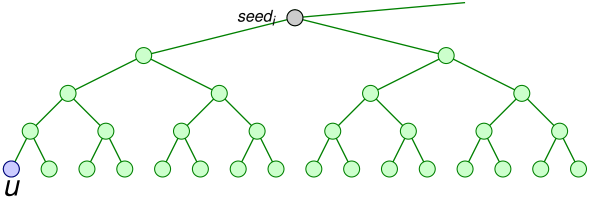


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes “falling-off” from the path between i and u, derived from $seed_i$.

User Storage

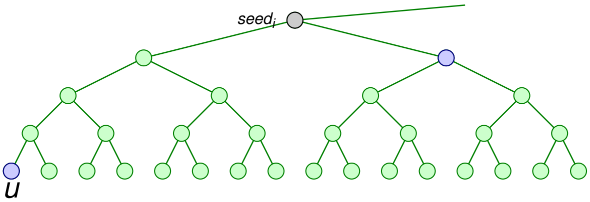


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes “falling-off” from the path between i and u, derived from $seed_i$.

User Storage

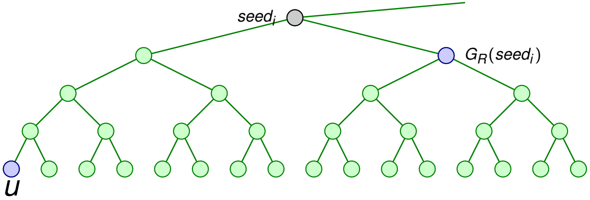


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes “falling-off” from the path between i and u, derived from $seed_i$.

User Storage

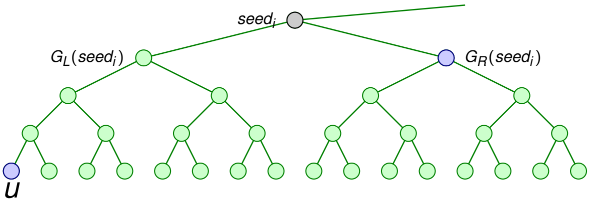


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes "falling-off" from the path between i and u, derived from $seed_i$.

User Storage

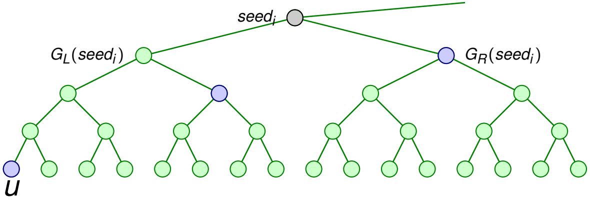


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes "falling-off" from the path between i and u, derived from $seed_i$.

User Storage

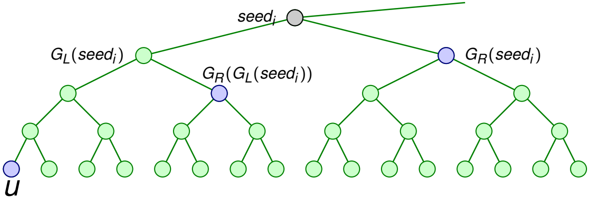


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes "falling-off" from the path between i and u, derived from $seed_i$.

User Storage

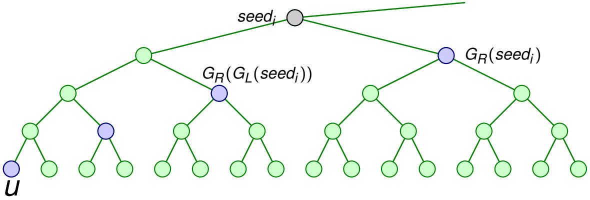


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes "falling-off" from the path between i and u, derived from $seed_i$.

User Storage

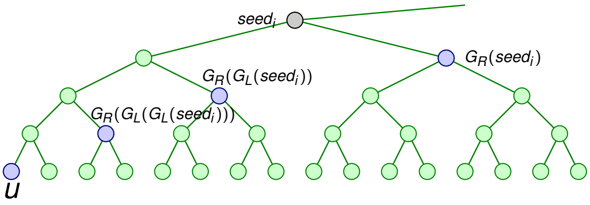


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes “falling-off” from the path between i and u, derived from $seed_i$.

User Storage

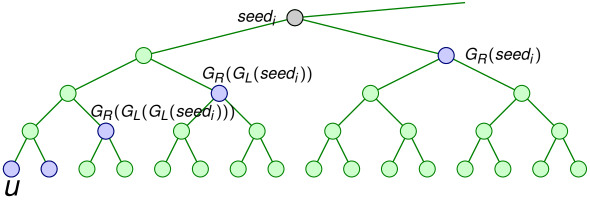


Figure : Secrets stored by u

User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes "falling-off" from the path between i and u, derived from $seed_i$.

User Storage

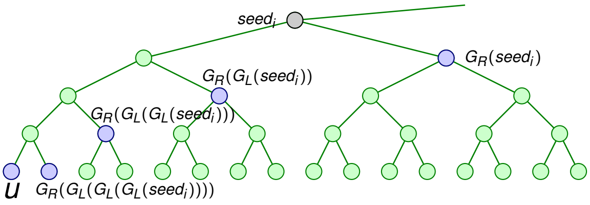
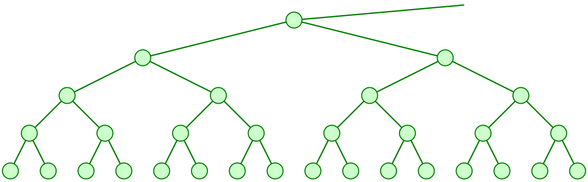


Figure : Secrets stored by u

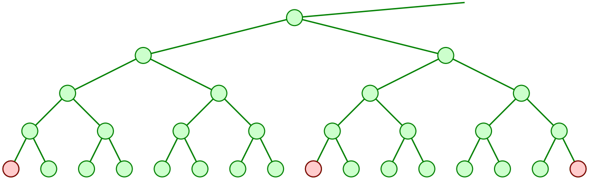
User u stores: for every $\mathcal{T}_i$ to which it belongs, the derived labels of nodes “falling-off” from the path between i and u, derived from $seed_i$.

Subset Cover Finding Algorithm



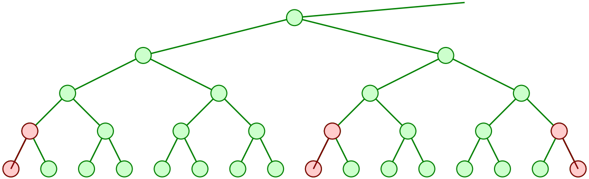
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



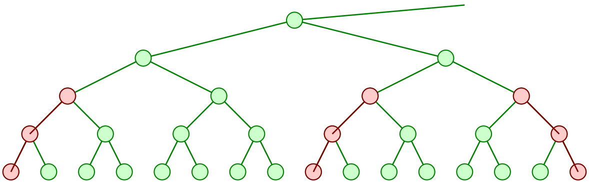
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



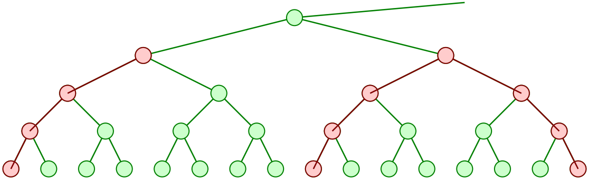
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



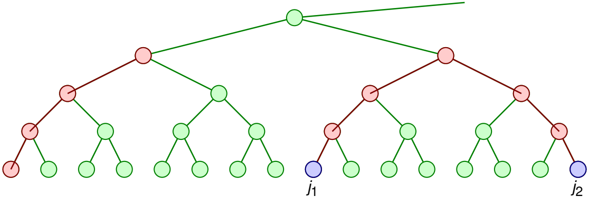
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



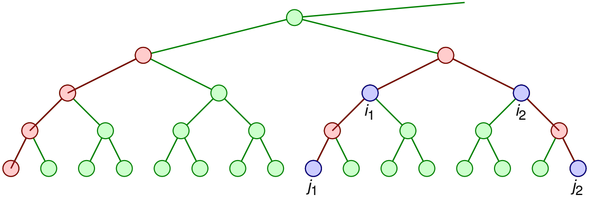
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



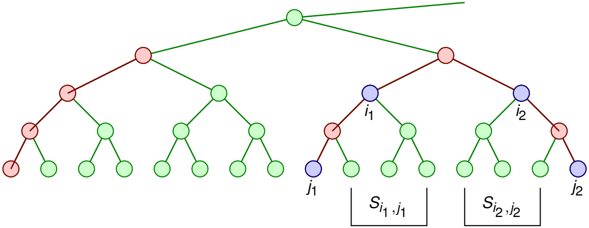
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



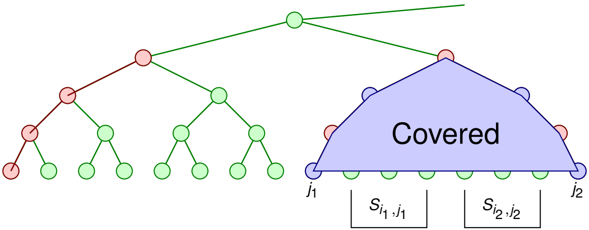
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



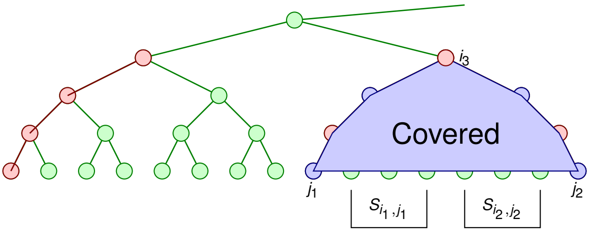
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



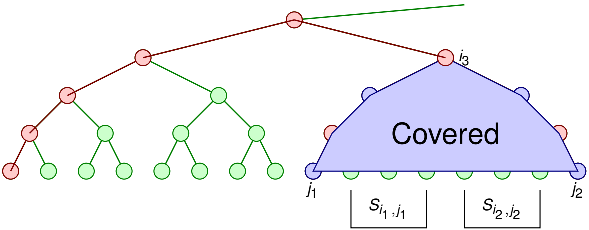
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



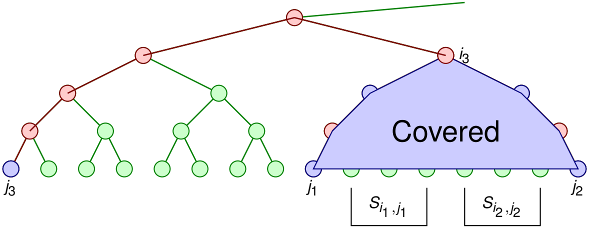
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



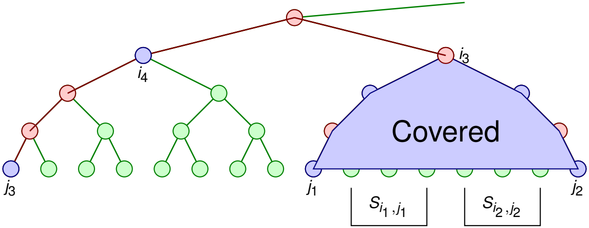
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



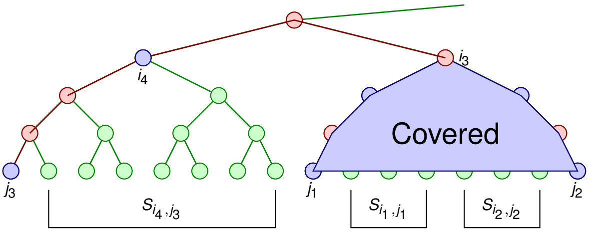
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



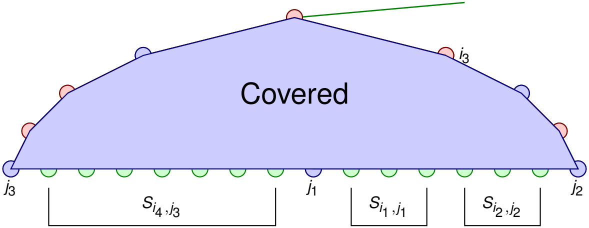
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



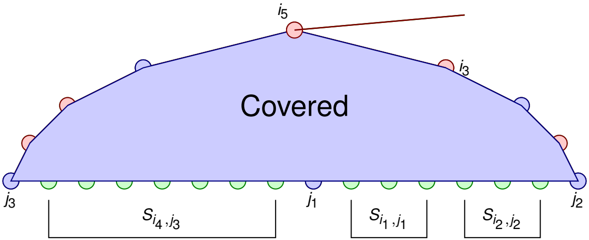
$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

Subset Cover Finding Algorithm



$$S_{i,j} = \mathcal{T}_i \setminus \mathcal{T}_j$$

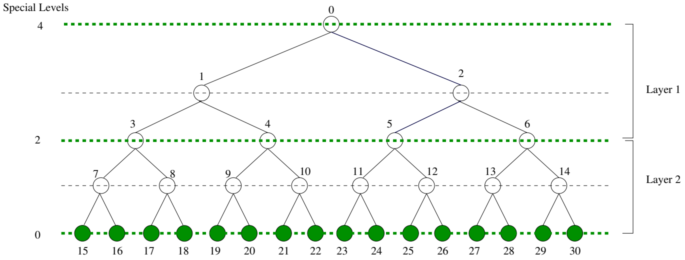
NNL-SD Parameters

For n users out of which r are revoked:

- User storage needed: $O(\log^2(n))$.
- Header length in the worst case: $2r - 1$.
- Decryption time in the worst case: $O(\log n)$.

Layered Subset Difference Scheme

Halevy-Shamir (CRYPTO, 2002) Some levels are marked as *"special"*.



Layered SD Scheme

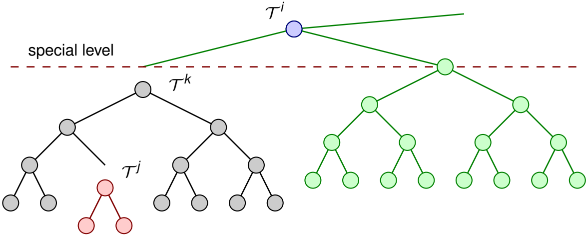


Figure : The subset $S_{i,j}$ split into $S_{i,k}$ (green leaves) and $S_{k,j}$ (grey leaves).

Layered SD Scheme

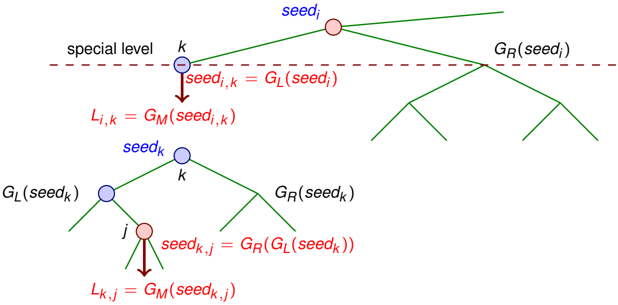


Figure : Key for $S_{i,k}$ is $L_{i,k} = G_M(G_L(seed_i))$ and for $S_{k,j}$ is $L_{k,j} = G_M(G_R(G_L(seed_k)))$.

Important Parameters

NNL-SD scheme:

- User storage needed: $O(\log^2(n))$.
- Maximum Header Length: $2r - 1$.

HS-LSD scheme:

- User Storage needed: $O(\log^{3/2} n)$.
- Maximum header length: $4r - 2$.

Some Questions

- What is the expected header length of the NNL scheme?
- The NNL and the HS schemes are based on *full* binary trees; What happens if the number of users is not a power of two?
- Is the user storage achieved in the HS scheme the minimum possible?
- Is the (expected) header length achieved in the NNL scheme the minimum possible?
- What happens if we use trees of arity higher than 2?

Tackling Arbitrary Number of Users

Complete Tree SD Scheme

Question: What happens when the number of users is not a power of two?

Answer: Add dummy users to get to the next power of two.

- If the dummy users are considered revoked, then the effect on the header length is disastrous.
- If the dummy users are privileged, the situation is better but, there is still a measureable effect on the header length.

Solution: Use a complete binary tree.

- "Completes" (and also subsumes) the NNL-SD scheme to work for any number of users.
- Conceptually simple; working out the details is a bit involved.

CTSD Scheme: Header Length Analysis

$N(n, r, h)$: number of revocation patterns with n users, out of which r users are revoked and the header length is h.

Recurrence relation for $N(n, r, h)$.

- $N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{j \in \mathrm{IN}(i)} T(\lambda_j, r_1, h_1 - 1)$ where $\mathrm{IN}(i)$ is the set of all internal nodes in the subtree $\mathcal{T}^i$ excluding the node i.
- $T(\lambda_i, r_1, h_1) = \sum_{r'=1}^{r_1-1} \sum_{h'=0}^{h_1} N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$ where λ_{2i+1} (respectively λ_{2i+2}) is the number of leaves in the left (respectively right) subtree of $\mathcal{T}^i$.

Boundary Conditions

$T(\lambda_i, r_1, h_1)$	$r_1 < 0$	$r_1 = 0$	$r_1 = 1$	$2 \leq r_1 < n$	$r_1 = n$	$r_1 > n$
$h_1 = 0$	0	0	0	0	1	0
$h_1 \geq 1$	0	0	0	from rec.	0	0
$N(\lambda_i, r_1, h_1)$	$r_1 < 0$	$r_1 = 0$	$r_1 = 1$	$2 \leq r_1 < n$	$r_1 = n$	$r_1 > n$
$h_1 = 0$	0	0	0	0	1	0
$h_1 = 1$	0	1	n	from rec.	0	0
$h_1 > 1$	0	0	0	from rec.	0	0

Table : Boundary conditions on $T(n, r, h)$ and $N(n, r, h)$.

Computing $N(n, r, h)$

Dynamic Programming:

- $N(n, r, h)$ can be computed in $O(r^2h^2 \log n + rh \log^2 n)$ time and $O(rh \log n)$ space.
- $N(n, r, h)$ for all possible h can be computed in $O(r^4 \log n + r^2 \log n)$ time and $O(r^2 \log^2 n)$ space.
- $N(n, r, h)$ for all possible r and h can be computed in $O(n^4 \log n + n^2 \log^2 n)$ time and $O(n^2 \log n)$ space.
- $N(i, r, h)$ for $2 \leq i \leq n$ and all possible r and h can be computed in $O(n^5 + n^3 \log n)$ time and $O(n^3)$ space.

Previous to our work, the only known method was to enumerate all possible $\binom{n}{r}$ revocation patterns, run the header generation algorithm and count the number of patterns leading to a header of size h.

CTSD: Maximum Header Length

Theorem: The maximum header length in the CTSD method for n users is $\min(2r - 1, \lfloor \frac{n}{2} \rfloor, n - r)$.

- For the NNL-SD scheme, the bound of $2r - 1$ was known.
- Complete picture: if $r \leq n/4$, the bound $2r - 1$ is appropriate; if $n/4 < r \leq n/2$, the bound $n/2$ is appropriate; and for $r > n/2$, the bound $n - r$ is appropriate.
- Using the CTSD method is never worse than individual transmission to privileged users.
- The proof requires extensive use of the recurrence for $N(n, r, h)$.

n_r: The value of n for which the header length of $2r - 1$ is achieved with r revoked users.

- A complete characterisation of n_r is obtained.

CTSD: Expected Header Length

Random experiment: Select a random subset of r users out of n users and revoke them.
Random variable $X^i_{n,r}$: takes the value 1 if $S_{i,j}$ is in the header for some j and 0 otherwise.

- $E[X^i_{n,r}] = \Pr[X^i_{n,r} = 1]$.

$H_{n,r}$: expected header length for n users with r revoked users.

- $H_{n,r} = \sum E[X^i_{n,r}] = \sum \Pr[X^i_{n,r} = 1]$ where the sum is over all the $n-1$ internal nodes i in the tree.

CTSD: Expected Header Length

- For all nodes i at the same level, $\Pr[X^i_{n,r} = 1]$ takes at most 3 possible values.
- As a consequence, the sum can be re-written to vary over the levels of the tree.
- $H_{n,r}$ can be computed in $O(r \log n)$ time and $O(1)$ space.
- Provides granular information: expected number of subsets in the header from all the nodes at a certain level.
- Since CTSD subsumes NNL-SD, all the results also hold for NNL-SD.

NNL-SD: Expected Header Length

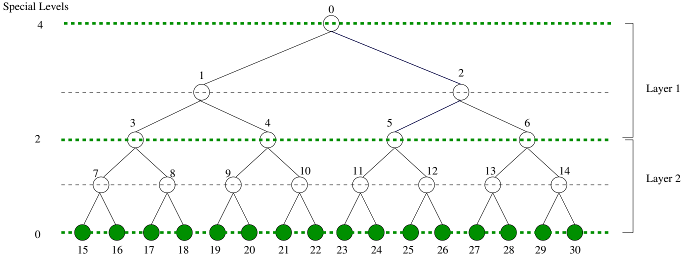
Theorem: For all $n \geq 1$, $r \geq 1$, the expected header length $H_{n,r} \uparrow H_r$, as n increases through powers of two, where

$$H_r = 3r - 2 - 3 \times \sum_{i=1}^{r-1} \left(\left(-\frac{1}{2} \right)^i + \sum_{k=1}^{i} (-1)^k \binom{i}{k} \frac{(2^k - 3^k)}{(2^k - 1)} \right).$$

r	2	3	4	5	6
H_r/r	1.25	1.25	1.2455	1.2446	1.2448

Reducing User Storage Below Halevy-Shamir Scheme

Halevy-Shamir LSD Scheme



"The root is considered to be at a special level, and in addition we consider every level of depth $k \cdot \sqrt{\log(n)}$ for $k = 1 \ldots \log(n)$ as special (wlog, we assume that these numbers are integers)."

Works for 2^{ℓ_0} users with $\ell_0 = 4, 9, 16, 25$ (in the practical range).

Halevy-Shamir LSD Scheme

- For the case of $n = 2^{28}$, HS suggests special levels to be 28, 22, 16, 10, 5, 0.
- Nothing is mentioned about how to choose the layer lengths when ℓ_0 is not a perfect square.

Extending the HS Scheme

Residual bottom layer: Write $\ell_0 = d(e-1) + p$ where $1 \leq p \leq d$. Then the special levels are

$$\ell_0, \ell_0 - d, \ell_0 - 2d, \ldots, \ell - d(e-1), 0.$$

Balanced layering: Write $\ell_0 = d(e-1) + p = (e-d+p)d + (d-p)(d-1)$. Define the layer lengths from the top to be

$$(\underbrace{d, \ldots, d}_{e-d+p}, \underbrace{d-1, \ldots, d-1}_{d-p}).$$

Extending the HS Scheme

- Both strategies (residual bottom; balanced) can be shown to provide the same user storage.
- Having smaller layers nearer the top increases the user storage.
- The balanced layering strategy provides slightly smaller expected header length. We call this the extended-HS (eHS) layering strategy.

Layering Strategy

A choice of special levels is called a layering strategy.

- A layering strategy $\boldsymbol{\ell}$ is denoted by the numbers of the special levels $\ell_0 > \ell_1 > ... > \ell_{e-1} > \ell_e = 0$.
- The layering strategy has $(e + 1)$ special levels.
- Let $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_e)$.
- In general, the layer lengths need not be (almost) equal.

Layering Strategy and User Storage

$$\mathrm{storage}_0(\boldsymbol{\ell}) = \sum_{i=0}^{e-1} \ell_i + \frac{1}{2}\sum_{i=0}^{e-1}(\ell_i - \ell_{i+1})(\ell_i - \ell_{i+1} - 1).$$

Recursive description:

$$\begin{aligned}&\mathrm{storage}_0(\ell_0, \ell_1, \ldots, \ell_e)\\ &= \ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} + \mathrm{storage}_0(\ell_1, \ldots, \ell_e).\end{aligned}$$

Root as a Non-Special Layer

Observations:

- It can be shown that the probability of the root generating a subset in the header is small.
- Having the root as a special layer increases the user storage.

Layering strategy with root as a non-special layer:

$$\text{storage}_1(\boldsymbol{\ell}) = \text{storage}_0(\boldsymbol{\ell}) - \ell_1.$$

Reduces user storage by ℓ_1 at a negligible increase in the expected header size.

Storage Minimal Layering

- Given ℓ_0, let $\mathsf{SML}_0(\ell_0)$ be a layering strategy which minimises the user storage among all layering strategies;
- $\#\mathsf{SML}_0(\ell_0)$: user storage required by $\mathsf{SML}_0(\ell_0)$;
- $\mathsf{SML}_1(\ell_0)$ and $\#\mathsf{SML}_1(\ell_0)$ corresponds to the case where the root is not special.

Relations/Recurrences for SML

$$\#\mathrm{SML}_0(\ell_0) = \min_{1 \le e \le \ell_0} \#\mathrm{SML}_0(e, \ell_0);$$

where $\#\mathrm{SML}_0(e, \ell_0)$ is the minimum storage that can be achieved with e special levels.

$$\#\mathrm{SML}_0(e, \ell_0) = \min_{(\ell_0, \ldots, \ell_e)} \mathrm{storage}_0(\ell_0, \ell_1, \ldots, \ell_e)$$

where the minimum is over all possible layering strategies $(\ell_0, \ell_1, \ldots, \ell_e)$.

Relations/Recurrences for SML

$$\begin{aligned}
&\#\mathrm{SML}_0(e, \ell_0) \\
&= \min_{1 \le \ell_1 < \ell_0} \left(\ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} + \#\mathrm{SML}_0(e-1, \ell_1) \right);
\end{aligned}$$

$$\begin{aligned}
&\#\mathrm{SML}_1(\ell_0) \\
&= \min_{e} \min_{\ell_1} \left(\#\mathrm{SML}_0(e-1, \ell_1) + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} \right).
\end{aligned}$$

Computing SML

Dynamic Programming:

- An $O(\ell^3)$ time and $O(\ell^2)$ space algorithm to compute $\#\mathrm{SML}_0(\ell_0)$
- The actual layering strategy $\mathrm{SML}_0(\ell_0)$ can also be recovered from the algorithm.
- Once the table has been computed using dynamic programming, it is possible to obtain $\#\mathrm{SML}_1(\ell_0)$ and $\mathrm{SML}_1(\ell_0)$.

Properties of SML

- SML_0 and SML_1 are not necessarily unique; choose the layering for which expected header length is lower.
- Removing ℓ_0 from SML_0 does not necessarily provide SML_1.
- Compared to NNL-SD, eHS reduces storage by a large amount; SML_0 reduces storage below eHS by a small amount; SML_1 reduces storage below eHS by 18% to 24% in the practical range.

Examples of SML

Suppose there are 2^{28} users, i.e., $\ell_0 = 28$:

- NNL-SD: layering: 28,0; storage: 406.
- eHS: layering: 28,22,16,10,5,0; storage: 146.
- SML_0: layering: 28,21,15,10,6,3,1,0; storage: 140.
- SML_1: layering: 22,16,11,7,4,2,0; storage: 119.

Complete Tree LSD Scheme

Question: What if the number of users n is not a power of 2?

Answer: Use a complete tree as in the case of the NNL-SD scheme.

- The notions of layering strategy and storage minimal layering carry over to this case.
- All users would not be required to store the same amount; the requirement is to minimise the maximum of all the user storages.

Header Length

Maximum Header Length:

- At most $\min\left(4r - 2, \left\lceil \frac{n}{2} \right\rceil, n - r\right)$.
- At most $\min\left(4r - 3, \left\lceil \frac{n}{2} \right\rceil, n - r\right)$ if the root level is special.

Expected Header Length:

- The splitting of subsets complicates the analysis.
- An $O(r \log^2 n)$ time algorithm to compute the expected header length.
- A very useful tool to analyse various schemes.

Constrained Minimisation

Question: Is it possible to obtain expected header length close to that of NNL-SD, but, with lower user storage?

- For each level, we have an expression for the expected number of subsets arising from the nodes at that level.
- Suppose ℓ is a level which maximises the above quantity.

Question: How to choose ℓ?
Answer: How to do this analytically is not clear. Extensive experimentation has shown that $\ell = \ell_0 - \log_2 r$ is a good choice.

Constrained Minimisation Layering

Fix a value of r and set $\ell = \ell_0 - \log_2 r$.

- Level ℓ is made special, so that subsets arising from level ℓ are not split.
- All levels below ℓ are made non-special.
- At most one level above ℓ (mid-way between ℓ and the root) is made special; all other levels are made non-special.

How to Choose r?

- Depending on the application, make an *assumption* on the minimum value of r, say r_{min}.
- If the actual r is greater than r_{min}, then there is no problem.
- If the acutal r is smaller than r_{min}, then the benefits on the header length is not attained.
- Choosing r_{min} to be too small will not lead to substantial savings in user storage; choosing r_{min} to be too large will not provide the desired reduction on header storage.

A CML Example

Number of users is $n = 2^{\ell_0}$ with $\ell_0 = 28$ and suppose $r_{\min} = 2^{10}$.

- NNL-SD: layering: 28,0; storage: 406.
- eHS: layering: 28,22,16,10,5,0; storage: 146;
  header lengths:
  $(1.69, 1.63, 1.64, 1.67, 1.69, 1.72, 1.73, 1.74, 1.75, 1.75)$.
- CML: layering: 23, 18,0; storage: 219;
  header lengths:
  $(1.14, 1.08, 1.04, 1.03, 1.01, 1.01, 1.00, 1.00, 1.00, 1.00)$.

Header lengths for 10 equispaced values of r from 2^{10} to 2^{14} normalised by the header length of the NNL-SD scheme.

References

The NNL and the HS papers:

Dalit Naor, Moni Naor, and Jeffery Lotspiech.
Revocation and tracing schemes for stateless receivers.
In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.

Dani Halevy and Adi Shamir.
The LSD broadcast encryption scheme.
In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2002.

Our Works

Sanjay Bhattacherjee and Palash Sarkar.
Complete tree subset difference broadcast encryption scheme and its analysis.
Des. Codes Cryptography, 66(1-3):335–362, 2013.

Sanjay Bhattacherjee and Palash Sarkar.
Concrete analysis and trade-offs for the (complete tree) layered subset difference broadcast encryption scheme.
IEEE Transactions on Computers, 63(7): 1709–1722, 2014.

Sanjay Bhattacherjee and Palash Sarkar.
Tree based symmetric key broadcast encryption.
Cryptology ePrint Archive, Report 2013/786, 2013.
`http://eprint.iacr.org/2013/786`.

Sanjay Bhattacherjee and Palash Sarkar.
Reducing communication overhead of the subset difference scheme.
Cryptology ePrint Archive, Report 2014/577, 2014.
`http://eprint.iacr.org/2014/577`.

Sanjay Bhattacherjee.
Implementations related to the above papers, `https://drive.google.com/folderview?id=0B7azs7qqqdS0UnB5aHp3WmJwcDQ&usp=sharing_eil`.
Uploaded on 13th August, 2014.

Thank you for your attention!