Open access • Book Chapter • DOI:10.1007/978-3-642-32344-7_2

Recoverable encryption through a noised secret over a large cloud

— Source link

Sushil Jajodia, Witold Litwin, Thomas Schwarz Sj

Institutions: George Mason University, Paris Dauphine University, Universidad Católica del Uruguay Dámaso Antonio Larrañaga

Published on: 01 Jan 2013 - International Conference on Data Management in Grid and P2P Systems

Topics: Key escrow, Key distribution, Encryption, Escrow and Backup

Recoverable Encryption through Noised Secret over Large Cloud

Sushil Jajodia[1], W. Litwin[2] & Th. Schwarz[3]

[1]George Mason University, Fairfax, VA {jajodia@gmu.edu}
[2]Université Paris Dauphine, Lamsade {witold.litwin@dauphine.fr}
[3]Thomas Schwarz, UCU, Montevideo {tschwarz@ucu.edu.uy}

What ?

- New schemes for backup of encryption keys entrusted to an Escrow
  - Collectively called RE_{NS} Schemes
  - They backup high quality encryption keys
    - AES (256b), DH 500+b…
- Backup itself is specifically encrypted
- Unlike a traditional simple key copy

What ?

- Fast brute-force recovery remains possible
  - In the absence of key owner
  - Within the timing wished by the recovery requestor
- But only over a large cloud

  1K – 100K nodes

What ?

- Unwelcome recovery is unlikely
  - E.g. could easily take, say, 70 or even 700 days at escrow's processor alone
  - Illegal use of a large cloud is implausible
    - Cloud providers do best to prevent it
    - Easily noticeable if ever starts
      - Follow the money
    - Leaves compromising traces in numerous logs

Why

- High quality key loss danger is Achilles' heel of modern crypto
  - Makes many folks refraining of *any* encryption
  - Other loose many tears if unthinkable happens

Why

- If you create key copies…
  - Every copy increases danger of disclosure
  - For an Escrow, her/his copy is an obvious temptation
  - Some Escrows may not resist to
- In short users face the dilemma:

Key loss or disclosure ? That is The Question

-

Why

- RE_{NS} schemes alleviate this dilemma
- Easily available large clouds make them realistic
- Our schemes should benefit numerous applications

How (Overview) : Key Owner Side

- *Key owner* or *client* chooses inhibitive timing of 1-node (brute-force) recovery
  - Presumably unwelcome at escrow's site alone
  - E.g. 70 days
  - Or 700 days for less trusted escrows
  - Or anything between

How : Key Owner Side

- Consequently , the owner fixes a large integer
  - Called *backup encryption complexity or hardness*
- Actually, this step may be programmed
  - The backup encryption *agent* on client node may be in charge of

How : Key Owner Side

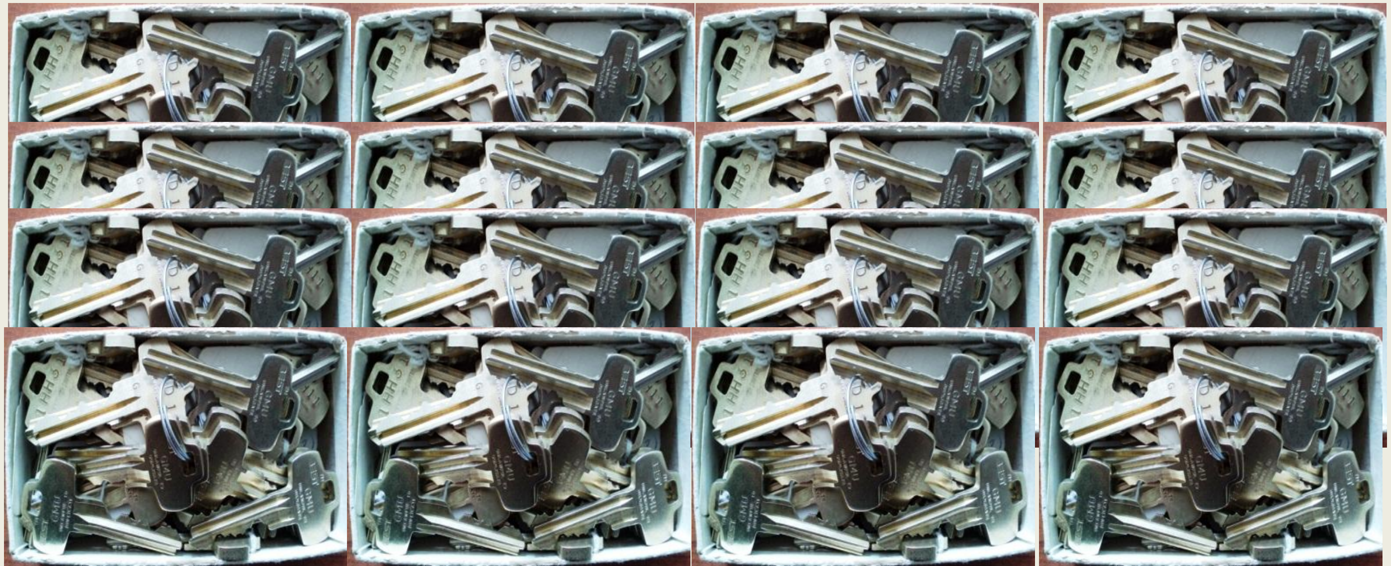
- Key owner or the agent creates the shared *noised* secret
  - Some share(s) of the actual secret become *noised* shares
  - « Burried » among very many look-alike but fake *noise* shares

How : Key Owner Side

- The only way to recognize whether a noise share is a noised one is to try out its « footprint »
- The owner/agent creates the footprint for each noised share
- Each footprint is unique
- *Remember Cinderella ?*

How : Key Owner Side

- Key owner/agent sends the noised secret to Escrow
- Noised secret is the backup
  - *Guess your key by its print in this mess (inspired by CSIS actual ex.)*



How (Overview) : Escrow Side

- *Key requestor* asks Escrow to recover data in acceptable max recovery time
  - E.g. 10 min
- Escrow's server sends the time and all **but one** shares of the noised secret to the cloud
- Intruder to the cloud cannot find the key

How : Escrow's Side

- RE_{NS} scheme executed at the cloud chooses the cloud size
  - To fit the calculus time limit <u>for sure</u>
  - Say 10K nodes
- Search for the noised share gets partitioned over the nodes
- Nodes work in parallel
  - Matching the "footprints"

How : Escrow’s Side

- Every lucky node reports back to Escrow the noised share found
- Escrow’ server recovers the key from all the shares
  - Using the clasical XORing
- Sends the recovered key to Requestor
  - Not forgetting the bill

What Else ?

- Well, everything is in details
  - Client Side Encryption
  - Server Side Recovery
    - Static Scheme
    - Scalable Scheme
  - Related Work
  - Conclusion

What Else ?

- More :
  - Res. Rep.

    http://www.lamsade.dauphine.fr/~litwin/Recoverable%20Encryption_10.pdf
  - S. Jajodia, W. Litwin & Th. Schwarz. Recoverable Encryption through a Noised Secret over a Large Cloud.
    - 5th Inl. Conf. on Data Management in Cloud, Grid and P2P Systems (Globe 2012)
    - Publ. Springer Verlag, Lecture Notes in Comp.

Client Side (Backup) Encryption

- Client *X* backs up encryption key *S*
- *X* estimates 1-node inhibitive time *D*
  - Say 70 days
- *D* measures trust to Escrow
  - Lesser trust ?
    - Choose 700 days

Client Side Encryption

- D determines minimal cloud size N for future recovery in any acceptable time R
  - Chosen by recovery requestor
    - E.g. 10 min
  - X expects $N > D / R$ but also $N \cong D / R$
    - E.g. $N \cong 10K$ for $D = 70$ days
      - $N \cong 100K$ for $D = 700$ days

Client Side Encryption

- *X* creates a classical shared secret for *S*
  - *S* is seen as a large integer,
    - E.g., 256b long for AES
  - Basically, *X* creates a 2-share secret
  - Share s_0 is a random integer
  - Share s_1 is calculated as $s_1 = s_0$ XOR *S*
- Common knowledge:
  - $S = s_0$ XOR s_1

Client Side Encryption

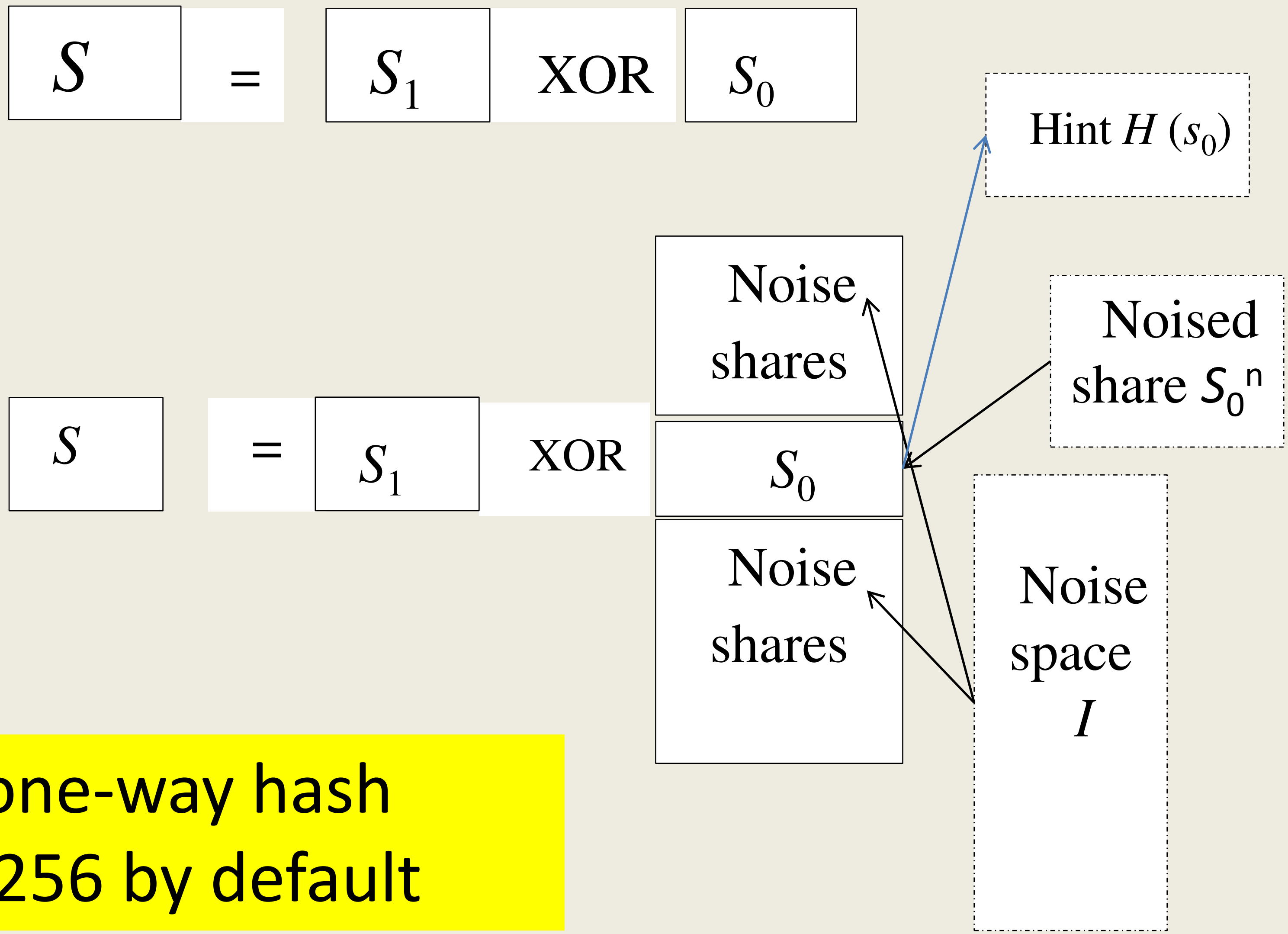
- *X* transforms the shared secret into a *noised* one
  - *X* makes s_0 a *noised* share :
    - Chooses a 1-way hash *H*
      - E.g. SHA 256
    - Computes the *hint* $h = H(s_0)$
  - Chooses the *noise* space
    $I = 0,1…,m,…M-1$
  - For some large *M* determined as we explain soon

Client-Side Encryption

- Each noise *m* and s_0 define a *noise* share *s*
  - In a way we show soon as well
- There are *M* different pseudo random noise shares
  - All but one are different from s_0
  - But it is not known which one is s_0
- The only way to find for any *s* whether $s = s_0$ is to attempt the *match*

$$H(s) ?= h$$

Shared Secret / Noised (Shared) Secret



H is one-way hash
SHA 256 by default

Client Side Encryption

- *X* estimates the 1-node *throughput T*
  - \# of match attempts *H* (*s*) ?= *h* per time unit
    - 1 Sec by default
- *X* sets *M* to *M* = Int (*DT*).
  - *M* should be $2^{40} \div 2^{50}$ in practice

Client Side Encryption

- *X* randomly chooses $m \in I = [0,1 \ldots M[$
- Calculates *base noise* share $f = s_0 - m$
- Defines *noised share* $s_0^n = (f, M, h)$.
- Sends the *noised secret* $S' = (s_0^n, s_1)$ to Escrow as the backup

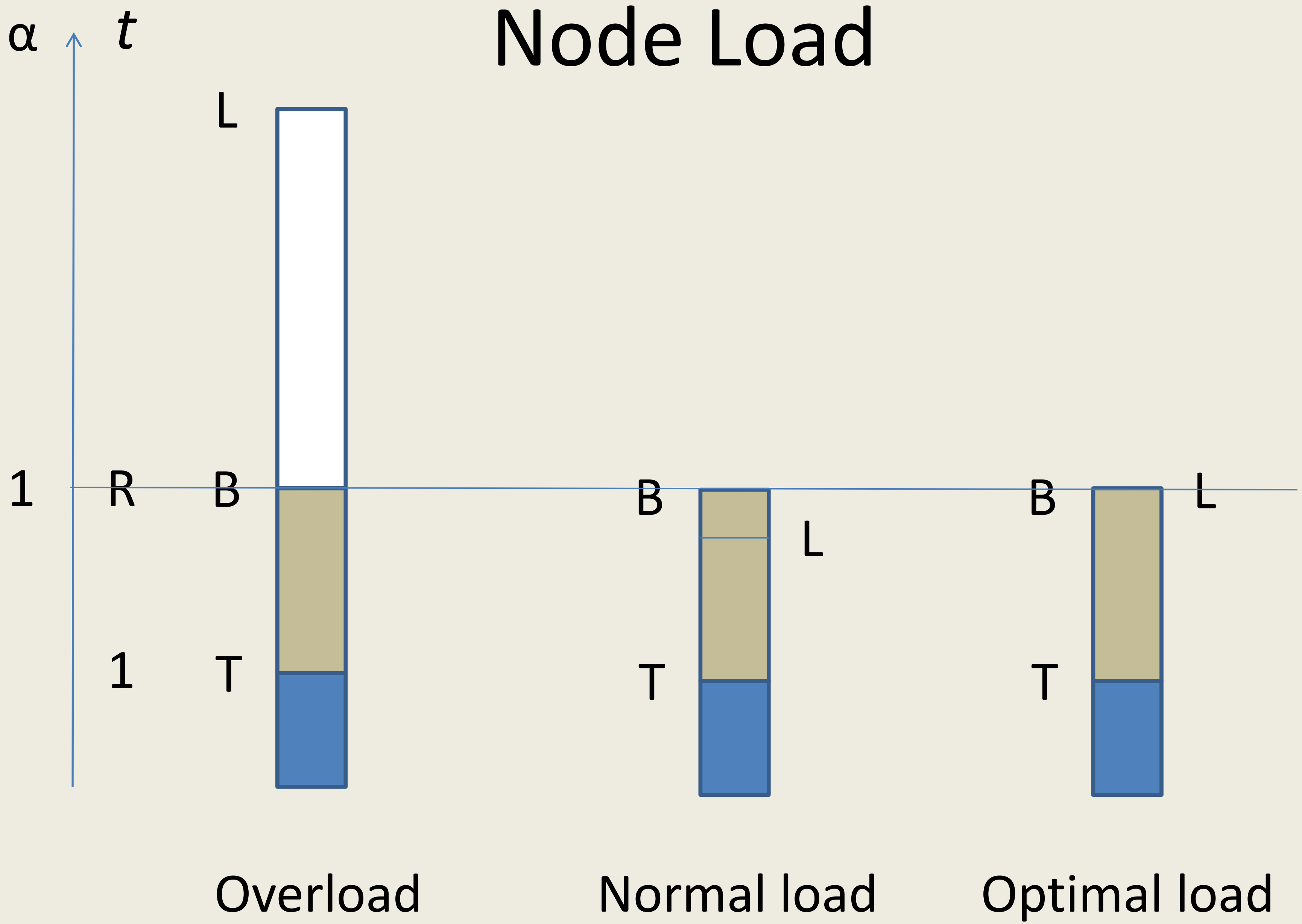
Escrow-Side Recovery (Backup Decryption)

- Escrow *E* receives legitimate request of *S* recovery in time *R* at most
- *E* chooses between *static* or *scalable* recovery schemes
- *E* sends data *S''* = (s_0^n, *R*) to some cloud node with request for processing accordingly
  - Keeps s_1 out of the cloud

Recovery Processing Parameters

- Node *load* L_n *:* # of noises among *M* *assigned to* node *n* for match attempts
- *Throughput* T_n : # of match attempts node *n* can process / sec
- *Bucket (node) capacity* B_n : # of match attempts node *n* can process / time *R*
  - $B_n = R\,T_n$
- *Load factor* $\alpha_n = L_n / B_n$

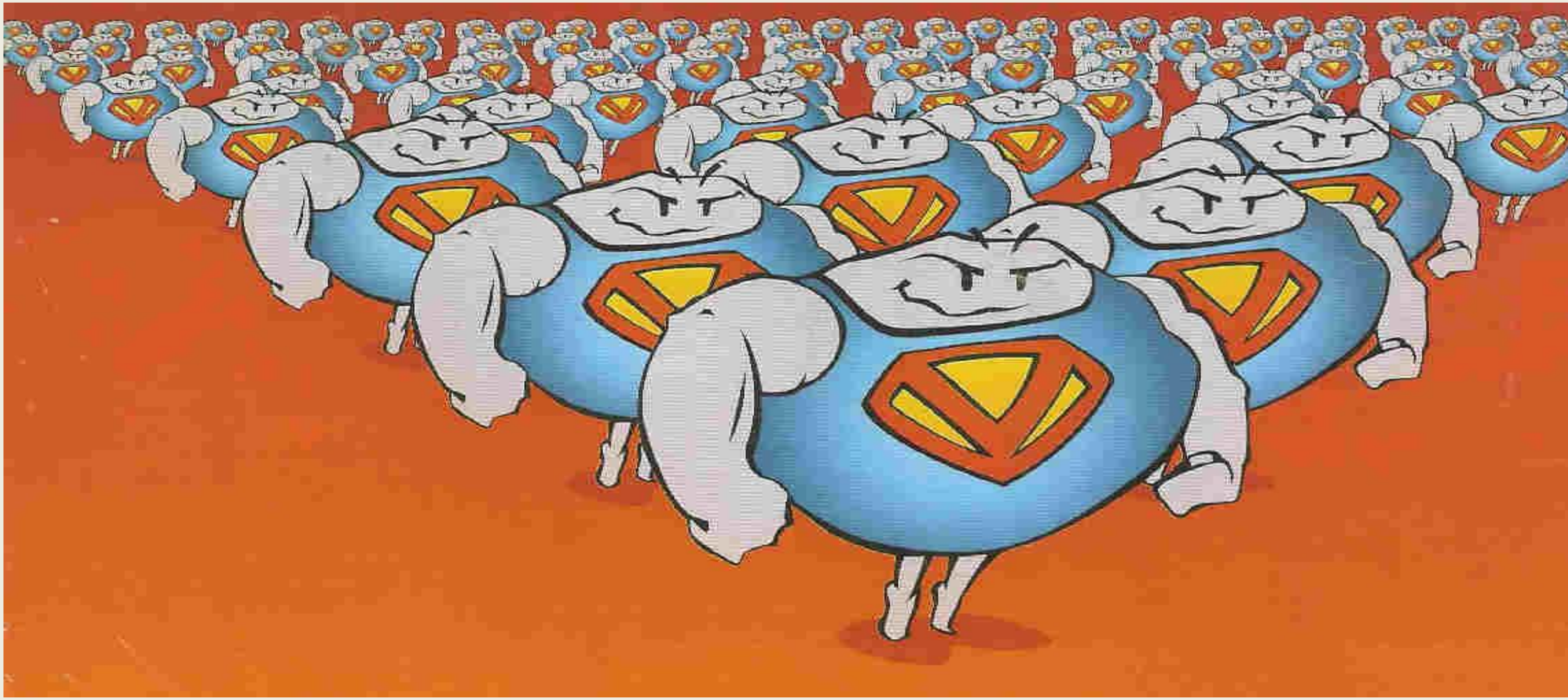
Node Load



Recovery Processing Parameters

- *Notice the data storage oriented vocabulary*
- Node *n* respects *R* iff $\alpha_n \leq 1$
  - Assuming *T* constant during the processing
- **The cloud respects *R* if for every *n* we have $\alpha_n \leq 1$**
- **This is our goal**
  - For both *static* and *scalable* schemes we now present

Static Scheme



- Intended for a homogenous Cloud
  - All nodes provide the same throughput

Static Scheme : *Init* Phase

- Node C that got S'' from E becomes *coordinator*
- Calculates $a(M) = M / B(C)$
  - Usually $\alpha(M) >> 1$
- Defines N as $\lceil a(M) \rceil$
  - Implicitly considers the cloud as *homogenous*
- E.g., $N = 10K$ or $N = 100K$ in our ex.

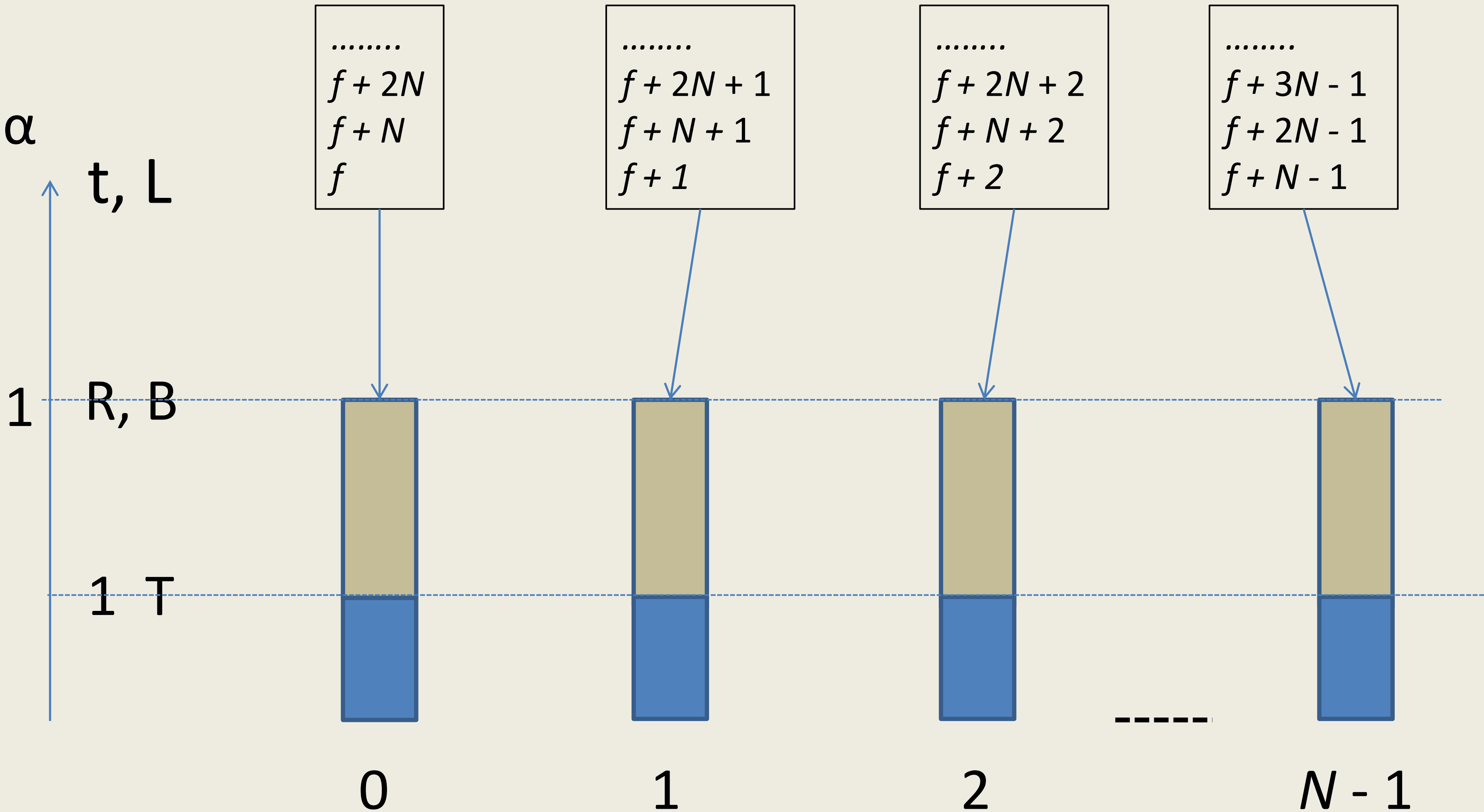
Static Scheme : *Map* Phase

- *C* asks for allocation of *N*-1 nodes
- Associates logical address *n* = 1, 2…*N*-1 with each new node & 0 with itself
- Sends out to every node *n* data (n, a_0, P)
  - a_0 is its own physical address, e.g., IP
  - *P* specifies *Reduce* phase

Static Scheme : *Reduce* Phase

- P requests node n to attempt matches for every noise share $s = (f + m)$ such that $n = m \bmod N$
- In practice, e.g., while $m < M$:
  - Node 0 loops over noise $m = 0, N, 2N$…
    - So over the noise shares $f, f + N, f + 2N$…
  - Node 1 loops over noise $m = 1, N+1, 2N+1$…
  - …..
  - Node $N - 1$ loops over $m =$ (your guess here)

Static Scheme : Node Load



Static Scheme

- Node *n* that gets the successful match sends *s* to *C*
- Otherwise node *n* enters *Termination*
- *C* asks every node to terminate
  - Details depend on actual cloud
- *C* forwards *s* as s_0 to *E*

Static Scheme

- *E* discloses the secret *S* and sends *S* to Requestor
  - Bill included (we guess)
- E.g., up to 400$ on CloudLayer for
  - D = 70 days
  - R = 10 min
  - Both implied N = 10K with private option

Static Scheme

- Observe that $N \geq D / R \; and \; N \cong D / R$
  - If the initial estimate of *T by S* owner holds
- Observe also that for every node *n*, we have $\alpha(n) \leq 1$
- Under our assumptions maximal recovery time is thus indeed *R*
- Average recovery time is $R / 2$
  - Since every noise share is equally likely to be the lucky one

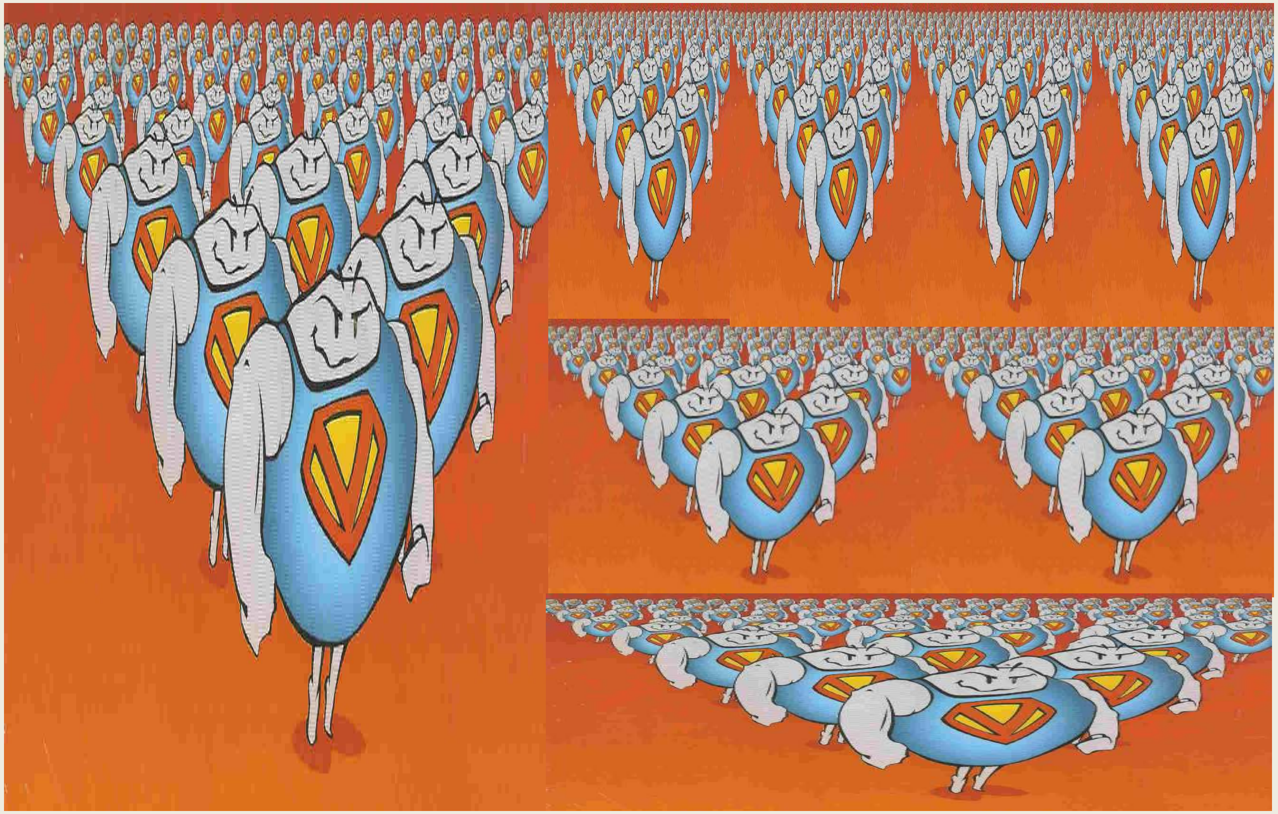
Static Scheme

- See papers for
  - Details,
  - Numerical examples
  - Proof of correctness
    - The scheme really partitions *I*
    - Whatever is *N* and s_0, one and only one node finds s_0

Static Scheme

- Safety
  - No disclosure method can in practice be faster than the scheme
  - Dictionary attack, inverted file of hints…
- Other properties

Scalable Scheme



- Heterogeneous cloud
  - Node throughputs may differ

Scalable Scheme

- Intended for heterogenous clouds
  - Different node throughputs
  - Basically only locally known
- E.g.
  - Private or hybrid cloud
  - Public cloud without so-called *private node* option

Scalable Scheme

- Init phase similar up to α (M) calculus
  - Basically $\alpha(M) >> 1$
  - Also we note it now α_0
- If $\alpha > 1$ we say that node *overflows*
- Node 0 sets then its *level j* to $j = 0$ and *splits*
  - Requests node $2^j = 1$
  - Sets j to $j = 1$
  - Sends to node 1, (S'', j, a_0)

Scalable Scheme

- As result
  - There are $N = 2$ nodes
  - Both have $j = 1$
  - Node 0 and node 1 should each process $M / 2$ match attempts
    - We show precisely how on next slides
  - Iff both α_0 and α_1 are no more than 1
- Usually it should not be the case
- The splitting should continue as follows

Scalable Scheme

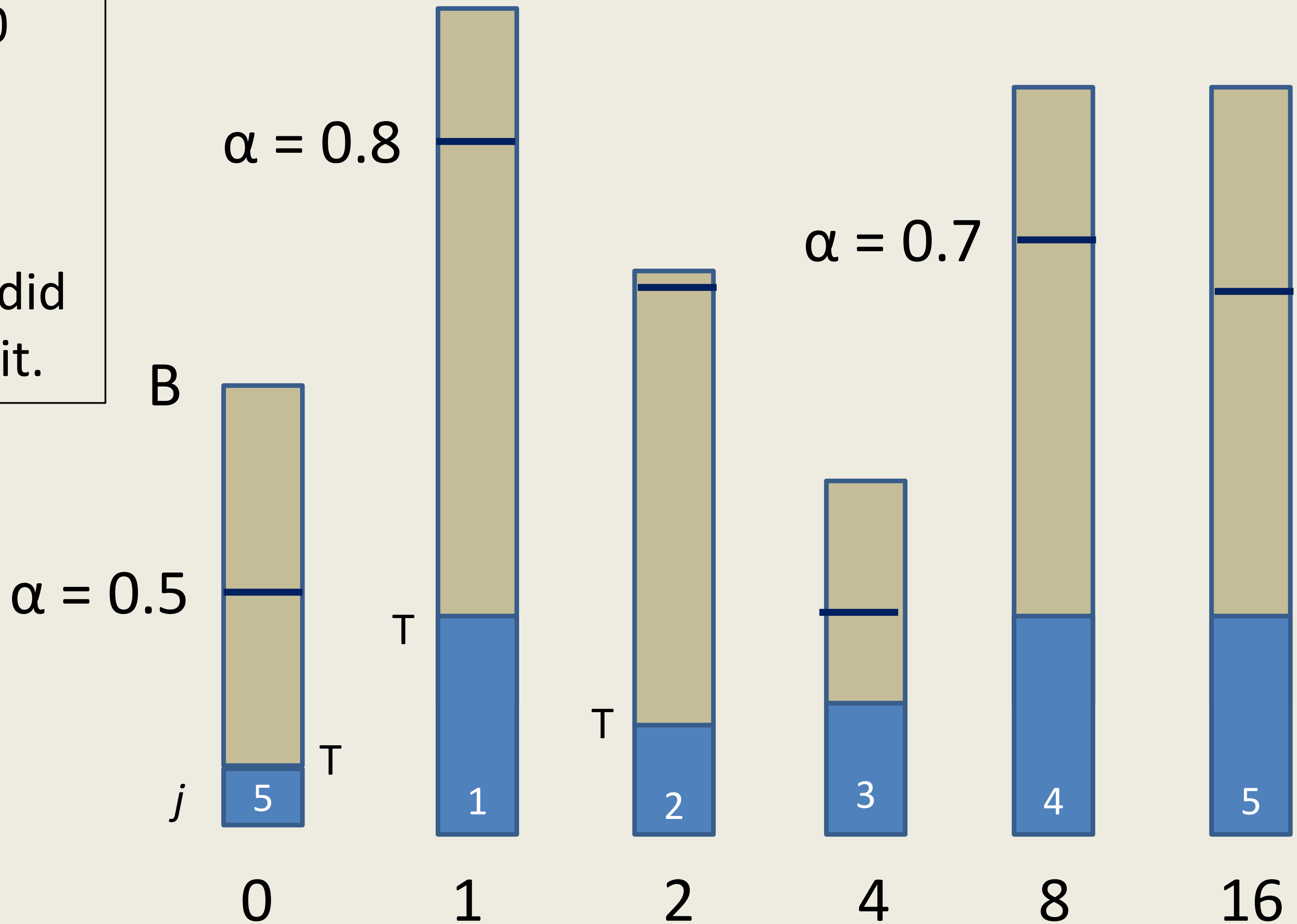
- Recursive rule
  - Each node *n* splits until $\alpha_n \le 1$
  - Each split increases node level j_n to $j_n + 1$
  - Each split creates new node $n' = n + 2^{j_n}$
  - Each node *n'* gets $j_{n'} = j_n$ initially
- Node 0 splits thus perhaps into nodes 1,2,4…
  - Until $\alpha_0 \le 1$
- Node 1 starts with $j = 1$ and splits into nodes 3,5,9…
  - Until $\alpha_1 \le 1$

Scalable Scheme

- Node 2 starts with $j = 2$ and splits into 6,10,18…
  - Until $\alpha_2 \leq 1$
- *Your general rule here*
- Node with smaller T splits more times and vice versa

Scalable Scheme : Splitting

Node 0 split 5 times. Other nodes did not split.



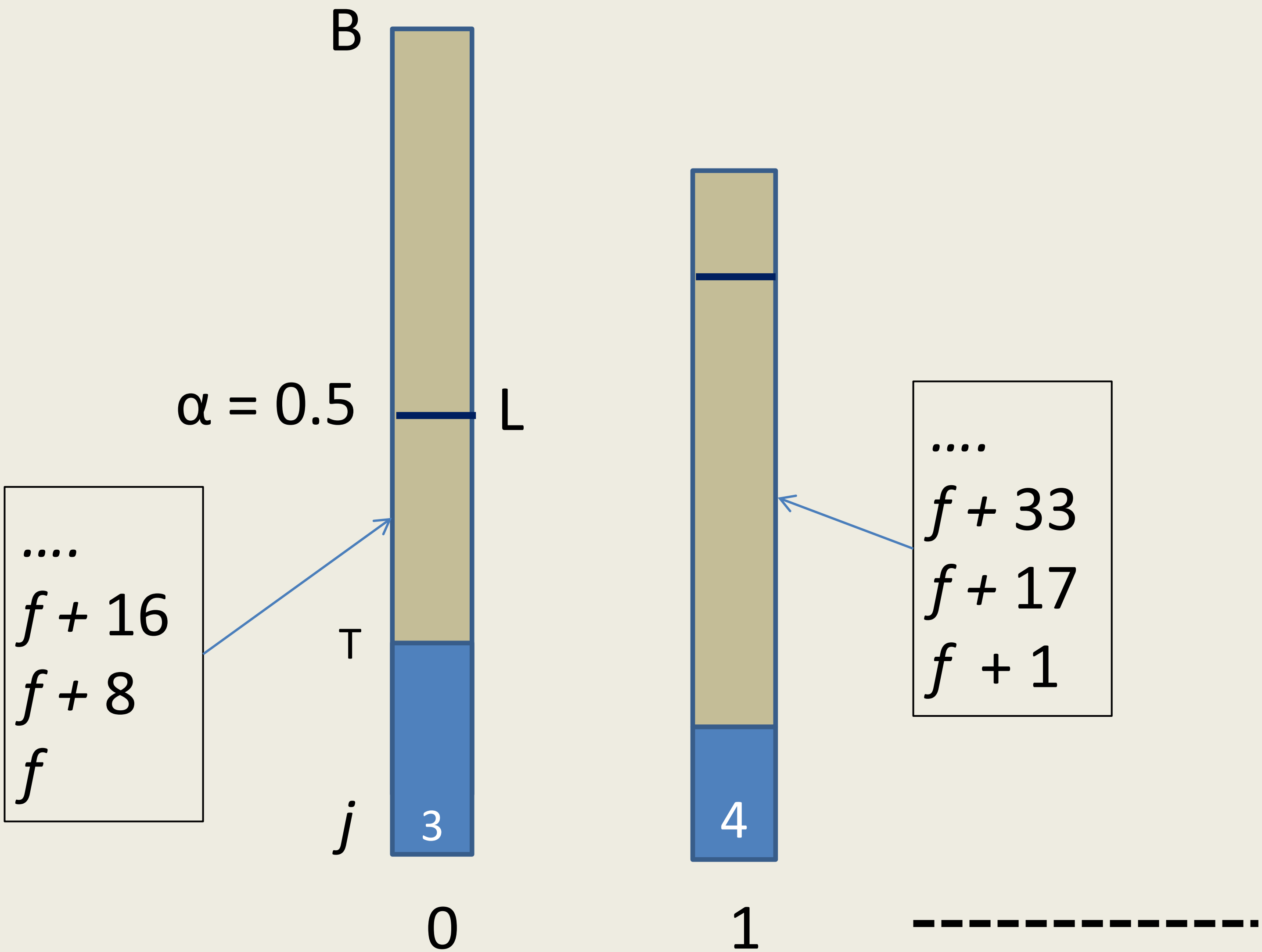
Scalable Scheme

- If cloud is homogenous, the address space is contiguous
- Otherwise, it is not
  - No problem
  - Unlike for a extensible or linear hash data structure

Scalable Scheme : *Reduce* phase

- Every node n attempts matches for every noise $k \in [0, M\text{-}1]$ such that $n = k \bmod 2^{j_n}$.
- If node 0 splits three times, in *Reduce* phase it attempts to match noised shares ($f + k$) with k = 0, 8, 16…
- If node 1 splits four times, it attempts to match noised shares ($f + k$) with k = 1, 17, 33…
- Etc.

Scalable Scheme : Reduce Phase



Scalable Scheme

- $N \geq D / R$
  - If *S* owner initial estimate holds
- For homogeneous cloud it is 30% greater on the average and twice as big at worst / static scheme
- Cloud cost may still be cheaper
  - No need for *private* option
- Versatility may still make it preferable besides

Scalable Scheme

- Max recovery time is up to *R*
  - Depends on homogeneity of the cloud
- Average recovery time is up to *R* /2
- See again the papers for
  - Examples
  - Correctness
  - Safety
  - …
  - Detailed perf. analysis remains future work

Related Work

- RE scheme for outsourced LH* files
- CSCP scheme for outsourced LH* records sharing
- Crypto puzzles
- One way hash with trapdoor
- 30-year old excitement around Clipper chip
- Botnets

Conclusion

- Key safety is Achilles' heel of cryptography
- Key loss or key disclosure ? That is The Question
- RE_{NS} schemes alleviate the dilemma
- Future work Deeper formal analysis
  - Proof of concept implementation
  - Variants

Thanks
for
Your Attention

Witold LITWIN & al