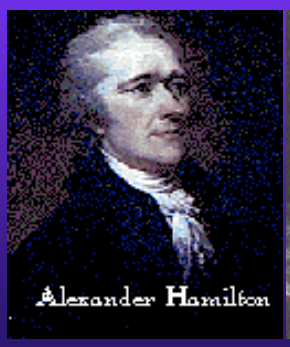
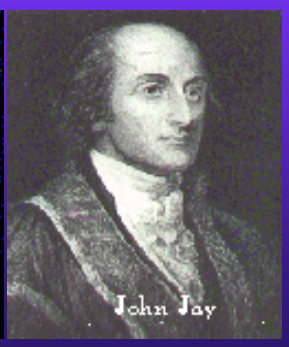
Publius

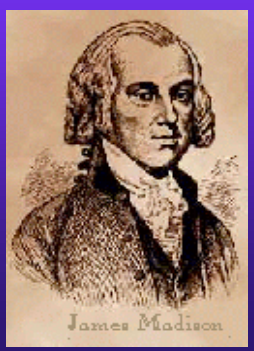
A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System

Publius

- Pen name used by authors of Federalist Papers
- Federalist Papers influential in convincing NY voters to ratify US constitution.








Publius Design Goals
♦ Web publishing system
– Censorship Resistant
– Tamper Evident
– Source Anonymous
– Resistant to DDOS attacks
– Updateable
– Host Content Deniability
– Persistent
– Extensible
– Freely Available


Why Censorship resistance?
♦ Political Dissent
♦ Corporate "Whistleblowing"
♦ Radical Ideas

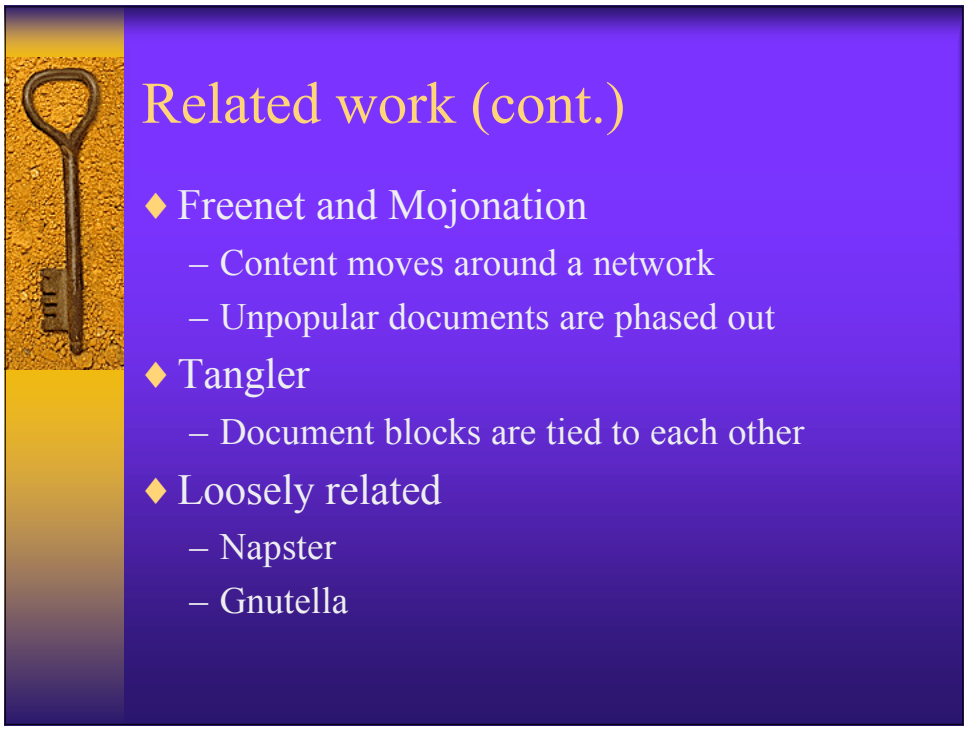
Related work

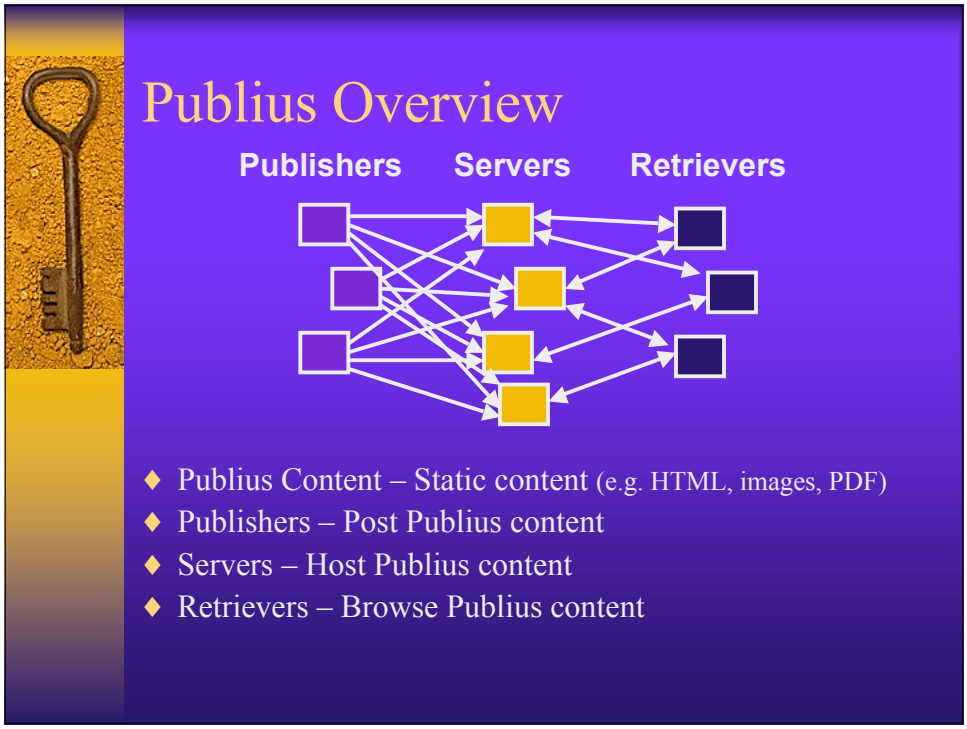
- Anonymous retrieval
  - Anonymizer (www.anonymizer.com)
    - HTTP proxy
    - URL rewriting
  - Proxymate (www.proxymate.com)
    - Formerly LPWA
    - HTTP Proxy
    - Pseudonym generation
  - Freedom (www.freedom.net)
    - Commercial product (Zero Knowledge Systems)
    - Implemented at the network layer
    - Nym creation – allows multiple pseudonyms
    - Supports HTTP, NNTP, POP3, Telnet , etc.

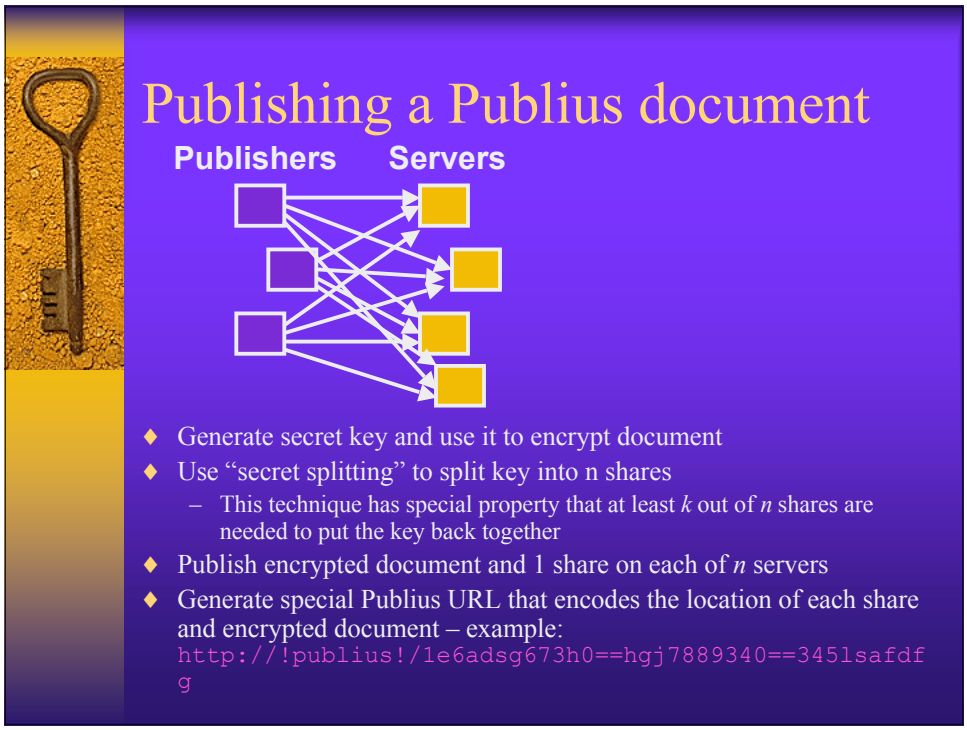
Related work (cont.)

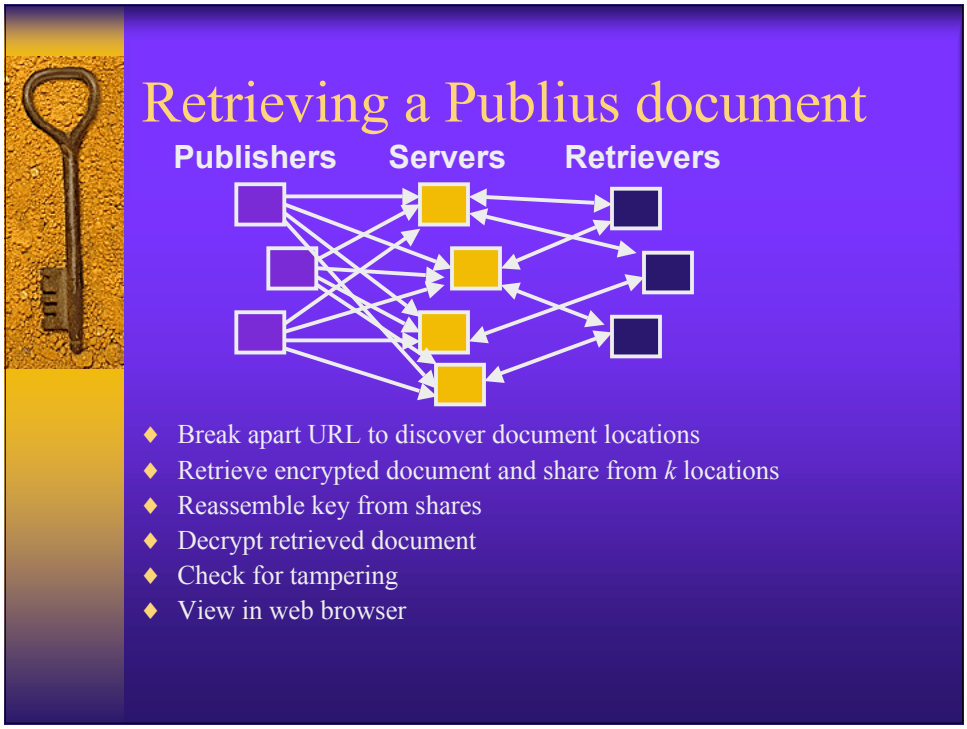
- Onion Routing
  - Mix Network (based on Chaum ’81)
  - HTTP Proxy
  - Experimental system deployed at NRL
- Crowds
  - Dynamic Path generation
  - Crowd members run proxies

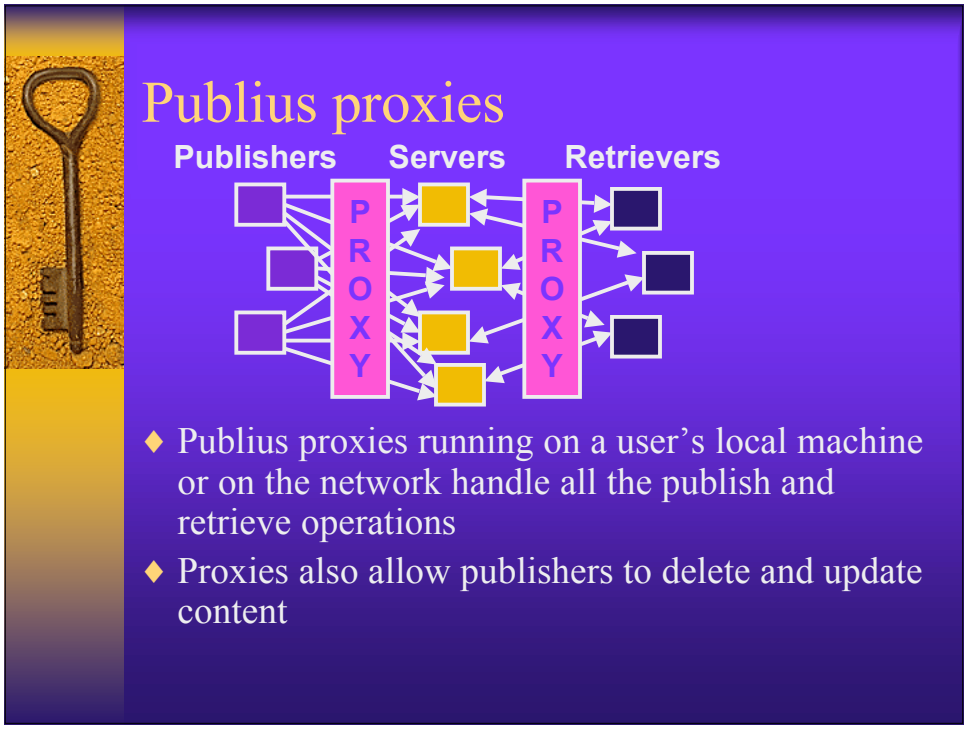

Related work (cont.)
♦ Eternity service (proposed by Anderson)
– Network of many, many servers
– Fee based
– Files cannot be removed or updated
– A USENET implementation (due to Back)
• Scaled Down from original System
• Usenet is storage medium
• Formatting using PGP, SHA1
• Send to alt.anonymous.messages
• Connect via WWW browser
♦ Free Haven
– Mix-based content retrieval
– Developed at the same time as Publius (many similarities)
– No update/delete, but more dynamic server set


Related work (cont.)
♦ Freenet and Mojonation
– Content moves around a network
– Unpopular documents are phased out
♦ Tangler
– Document blocks are tied to each other
♦ Loosely related
– Napster
– Gnutella


Publius Overview
Publishers
Servers
Retrievers
Publius Content – Static content (e.g. HTML, images, PDF)
Publishers – Post Publius content
Servers – Host Publius content
Retrievers – Browse Publius content


Publishing a Publius document
Publishers
Servers
Generate secret key and use it to encrypt document
Use "secret splitting" to split key into n shares
This technique has special property that at least k out of n shares are needed to put the key back together
Publish encrypted document and 1 share on each of n servers
Generate special Publius URL that encodes the location of each share and encrypted document – example:
http://!publius!/1e6adsg673h0==hgj7889340==345lsafdfg


Retrieving a Publius document
Publishers
Servers
Retrievers
Break apart URL to discover document locations
Retrieve encrypted document and share from k locations
Reassemble key from shares
Decrypt retrieved document
Check for tampering
View in web browser


Publius proxies
Publishers
Servers
Retrievers
PROXY
PROXY
Publius proxies running on a user's local machine or on the network handle all the publish and retrieve operations
Proxies also allow publishers to delete and update content

Example


whitehouse.gov


www.redcross.org


att.com


www.nyu.edu

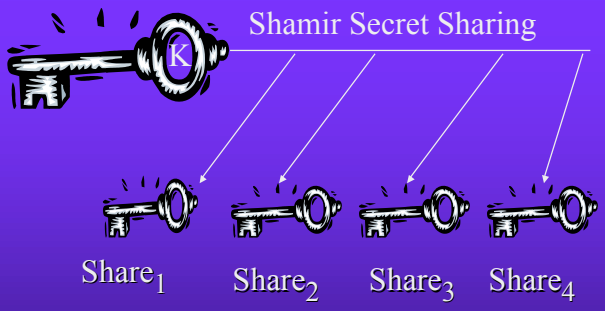

publius.uk

Publius Server Table

www.redcross.org
whitehouse.gov
att.com
www.nyu.edu
publius.uk

Publish Operation

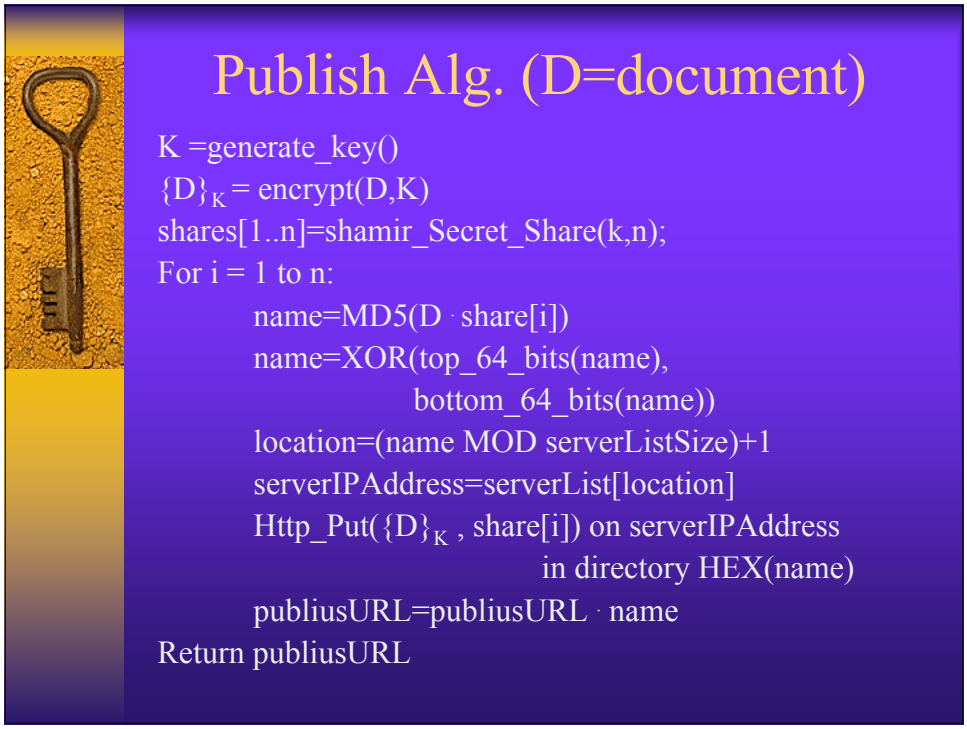
D = Document To Publish K=Key

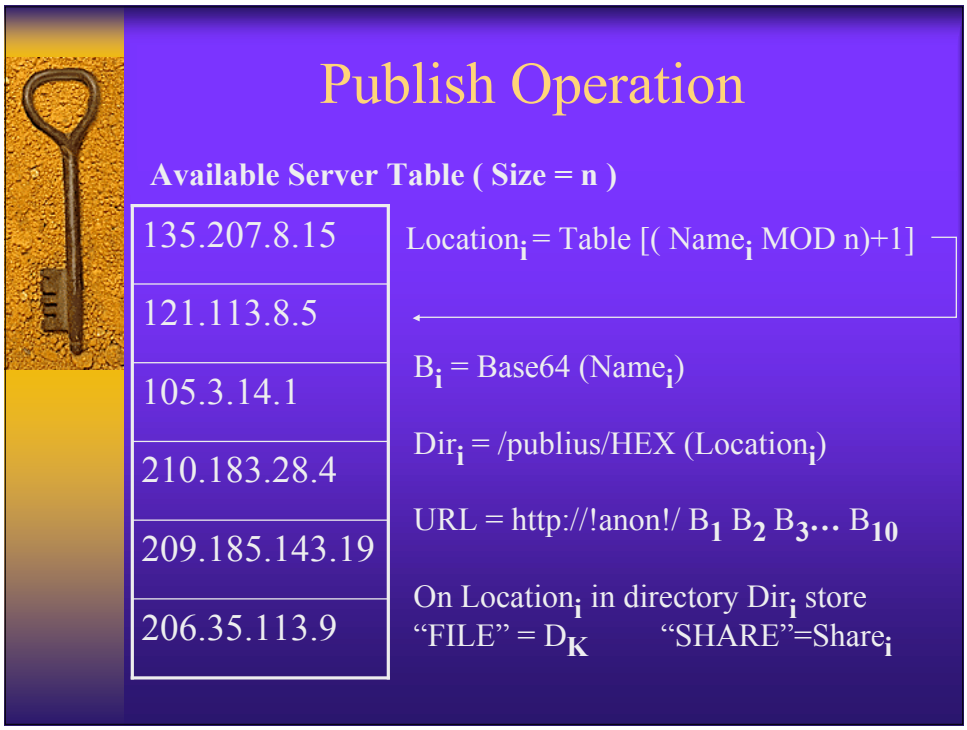


MD5 (D · $Share_i$) Mod 5 = Index Into Server Table

$Index_0$= www.redcross.org $Index_3$ = www.nyu.edu

Store D encrypted under K, and one Share on Server


Publish Alg. (D=document)
K =generate_key()
{D}K = encrypt(D,K)
shares[1..n]=shamir_Secret_Share(k,n);
For i = 1 to n:
name=MD5(D · share[i])
name=XOR(top_64_bits(name),
bottom_64_bits(name))
location=(name MOD serverListSize)+1
serverIPAddress=serverList[location]
Http_Put({D}K , share[i]) on serverIPAddress
in directory HEX(name)
publiusURL=publiusURL · name
Return publiusURL


Publish Operation
Available Server Table (Size = n)
135.207.8.15
121.113.8.5
105.3.14.1
210.183.28.4
209.185.143.19
206.35.113.9
Locationi = Table [(Namei MOD n)+1]
Bi = Base64 (Namei)
Diri = /publius/HEX (Locationi)
URL = http://!anon!/ B1 B2 B3… B10
On Locationi in directory Diri store
“FILE” = DK “SHARE”=Sharei

Retrieve Operation

http://!publius!/MD5(D·Share$_1$)MD5 (D·Share$_2$)…

http://!publius!/unReaDableUrL

Index = MD5(D·Share$_1$) MOD Table_Size

From www.redcross.org Get Encrypted File, Share

Key = combine Shares

D = Decrypt File with Key

Tamper Check = MD5(D·Share$_1$) = value in URL

Retrieval Strategies

Have n shares. Need k shares to form a key

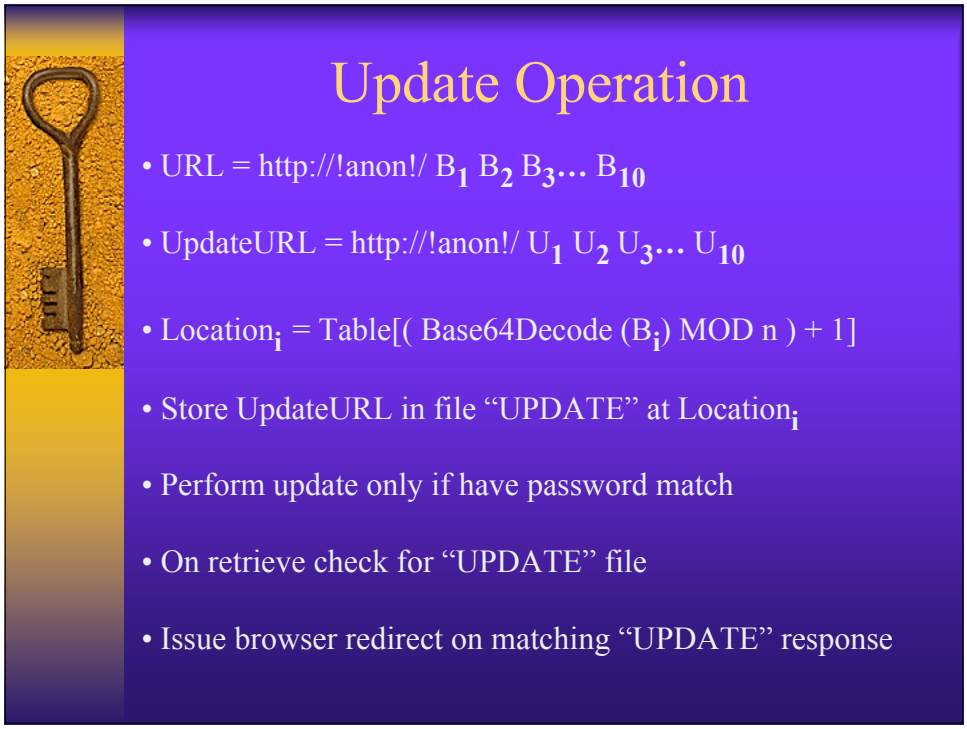
- Current implementation
  - worst case: check (n choose k) combinations
    tamper check one document with all possible keys
- Exhaustive case
  - worst case: Check n * (n choose k) combinations
    tamper check n documents with all possible keys
- Better solution: store hash of D and share

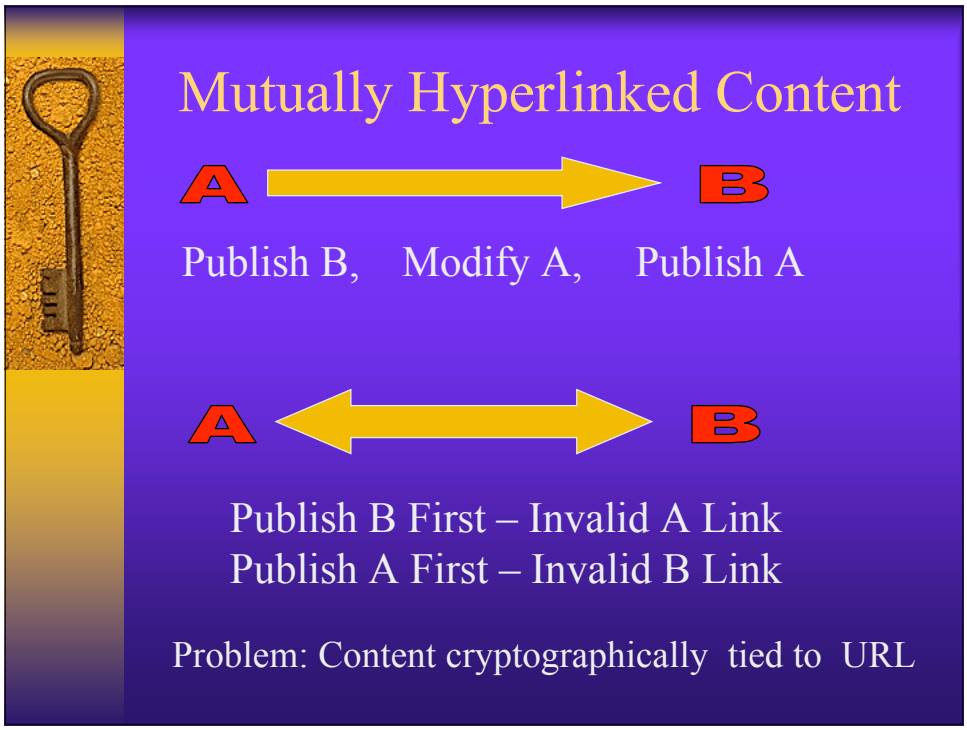
Tradeoffs

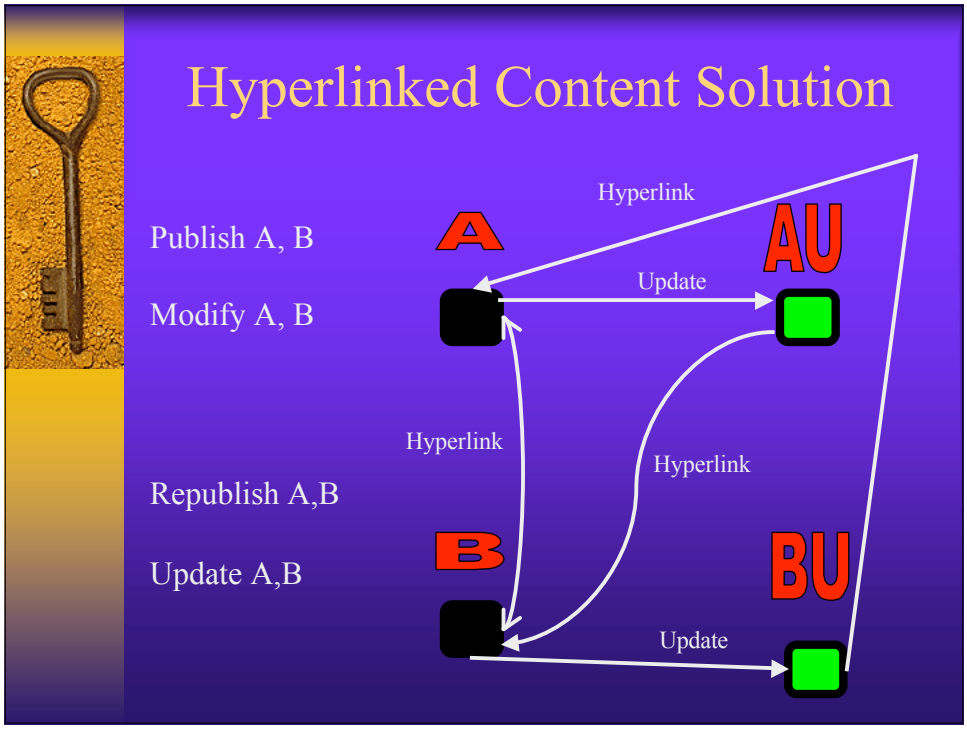
- N = # servers with Content & Share
- K = # Shares needed to reconstruct the Key
- Higher N
  - Greater availability
  - Harder to censor
- Higher K
  - Decreased performance
  - Greater tamper protection
  - Possibly Easier To Censor

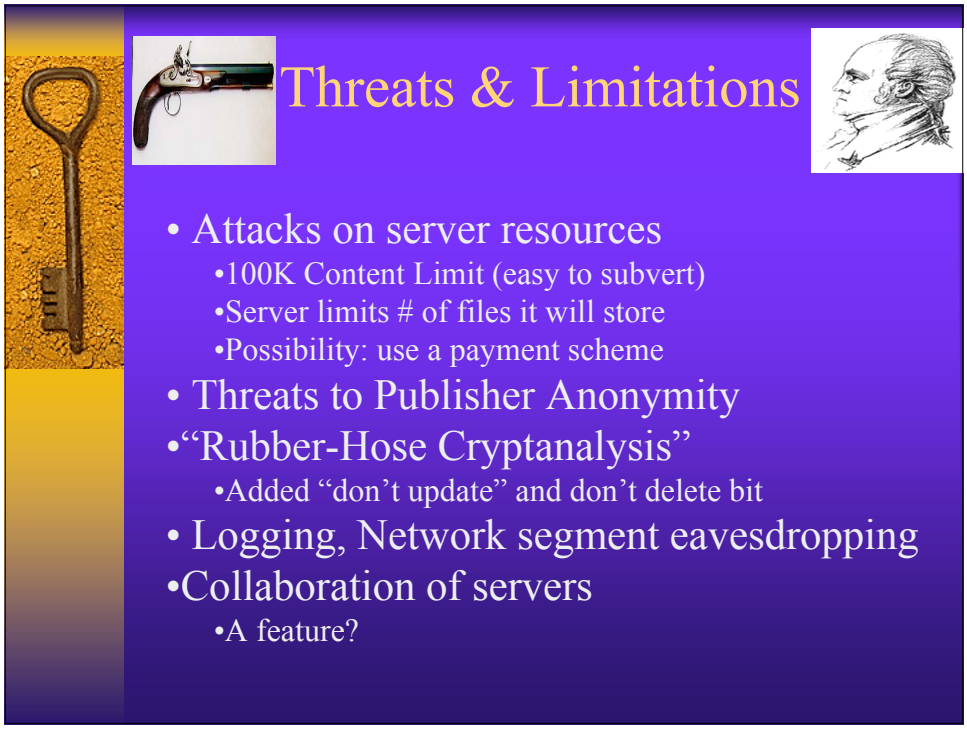
Delete Operation

- URL = http://!anon!/ B_1 B_2 B_3… B_{10}
- $Location_i$ = Table[(Base64Decode (B_i) MOD n) + 1]
- PW = User Password
- P = MD5($Location_i$ · PW)
  - avoids storing PW on all servers
- Store P on Server in file “PASSWORD” during Publish
- Server verifies PW before deleting file


Update Operation
• URL = http://!anon!/ B_1 B_2 B_3… B_{10}
• UpdateURL = http://!anon!/ U_1 U_2 U_3… U_{10}
• $Location_i$ = Table[(Base64Decode (B_i) MOD n) + 1]
• Store UpdateURL in file "UPDATE" at $Location_i$
• Perform update only if have password match
• On retrieve check for "UPDATE" file
• Issue browser redirect on matching "UPDATE" response


Mutually Hyperlinked Content
A
B
Publish B, Modify A, Publish A
A
B
Publish B First – Invalid A Link
Publish A First – Invalid B Link
Problem: Content cryptographically tied to URL


Hyperlinked Content Solution
Publish A, B
Modify A, B
Republish A,B
Update A,B
Hyperlink
A
AU
Update
Hyperlink
Hyperlink
B
BU
Update


Threats & Limitations
• Attacks on server resources
•100K Content Limit (easy to subvert)
•Server limits # of files it will store
•Possibility: use a payment scheme
• Threats to Publisher Anonymity
•"Rubber-Hose Cryptanalysis"
•Added "don't update" and don't delete bit
• Logging, Network segment eavesdropping
•Collaboration of servers
•A feature?

User Interface

Browser Based GUI

http://!publius!/URL

http://!publius!/PUBLISH

http://!publius!/UPDATE

http://!publius!/DELETE

Store file extension in first four bytes of file
Send correct Content-Type to browser

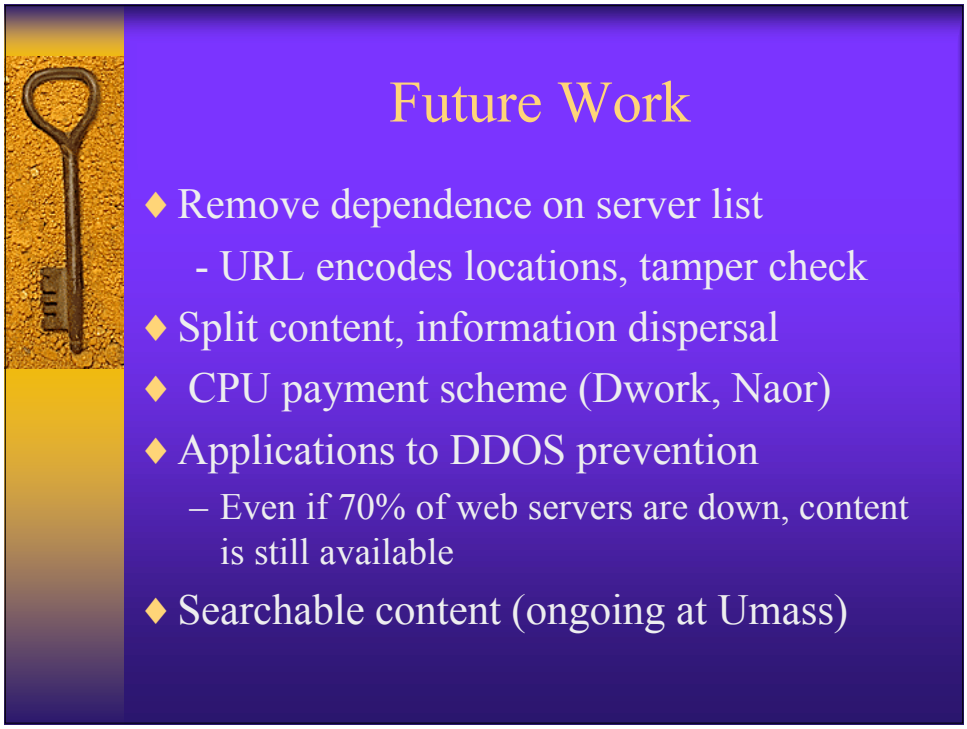
Live Trial

_ 3 Week Server Recruitment Period

_ over 100 Volunteers to be servers
  _ about 53 operational

_ Proxy – running remotely: over a dozen volunteers
  _Users must trust proxy – sees file, password for Publish and URL for retrieve

- Also available: local proxy code
- System still up and running: no plans to turn off
- Occasional surveys of volunteers
- Adding more volunteers all the time


Future Work
♦ Remove dependence on server list
- URL encodes locations, tamper check
♦ Split content, information dispersal
♦ CPU payment scheme (Dwork, Naor)
♦ Applications to DDOS prevention
– Even if 70% of web servers are down, content is still available
♦ Searchable content (ongoing at Umass)