The Free Haven Project:

Distributed Anonymous Storage Service
Seminar: P2P Networks

Michael Janczyk

Albert Ludwigs University Freiburg
Faculty of Applied Sciences
Department of Computer Science
Computer Networks and Telematics
Prof. Schindelhauer

March 1, 2007

Outline

1. Motivation
2. Anonymity
3. Design
4. Future work
5. Conclusion
6. Appendix: Communication

Outline

1 Motivation

2 Anonymity

3 Design

4 Future work

5 Conclusion

6 Appendix: Communication

Motivation

Examples for a requirement of an anonymous storage

- publish freely and . . .
- access to information without fear oft being persecuted
- prevent influential parties from silencing its opponents and critics
- famous example: napster

Motivation

Objectives

- **anonymous persistent distributed** storage
- **protection** against strong adversaries to find or destroy stored data
- **anonymity** – for publishers, readers, servers
- **persistence** – availability of each document for a publisher-specific lifetime
- **flexibility** – system survives as servers leave and join the network
- **accountability** – reputation system limits server-caused damage

Motivation

Design: Entities / Units

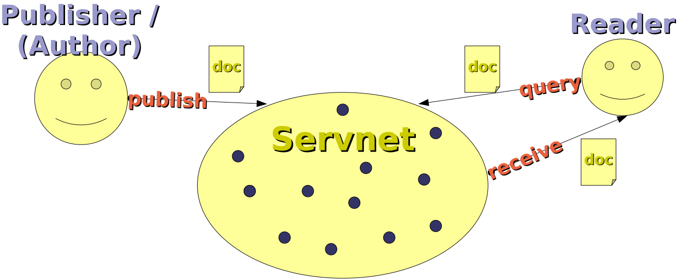
document : unit where information is stored

author : entity who initially creates the document

publisher : entity who places the document into the system

reader : entity who retrieves the document

server : entity who provides services required to keep the system running



Outline

1 Motivation

2 Anonymity

3 Design

4 Future work

5 Conclusion

6 Appendix: Communication

Anonymity

What for . . . ?

- protects the system from adversaries
- provides **'plausible deniability'** [1] for server
- there are different types of anonymity
- anonymity of communication channels needed
- anonymity for:
  - document
  - author
  - publisher
  - reader
  - server

[1] "little or no evidence of wrongdoing or abuse"
(Source: "http://en.wikipedia.org/wiki/Plausible_deniability, 21.02.07")

Anonymity

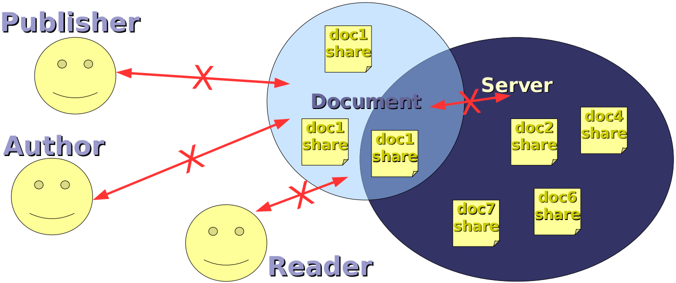
Different types . . .

author anonymity : adversary cannot link author / document

publisher anonymity : adversary cannot link publisher / document

reader anonymity : adversary cannot link reader / document

server anonymity : adversary cannot link server / document



Anonymity

Different types . . .

document anonymity : server doesn't know which documents it's storing

- 1. passive-server : only allowed to look at data it's stroing
  unable to figure out contents of the document
- 2. active-server : communicate and compare data with other servers
  can participate in the network as reader

query-anonymity : server cannot determine document it's serving

- server deniability : weaker form, server knows id. of doc,
  but no 3rd party can be sure of

⇒ **plausible deniability for servers**

Anonymity

. . . and Pseudonymity

- why? participants need to be able to address each other (→ communication)
- pseudonym: attributes of two transactions which can be linked
- example for an author-pseudonymous system:

  > *"documents digitally signed by 'publius' could all be verified 'belonging to publius' without anyone coming to know who 'publius' is in 'real life'."*

- anonymity and pseudonymity protect privacy of user's *location* and *true name*
- anonymity allows no linking at all
- pseudonymity allows *pseudonym* to acquire *reputation* by linking → server reputation

Anonymity

Partial . . .

- anonymity may be impossible, question: *"is it anonymous enough?"*
- example: user lives in california and uses high-bandwidth connection
- adversary can narrow down to a *"set of suspects"*
- set has to be large enough → take action? ↔ too many suspects?
- if an user signs a document with his true name, is the system still anonymous?

⇒ **"what is the responsibility of the system?"**

Outline

1 Motivation

2 Anonymity

3 Design

4 Future work

5 Conclusion

6 Appendix: Communication

Design

Ideas

- system consists of the publication system and the communications channel
- publication system acts as a backend for the communications channel
- based on a community of servers: **'servnet'** (*client* $\neq$ *server*)
- servers host data from other servers in exchange for the opportunity to store its own data

1. publication
2. retrieval
3. expiration
4. revocation
5. trading
6. receipts
7. accountability
8. reputation
9. introducers
10. communication

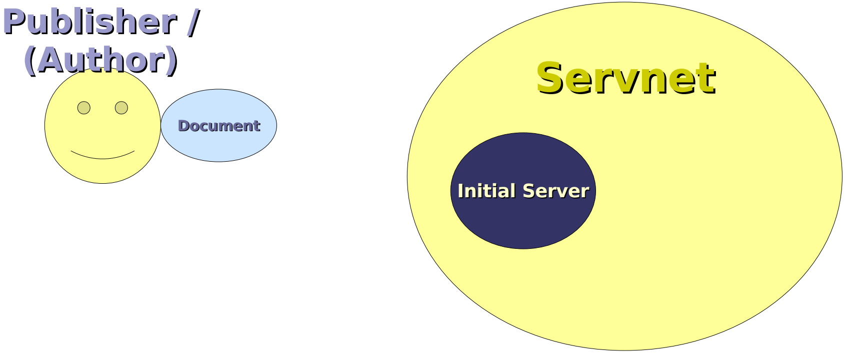
1. identify server which is willing to store document F
   - a run server him-, herself
   - b servers with public interfaces or publically available reply blocks
2. break document F into n shares with IDA [2] $(f_1, \ldots, f_n)$
3. create key pair (PK_{doc}, SK_{doc}) [3]
4. for each share build a data segment and sign it with SK_{doc}
5. save shares into local server's space (next → trade shares f_i)

(steps $2 + 3$ can be performed by the publisher, requires trust of the publisher)

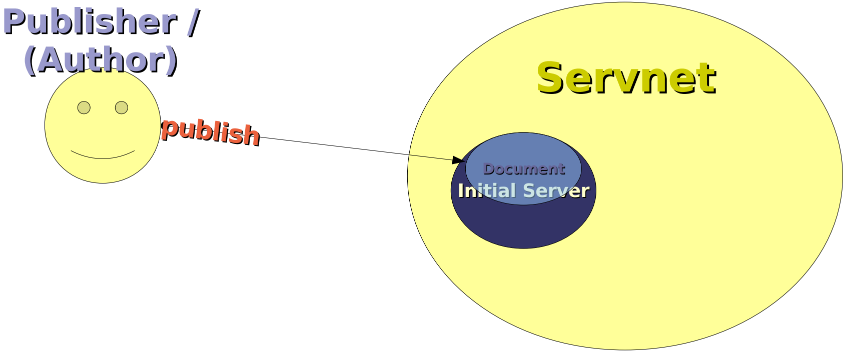
[2] **I**nformation **D**ispersal **A**lgorithm, any i shares are sufficient for recreation

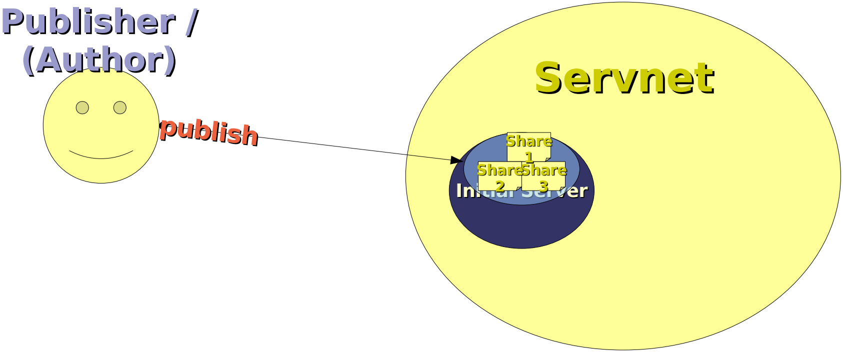
[3] **P**ublic **K**ey, **S**ecret **K**ey, keys for signing the document

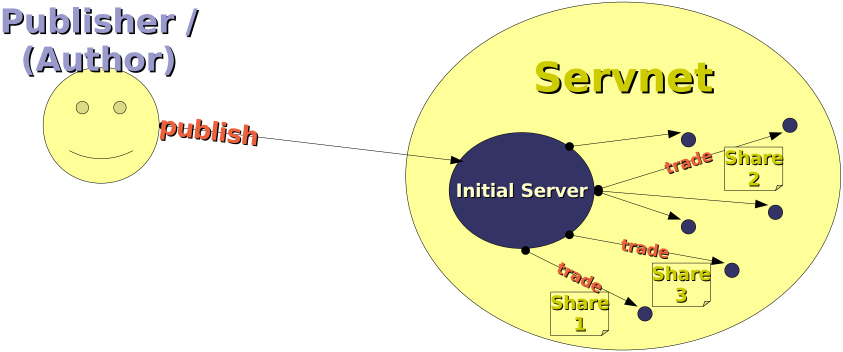
Design
Publication



Design
Publication




Publisher /
(Author)
publish
Servnet
Initial Server
Share
1
Share
2
Share
3


Publisher /
(Author)
publish
Servnet
Initial Server
trade
Share
2
trade
Share
3
trade
Share
1

Design

Publication: Share

Example

```
<share>
<PKdoc>cec41f889d75697304e89edbdddf243662d8c784</PKdoc>
<sharenum>1</sharenum>
<buddynum>0</buddynum>
<totalshares>100</totalshares>
<sufficientshares>60</sufficientshares>
<expiration>2000-06-11-22:25:24</expiration>
<data>Ascii-armored characters here</data>
<signature>cec41f889d75697304e89edbdddf243662d8c784</signature>
</share>
```

- `<expiration>` GMT, when the share is free to be deleted
- all information up to and including `</data>` is signed
  the value is placed inside the `<signature>` tags

Design

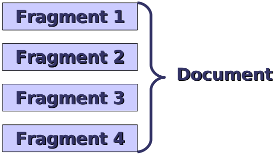
Publication: Create shares

- with Rabin's Information Dispersal Algorithm
- each document is split up into k fragments
- Rabin's IDA disperses the k input fragments into n output fragments ($n \geq k$)
- to rebuild the original fragments use any subset of i shares ($k \leq i \leq n$)

Design

Publication: Create shares

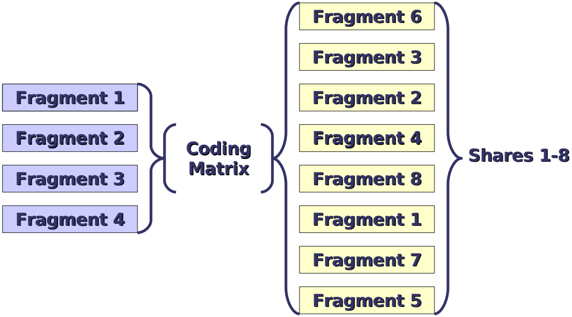
- with Rabin's Information Dispersal Algorithm
- each document is split up into k fragments
- Rabin's IDA disperses the k input fragments into n output fragments $(n \geq k)$
- to rebuild the original fragments use any subset of i shares $(k \leq i \leq n)$



Design

Publication: Create shares

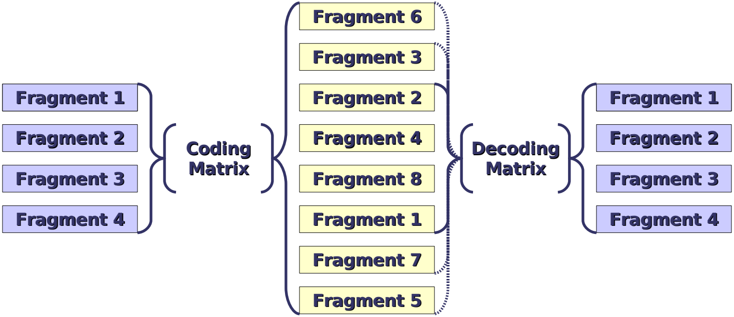
- with Rabin's Information Dispersal Algorithm
- each document is split up into k fragments
- Rabin's IDA disperses the k input fragments into n output fragments ($n \geq k$)
- to rebuild the original fragments use any subset of i shares ($k \leq i \leq n$)



Design

Publication: Create shares

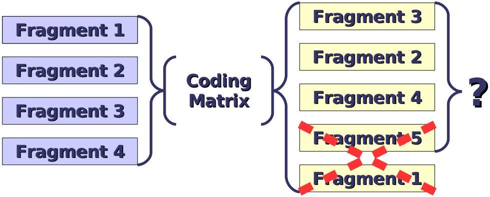
- with Rabin's Information Dispersal Algorithm
- each document is split up into k fragments
- Rabin's IDA disperses the k input fragments into n output fragments ($n \geq k$)
- to rebuild the original fragments use any subset of i shares ($k \leq i \leq n$)



Design

Publication: Number of shares

- k based on compromize between importance and size
  high k ⇒ file brittle, unrecovable after a few shares are lost
  low k ⇒ indicates large file, since more data is stored in each share
- redundancy of $r = \frac{n}{k}$ (robustness parameter)



Blakley's scheme, 3 dimensions ($k = 3$):

- a plane symbolizes a share
- two shares aren't sufficient to determine the secret (enough information to narrow it down to a straight line)
- the point at which the three planes intersect represents the secret

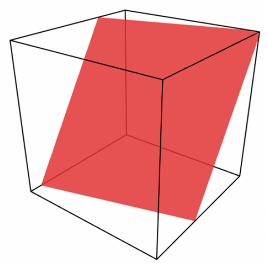


Figure: Secret sharing (Blakley's scheme, 3 dimensions) [4]

[4] (Source: "http://en.wikipedia.org/wiki/Secret_sharing, 26.02.07")

Blakley's scheme, 3 dimensions ($k = 3$):

- a plane symbolizes a share
- two shares aren't sufficient to determine the secret (enough information to narrow it down to a straight line)
- the point at which the three planes intersect represents the secret

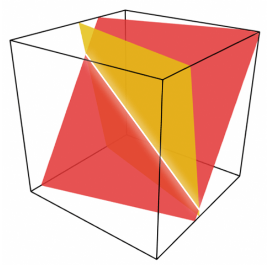


Figure: Secret sharing (Blakley's scheme, 3 dimensions) [4]

[4] (Source: "http://en.wikipedia.org/wiki/Secret_sharing, 26.02.07")

Design

Publication: Excursus - Secret sharing

Blakley's scheme, 3 dimensions ($k = 3$):

- a plane symbolizes a share
- two shares aren't sufficient to determine the secret (enough information to narrow it down to a straight line)
- the point at which the three planes intersect represents the secret

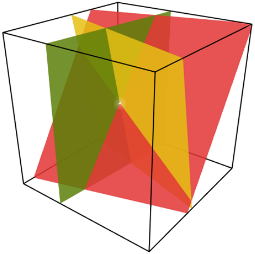
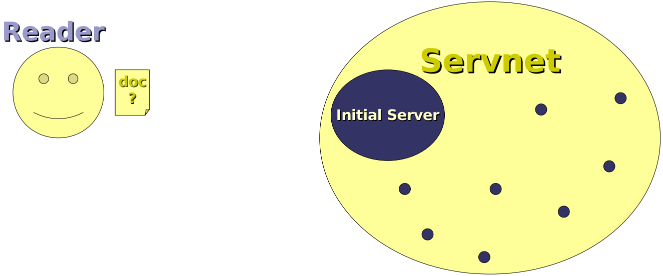


Figure: Secret sharing (Blakley's scheme, 3 dimensions) [4]

[4] (Source: "http://en.wikipedia.org/wiki/Secret_sharing, 26.02.07")

Design
Retrieval

- document is indexed $H(PK_{doc})$
  1. reader generates keypair $(PK_{client}, SK_{client})$ and an one-time remailer reply block [5] and sends it to a server (UI or reply block)
  2. this server broadcasts $('request', H(PK_{doc}), PK_{client}, reply\ block)$
  3. when one server finds index $H(PK_{doc})$
  4. it encrypts the share $PK_{client}(f_i)$ and sends it through the remailer
  5. when the reader receives enough shares f_i $(\geq k)$, the document can be recreated

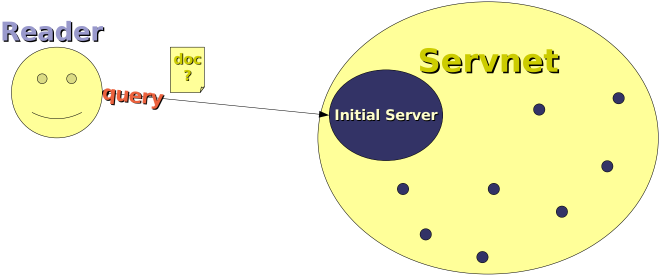


[5] routing instructions, anonymous communication → Communication

Design

Retrieval

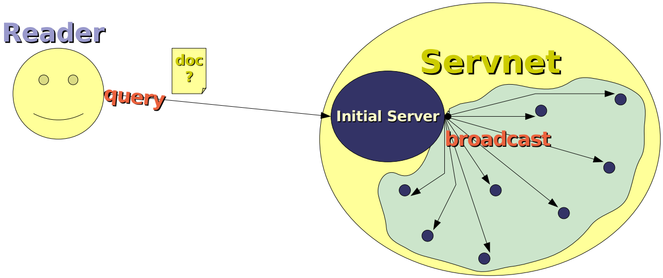
- document is indexed $H(PK_{doc})$
  1. reader generates keypair $(PK_{client}, SK_{client})$ and an one-time remailer reply block [5] and sends it to a server (UI or reply block)
  2. this server broadcasts $('request', H(PK_{doc}), PK_{client}, reply\ block)$
  3. when one server finds index $H(PK_{doc})$
  4. it encrypts the share $PK_{client}(f_i)$ and sends it through the remailer
  5. when the reader receives enough shares f_i $(\geq k)$, the document can be recreated



[5] routing instructions, anonymous communication → Communication

Design
Retrieval

- document is indexed $H(PK_{doc})$
  1. reader generates keypair $(PK_{client}, SK_{client})$ and an one-time remailer reply block [5] and sends it to a server (UI or reply block)
  2. this server broadcasts $('request', H(PK_{doc}), PK_{client}, reply\ block)$
  3. when one server finds index $H(PK_{doc})$
  4. it encrypts the share $PK_{client}(f_i)$ and sends it through the remailer
  5. when the reader receives enough shares f_i $(\geq k)$, the document can be recreated

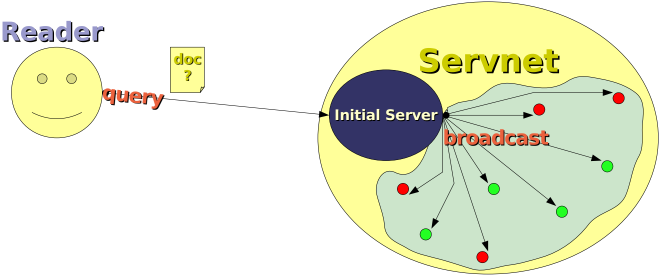


[5] routing instructions, anonymous communication → Communication

Design

Retrieval

- document is indexed $H(PK_{doc})$
  1. reader generates keypair $(PK_{client}, SK_{client})$ and an one-time remailer reply block [5] and sends it to a server (UI or reply block)
  2. this server broadcasts $('request', H(PK_{doc}), PK_{client}, reply\ block)$
  3. when one server finds index $H(PK_{doc})$
  4. it encrypts the share $PK_{client}(f_i)$ and sends it through the remailer
  5. when the reader receives enough shares f_i $(\geq k)$, the document can be recreated

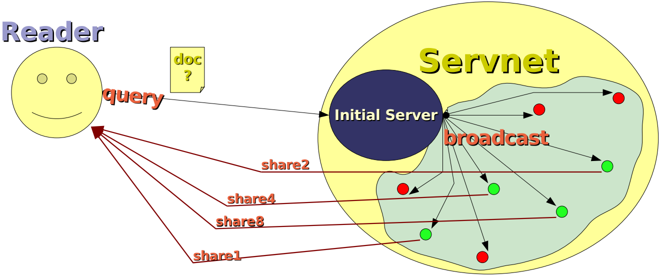


[5] routing instructions, anonymous communication → Communication

Design

Retrieval

- document is indexed $H(PK_{doc})$
  1. reader generates keypair $(PK_{client}, SK_{client})$ and an one-time remailer reply block [5] and sends it to a server (UI or reply block)
  2. this server broadcasts $('request', H(PK_{doc}), PK_{client}, reply\ block)$
  3. when one server finds index $H(PK_{doc})$
  4. it encrypts the share $PK_{client}(f_i)$ and sends it through the remailer
  5. when the reader receives enough shares f_i $(\geq k)$, the document can be recreated

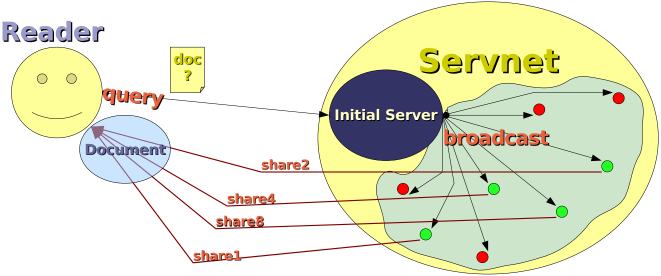


[5] routing instructions, anonymous communication → Communication

Design

Retrieval

- document is indexed $H(PK_{doc})$
  1. reader generates keypair $(PK_{client}, SK_{client})$ and an one-time remailer reply block [5] and sends it to a server (UI or reply block)
  2. this server broadcasts $('request', H(PK_{doc}), PK_{client}, reply\ block)$
  3. when one server finds index $H(PK_{doc})$
  4. it encrypts the share $PK_{client}(f_i)$ and sends it through the remailer
  5. when the reader receives enough shares f_i $(\geq k)$, the document can be recreated



[5] routing instructions, anonymous communication → Communication

Design

Expiration

- absolute timestamp (GMT)
- indicating time after a server may delete a share with no ill consequences
- Freenet and Mojo Nation favor popular documents (LRU)
- $prize\ of\ share = size * lifetime$ ($\rightarrow$ 'currency' for trading)

Design

Revocation

- allows updating documents
- delete documents with infinite lifetime
- one solution:
  1. store hash of private value $H(RevKey)$ into each share
  2. to revoke broadcast *'RevKey'* to all servers
- but new problems:
  1. new attacks
  2. inconsistency ⇒ revocation may not reach all servers
  3. authors may use same value *'RevKey'* for new shares and so 'link' them
  4. presence of a hash in a share assigns 'ownership' to a share
  5. adversary has incentive to find who controls capability to revoke and force him/her to revoke

⇒ *"revocation is left out of the current design" (Dec 2000)*

Design

Trading: Benefits

1. provide cover for publishing:
   if trades are common there is no indication that *trader* = *publisher*
   ⇒ publisher anonymity enhanced
2. let servers join / leave:
   trade for short-lived shares and wait them to expire
3. permit longer expiration dates:
   long-lasting share would be rare if shares had to be kept several years
4. accomodate ethnical concerns of server operators:
   trade away documents you don't want to be associated with
5. provide moving target:
   no static target to attack

Design

Trading: How it works

- frequency set by server
- server (Alice) offers share to another server (Bob) and requests size and duration of a return share
- a 'fair' trade is based on *size* $*$ *duration* ('currency')
  long duration + *larger size* ⇒ *more expensive*
- 4-round handshake:
  1+2 shares are being exchanged
  3+4 receipts are being sent to each other and to each buddy
- with the receipt a server makes a commitment to store a share

Design

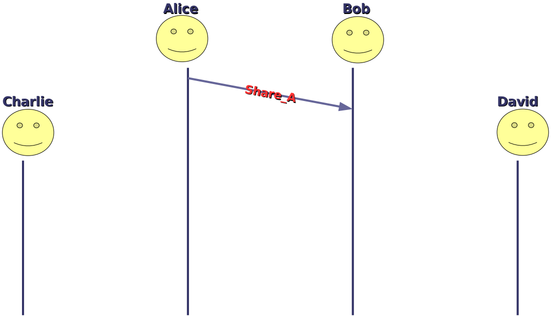
Trading: 4-round handshake

1 Alice trades `Share_A` to Bob

2 Bob trades `Share_B` to Alice

3 Alice sends receipt of `Share_B` to Bob and to `Share_B`'s buddy

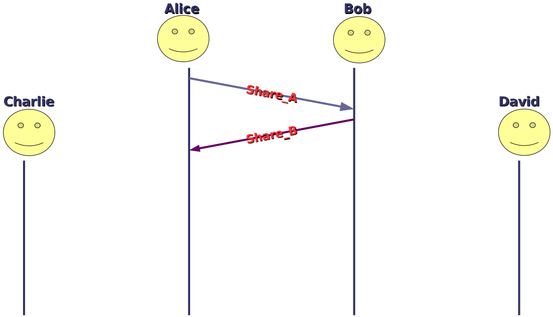
3 Bob sends receipt of `Share_B` to its buddy



Design

Trading: 4-round handshake

1 Alice trades `Share_A` to Bob
2 Bob trades `Share_B` to Alice
3 Alice sends receipt of `Share_B` to Bob and to `Share_B`'s buddy
3 Bob sends receipt of `Share_B` to its buddy



Design

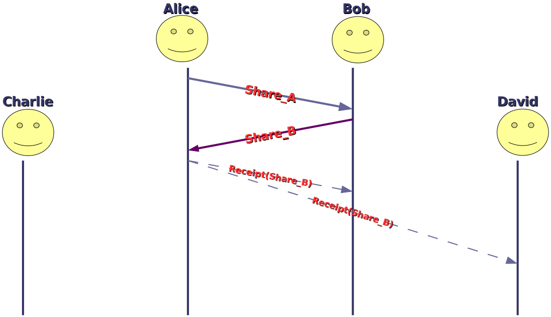
Trading: 4-round handshake

1 Alice trades `Share_A` to Bob

2 Bob trades `Share_B` to Alice

3 Alice sends receipt of `Share_B` to Bob and to `Share_B`'s buddy

3 Bob sends receipt of `Share_B` to its buddy



Design

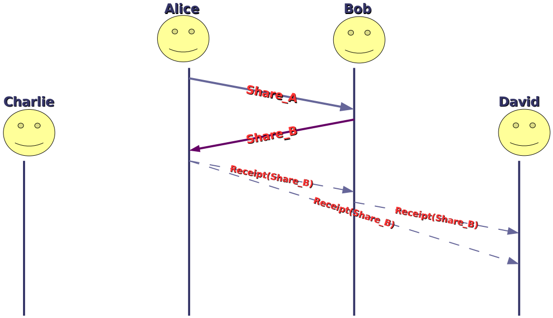
Trading: 4-round handshake

1 Alice trades `Share_A` to Bob

2 Bob trades `Share_B` to Alice

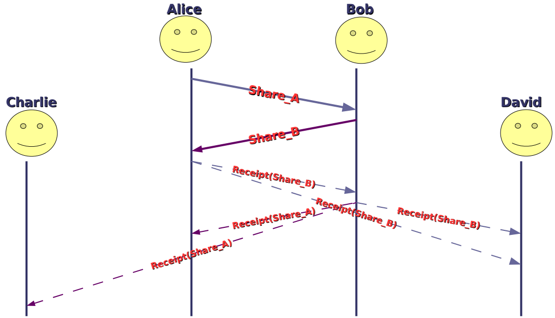
3 Alice sends receipt of `Share_B` to Bob and to `Share_B`'s buddy

3 Bob sends receipt of `Share_B` to its buddy



4 Bob sends receipt of `Share_A` to Alice and to `Share_A`'s buddy

4 Alice sends receipt of `Share_A` to its buddy

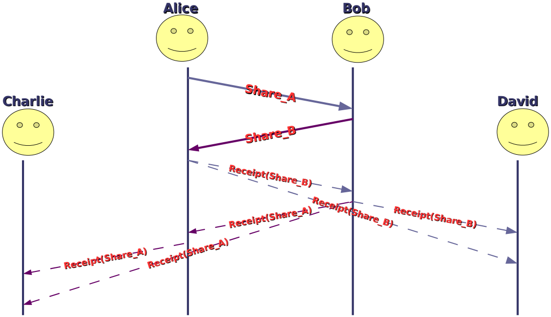


Design

Trading: 4-round handshake

4 Bob sends receipt of `Share_A` to Alice and to `Share_A`'s buddy

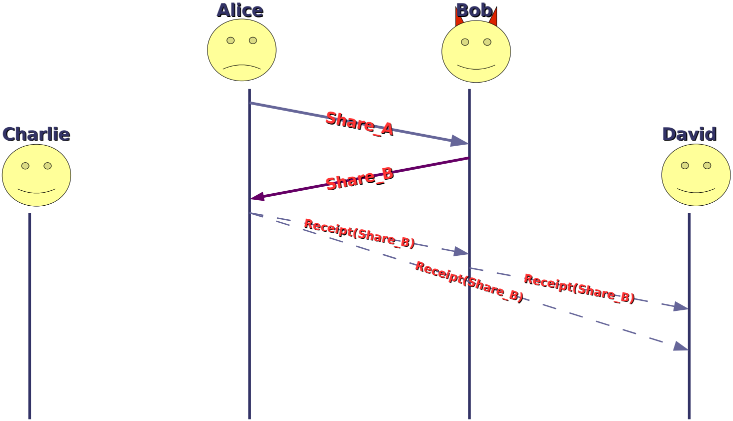
4 Alice sends receipt of `Share_A` to its buddy



Design

Trading: Problems

- after the third step Bob could cheat and refuse to send a receipt (with the receipt a server makes a commitment to store a share)



Trading: Problems

- only possibility for Alice is to send a complaint and hope that the reputation system punishes Bob
- servers should keep traded share for a while, just in case the other server proves untrustworthy
- this means an overhead (about 2x)
- but provides greatly increased rubustness

Design

Receipts

Receipt, signed by server (Alice)

```
'I am'        : Alice
'I traded to': Bob
'I gave away': H(PK_[S_A]), share_num_[S_A], expiration_date_[S_A], size_[S_A]
'I received' : H(PK_[S_B]), share_num_[S_B], expiration_date_[S_B], size_[S_B]
'Timestamp'  : timestamp_[GMT]
```

- when a server (Alice) complains about another server (Bob) it can broadcast a complaint including this receipt
- receipt gives information if a share should be valid (`expiration_date_[S_A]` and document index `H(PK_[S_A])`)
- reputation system computes gravity of this misbehaviour
- receipt proves half of the transaction

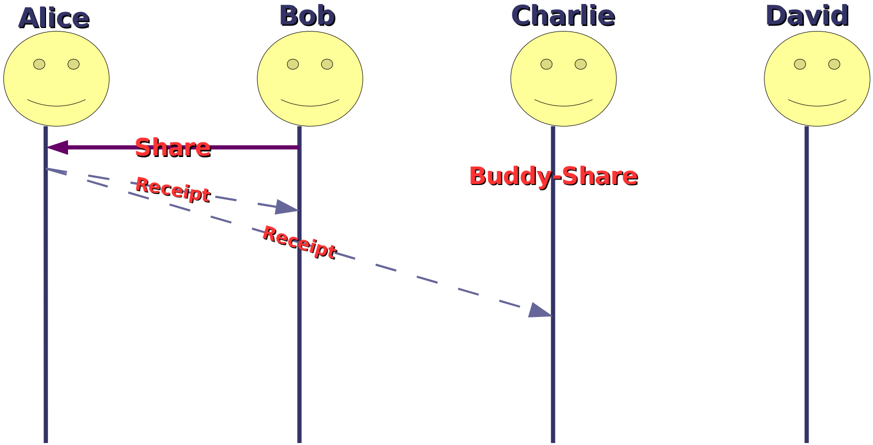
Design
Accountability

- **’buddy system’** (pairs of buddies)
- each share maintains information about the other share
- if a share moves, it notifies its buddy
- periodically querying for buddies ⇒ still alive? ⇒ report anomalities
- during a trade two receipts are sent to each buddy
- if buddy was traded away during that, the receipts should be forwarded [6]
- receipts, and so forwarding address, is kept until expiration date of the document
- share spawning when buddy disappears?
  Free Haven: **NO**
  fear of ”*exponential population explosion of shares*”

[6] latency can be hours (days) → Communication

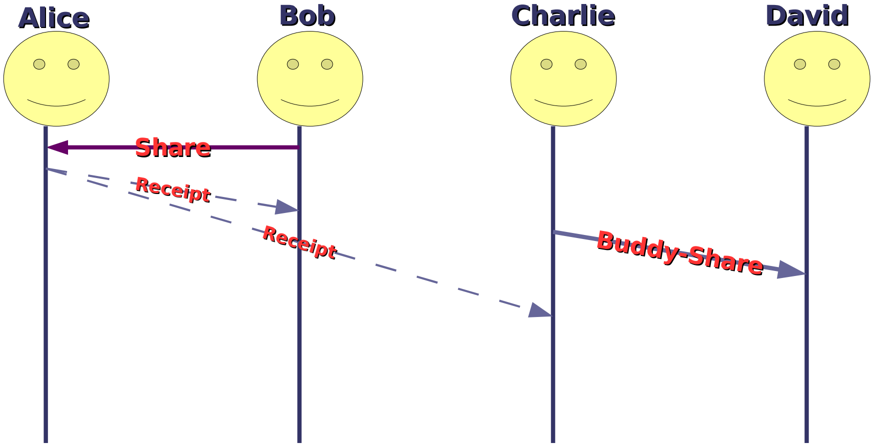
Design

Accountability: Forwarding



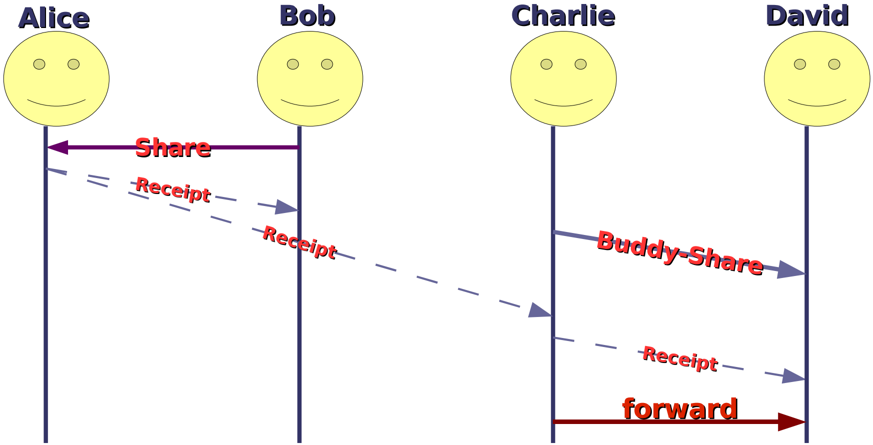
Design

Accountability: Forwarding



Design

Accountability: Forwarding



Design

Accountability

Example

```
// share with buddy
<share>
<PKdoc>cec41f889d75697304e89edbdddf243662d8c784</PKdoc>
<sharenum>1</sharenum>     // buddy-
<buddynum>0</buddynum>     // pairs
<totalshares>100</totalshares>
<sufficientshares>60</sufficientshares>
...
</share>
-------------------------------------------------------------------------------
// receipt with forwarding address
’I am’       : Alice
’I traded to’: Bob     // forwarding address
’I gave away’: H(PK_[S_A]), share_num_[S_A], expiration_date_[S_A], size_[S_A]
’I received’ : H(PK_[S_B]), share_num_[S_B], expiration_date_[S_B], size_[S_B]
’Timestamp’  : timestamp_[GMT]
```

Design

Reputation

- create accountability
- each server should keep track of servers it knows:
  - reputation: belief that a server will obey the protocol
  - credibility: belief that utterances of a server are valuable
  - confidence rating: represents the 'stiffness' of the two values
- a server broadcasts referrals
  - after a completing a trade
  - when buddies are lost
  - when reputation / credibility change substantially
- difficult in a system commited to anonymity
- there are many attacks

Introducers

- servers with high reputation
- add new servers to the network and remove inactive ones from the network
- at the beginning a new server has no reputation
  $\Rightarrow$ no server wants to trade
  $\Rightarrow$ offer storage space to the network and make one-way trades

Design

Communication: Required operations

the design specifification leads to following required operations:

- anonymously send a message to a node
- anonymous broadcast
- pseudonymously name a node within the network
- add nodes to the communications channel, and . . .
- remove nodes from the channel without impacting functionality

Design

Communication: Desired goals

- low latency to provide timely message transmission
- delivery robustness for messages, messages are reliably transmitted
- routing robustness between any two parties:
  loss of nodes should not imply loss of anonymous communicatation
- resistant to attack
- decentralized, to maintain effciency, security, and reliability

- *Free Haven* will use existing anonymous communication modules
- one solution is to use remailers as communication channel
  ▸ Example Remailer
- the first implementation was intended to use **Cypherpunk(s)** and **Mixmaster** remailers as anonymous channel (Dec 2000)
- a new remailer **Mixminion** combines **Cypherpunk** and **Mixmaster**
- but all remailers have a high latency ⇒ up to hours (days)

Communication: How could it work

- the entities communicate via addresses inside remailer reply blocks
- a remailer reply block is a collection of encrypted routing instructions a bodyless email, addressed to the server itself ▸ Example Onion Routing
- each server has a public key and one (or more) reply blocks
- these provide secure, authenticated, pseudonymous communication
- every server in the servnet has a database with the public keys and reply blocks of the other servers on the network

Outline

1 Motivation

2 Anonymity

3 Design

4 Future work

5 Conclusion

6 Appendix: Communication

Future work

- low-latency pseudonymous channel:
  current channels which support pseudonyms have high latency
- accountability and reputation:
  extremely difficult to reason about accountability, especially 'buddy system'
  an 'anonymous system reputation algebra' for formally reasoning to verify trust protocols
- modelling and metrics:
  a mathematical model of anonymous storage would allow to test and run simulations

Future Work

- formal definition of anonymity
- usability requirements and interface
- efficiency:
  "the efficiency and perceived benefit of the system is more important to an end user than its anonymity properties"

Outline

1 Motivation

2 Anonymity

3 Design

4 Future work

5 Conclusion

6 Appendix: Communication

Conclusion

- project was never realized (27.02.07)
- *freehaven.net* last changed December 1st, 2004
- latest news from Aug 15, 2002:
  "We're not updating this news anymore. :)" [7]
- but different other projects were launched like **Tor** and **Mixminion**

[7] (Source: "http://freehaven.net, 27.02.07")

Conclusion

- the current design is unsuitable for wide deployment
- if inefficient it will lead to few users
  $\Rightarrow$ leads to insufficient anonymity
- one solution: join with efficient file sharing systems
  answer queries for less popular documents, which would have been deleted (LRU)
- high latency of the communications channel

Conclusion

Anonymity

- perfect forward anonymity (pf):
  after a given transaction there is nothing new that can help an adversary
- computational anonymity (c):
  anonymity cannot be broken with 'reasonable' computing power

Free Haven Anonymity

(author)	publisher	reader	server	document	query
$((pf)+(c))$ [8]	$(pf)+(c)$	$(pf)+(c)$	(c)	(c)	–

Thanks!

[8] it's not the goal of the system to provide communication to the publisher, it's the choice of the author himself

Outline

1 Motivation

2 Anonymity

3 Design

4 Future work

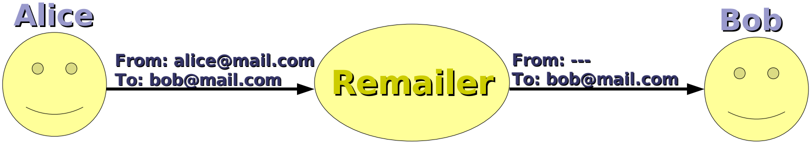
5 Conclusion

6 Appendix: Communication

Appendix: Communication

Remailer

- anonymous communication channel (via e-mail)
- embedded instructions where to forward message
- removes personal information from the header (e-mail address)
- there are 4 types of remailers:
  1. (Pseudo-)Nym(-ous) remailer (type 0)
  2. Cypherpunk remailer (type I)
  3. Mixmaster remailer (type II)
  4. Mixminion remailer (type III)



▸ Back to Conclusion: Anonymity

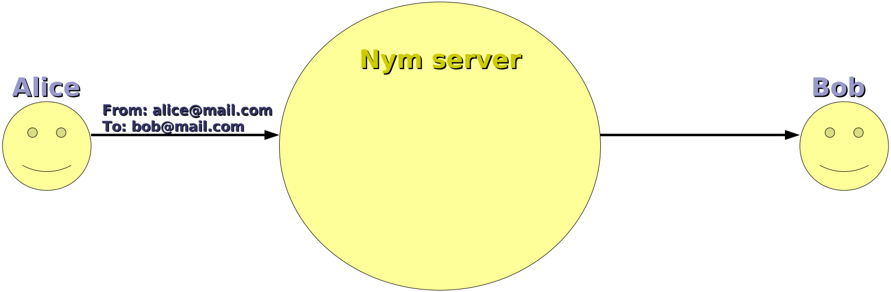
▸ Back to Communication

Nym Server (type 0)

- participants need to be able to address each other ⇒ pseudonyms
- pseudonym remailer allows bidirectional communication
- 'real' e-mail-address is replaced by a pseudonym-address
- contemporary nym servers use encrypted remailer chains
  ⇒ **Cypherpunk, Mixmaster**

▸ Back to Conclusion: Anonymity

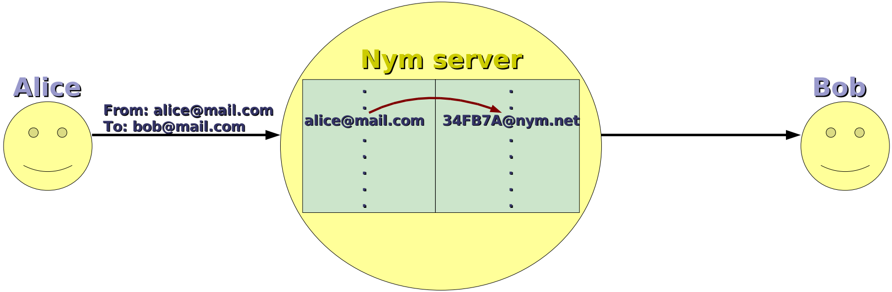
Nym Server (type 0)



▸ Back to Conclusion: Anonymity

Appendix: Communication

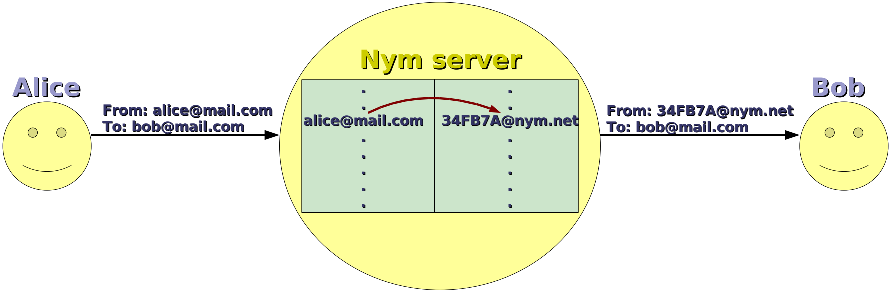
Nym Server (type 0)



▸ Back to Conclusion: Anonymity

Appendix: Communication

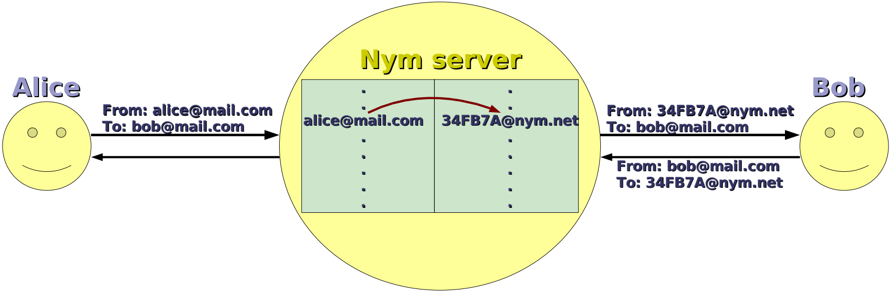
Nym Server (type 0)



▸ Back to Conclusion: Anonymity

Appendix: Communication

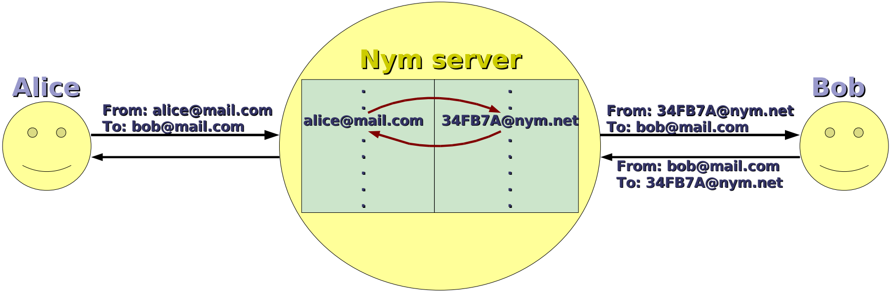
Nym Server (type 0)



Back to Conclusion: Anonymity

Appendix: Communication

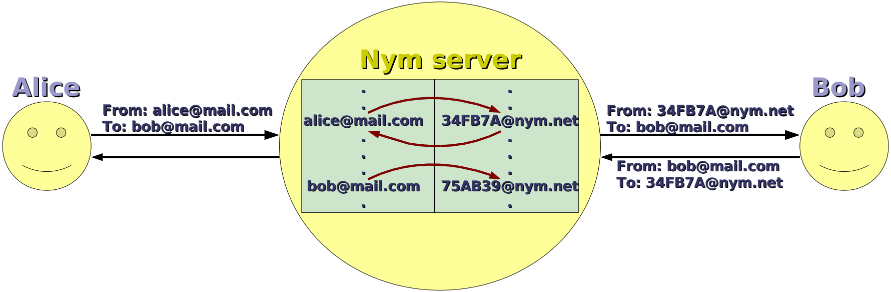
Nym Server (type 0)



▸ Back to Conclusion: Anonymity

Appendix: Communication

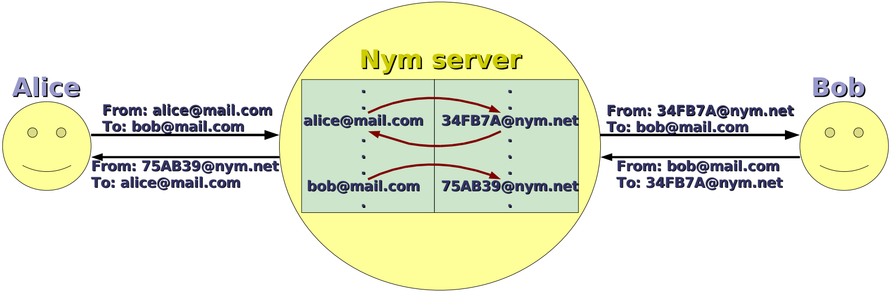
Nym Server (type 0)



▸ Back to Conclusion: Anonymity

Appendix: Communication

Nym Server (type 0)



▸ Back to Conclusion: Anonymity

Appendix: Communication

Cypherpunk Remailer (type I)

- Cypherpunk remailer brought new possibilities:
    - mail can be sent across a chain of remailers
        - first remailer in the chain knows the sender
        - the last remailer knows the recipient
        - and the middle remailers know neither
    - mail can be ecrypted with PK of remailer, even between hops
    - add or remove random data to a mail
    - delay delivery of mail

▸ Back to Conclusion: Anonymity

Appendix: Communication

Cypherpunk Remailer (type I)

Example

1. write message
2. add following lines at the beginning:

```
::
Request-Remailing-To: mail@cypherremailer.net
```

3. encrypt message with PK of the remailer (optional)
4. if message encrypted add this lines at the beginning:

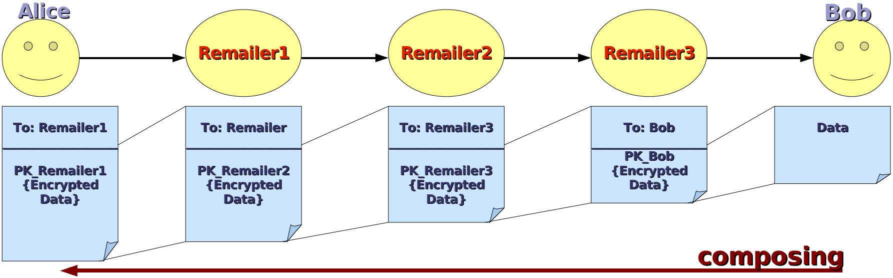
```
::
Encrypted PGP
```

5. repeat steps 1-4 for each hop (optional)

▸ Back to Conclusion: Anonymity

Appendix: Communication

Cypherpunk Remailer (type I): Onion Routing



▸ Back to Conclusion: Anonymity ▸ Back to Communication: How could it work

Appendix: Communication

Mixmaster Remailer (type II)

- needs client / server software which uses spacial packet format
- all packets are the same length
- every message is encrypted
- messages are stored in 'pools'
- once enough messages are in a 'pool' the node forwards a message ramdomly
- for reply blocks use Cypherphunk remailer

▸ Back to Conclusion: Anonymity