Building Dependable Trusted Services

Christian Cachin, Klaus Kursawe, Jonathan Poritz,
Victor Shoup, Reto Strobl

cca@zurich.ibm.com

February 2003

Motivation — Trusted services in distributed systems

- certification authority and revocation server in PKI
- directories and name services (LDAP, DNS)
- key distribution (Kerberos)
- authentication (Kerberos, RADIUS)
- authorization (Tivoli Policy Director)
- digital rights management
- fair exchange, digital receipts
- anonymous communication (re-mailers, anonymizers)

Trusted services

- State-of-the-art: trusted service runs on a dedicated server
- How trustworthy is a trusted service?
  - \+ faith in administrators
  - \+ isolation of function
  - – performance may become a bottleneck
  - – single point of failure

Trusted services

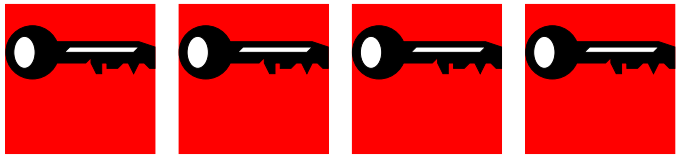
- State-of-the-art: trusted service runs on a dedicated server
- How trustworthy is a trusted service?
  - \+ faith in administrators
  - \+ isolation of function
  - – performance may become a bottleneck
  - – single point of failure
- Improving trust-worthiness:
  - – perfect the single-server approach
  - – add redundancy to tolerate faults and intrusions

Security and redundancy

Seemingly incompatible:

one trusted component

- good security
- poor availability



redundant trusted components

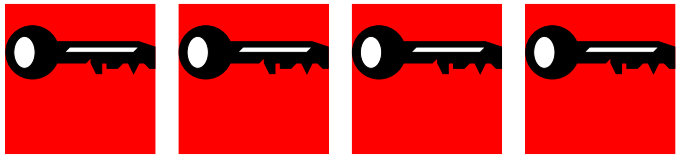
- good availability
- poor security

Security and redundancy

Seemingly incompatible:

one trusted component

- good security
- poor availability

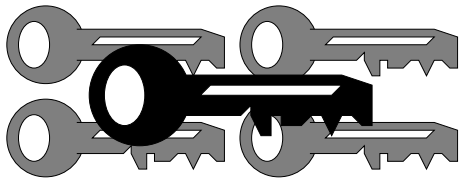


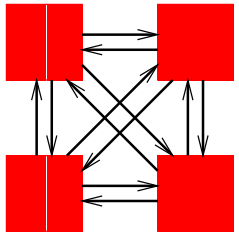
redundant trusted components

- good availability
- poor security

Solution: **distribute keys and control** by using

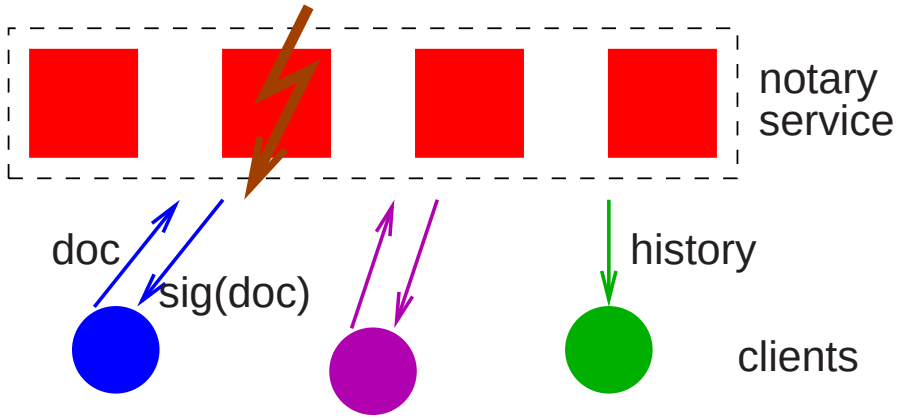
threshold cryptography *and* replication protocols





Service replication in Asynchronous Networks

Ex.: a digital notary service with four servers, of which one may fail.

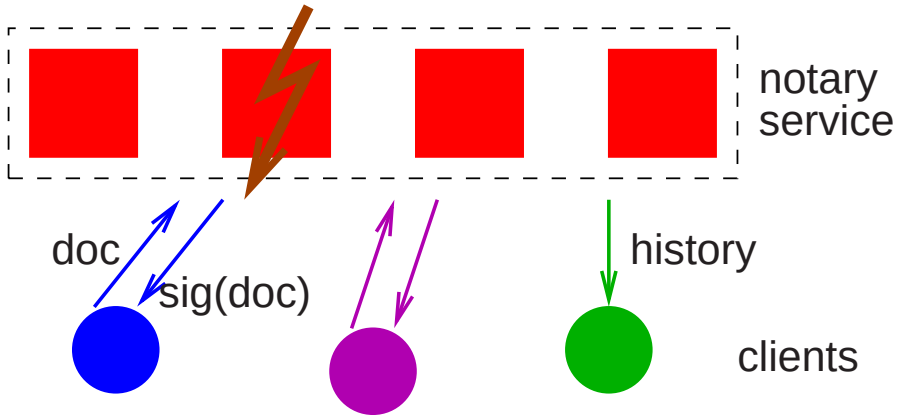


Technical problems:

- how to sign distributedly?
- which document to sign?
- all servers in same order?
- how to sort the requests?

Service replication in Asynchronous Networks

Ex.: a digital notary service with four servers, of which one may fail.



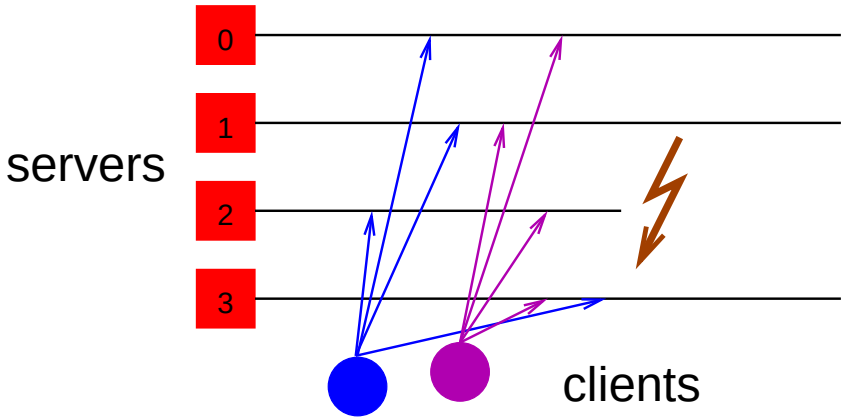
Technical problems:

- how to sign distributedly? $\Rightarrow$ threshold cryptography
- which document to sign? $\Rightarrow$ Byzantine agreement
- all servers in same order? $\Rightarrow$ atomic broadcast
- how to sort the requests? $\Rightarrow$ causal order

SINTRA – asynchronous MAFTIA middleware

- SINTRA – secure intrusion-tolerant replication architecture
- APIs for secure replication library and prototype implementation
- provides coordination, agreement, and atomic broadcast
- Java 1.2 and above
- benchmarks show that approach is feasible (0.5–3s latency)
- applications to
  - reliable, secure distributed services with active replication
  - secure group communication

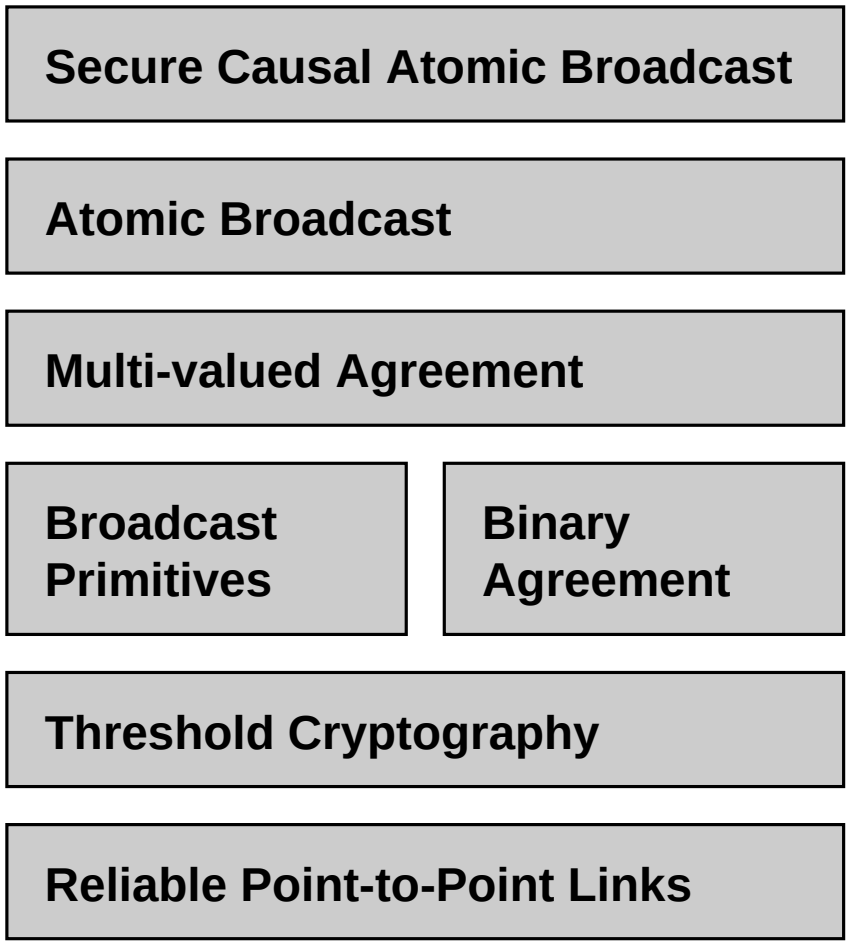
System model



- static* group of n servers
  - on different networks
  - with different operating systems and configurations
- tolerate $t < n/3$ corrupted servers (Byzantine faults and crashes*)
- reliable asynchronous* point-to-point network
- trusted* initialization and key distribution
- cryptographic security proofs in random oracle model*
- any number of potentially corrupted clients

(* potential for future work!)

Protocol architecture



causal order through encryption

total order of requests

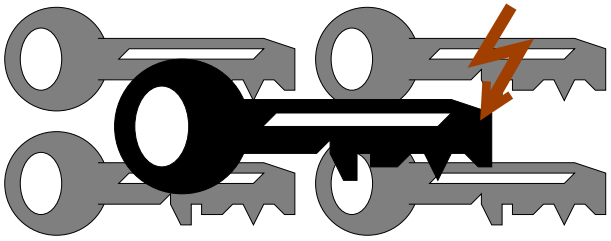
agreement on requests

consistent and reliable broadcast

agreement on binary values

encryption, signatures, and unpredictable pseudorandom values

Threshold cryptography



- global public key, every server holds a share of the secret key
- in (n, k, t) dual-threshold scheme (traditionally, $k = t + 1$)
  - any k servers may carry out cryptographic operation
  - up to t servers corrupted
- operation proceeds without leaking information about the secret key
- many public-key cryptosystems: signatures, encryption, unpredictable pseudorandom bits
- communication model is critical (non-interactive schemes preferred: [S00], [SG98], [CKS00])

Threshold pseudorandom coin
[NPR99], [CKS00]

- based on Diffie-Hellman problem in group $G =< g >$ of prime order q
- secret key $x \in \mathbb{Z}_q$, public key $y = g^x$
- shares $x_1, \ldots, x_n$ that interpolate to x on polynomial of degree $k-1$
- pseudorandom coin function F is defined as

$$F(s) \;=\; H\big(\,\mathcal{H}(s)^x\,\big)$$

  using hash functions $\mathcal{H} : \{0,1\}^* \to G$ and $H : G \to \{0,1\}^\ell$
- non-interactive: server i reveals a coin share $\mathcal{H}(s)^{x_i}$ with a *correctness proof*; coin value can be assembled from k correct shares
- security analysis in the random oracle model (for $\mathcal{H}$)

Threshold cryptography

- public-key encryption
  - based on Diffie-Hellman problem [SG98]
- digital signatures
  - based on RSA [S00]
  - vector of n standard digital signatures ("multi-signature")
- common coin (provides unpredictable pseudorandom bits)
  - based on Diffie-Hellman (DH) problem [CKS00]
- all schemes are non-interactive

Reliable broadcast with crash failures only

Validity: non-faulty sender r-broadcasts m $\Rightarrow$ sender eventually r-delivers a message

Agreement: some non-faulty server eventually r-delivers m $\Rightarrow$ every non-faulty server r-delivers m

Termination: sender not faulty $\Rightarrow$ all servers eventually terminate

Protocol —

upon r-broadcast(m) // sender only
  send message (send, m) to all parties

upon receiving message (send, m)
  send message (send, m) to all parties
  r-deliver(m)

Consistent broadcast

Validity: honest sender c-broadcasts $m \Rightarrow$ sender eventually c-delivers a message

Consistency: some honest server c-delivers m and a distinct honest server c-delivers $m' \Rightarrow m = m'$

Authenticity: sender honest and c-broadcasts $m \Rightarrow$ sender c-delivers m

Termination: sender honest $\Rightarrow$ all servers eventually terminate

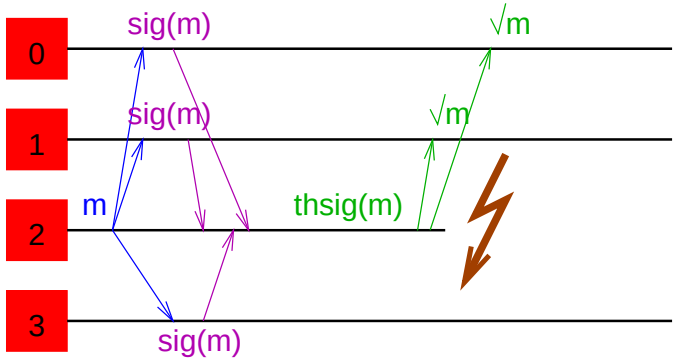
Consistent broadcast

Validity: honest sender c-broadcasts $m \Rightarrow$ sender eventually c-delivers a message

Consistency: some honest server c-delivers m and a distinct honest server c-delivers $m' \Rightarrow m = m'$

Authenticity: sender honest and c-broadcasts $m \Rightarrow$ sender c-delivers m

Termination: sender honest $\Rightarrow$ all servers eventually terminate



Protocol echo broadcast [R94] — server 2 broadcasts message m, by obtaining threshold signature from quorum of $\lceil \frac{n+t+1}{2} \rceil$ witnesses

Reliable broadcast I

Validity: honest sender r-broadcasts $m \Rightarrow$ sender eventually r-delivers a message

Agreement: some honest server r-delivers $m \Rightarrow$ every honest server eventually r-delivers m

Authenticity: sender honest and r-broadcasts $m \Rightarrow$ sender r-delivers m

Termination: sender honest $\Rightarrow$ all servers eventually terminate

Reliable broadcast II

Protocol [B85] —

upon r-broadcast(m) // sender only
send message (send, m) to all parties

upon receiving message (send, m) from sender
send message (echo, m) to all parties

upon receiving $\lceil \frac{n+t+1}{2} \rceil$ messages (echo, m^*) for m^* and not sent a ready-message
send message (ready, m^*) to all parties

upon receiving $t+1$ messages (ready, m^*) for m^* and not sent a ready-message
send message (ready, m^*) to all parties

upon receiving $2t+1$ messages (ready, m^*) for m^*
r-deliver(m^*)

Asynchronous Byzantine agreement

- agreement on a binary value proposed by honest servers
- conditions —

  Validity: all honest servers propose $v \Rightarrow$ some honest server eventually decides for v

  Agreement: some honest server decides for b and a distinct honest server decides for $b' \Rightarrow b = b'$

  Termination: all servers eventually terminate

- impossible to achieve in asynchronous networks with deterministic protocols [FLP85]
- efficient randomized protocols exist, using public-key cryptography

Asynchronous binary Byzantine agreement protocols

- constant expected rounds, using unpredictable common coin
- protocol structure [BO83], [R83], [CKS00] —

  $v \leftarrow$ proposed value in $[0, 1]$; $r \leftarrow 0$;
  while not decided **do**
    vote(v); receive $n - t$ *properly justified* votes from others;
    if all votes are v **then**
      decide(v);
    else if *sufficient majority* of votes is v' **then**
      $v \leftarrow v'$;
    else
      $v \leftarrow$ *coin(r)*;
    $r \leftarrow r + 1$;

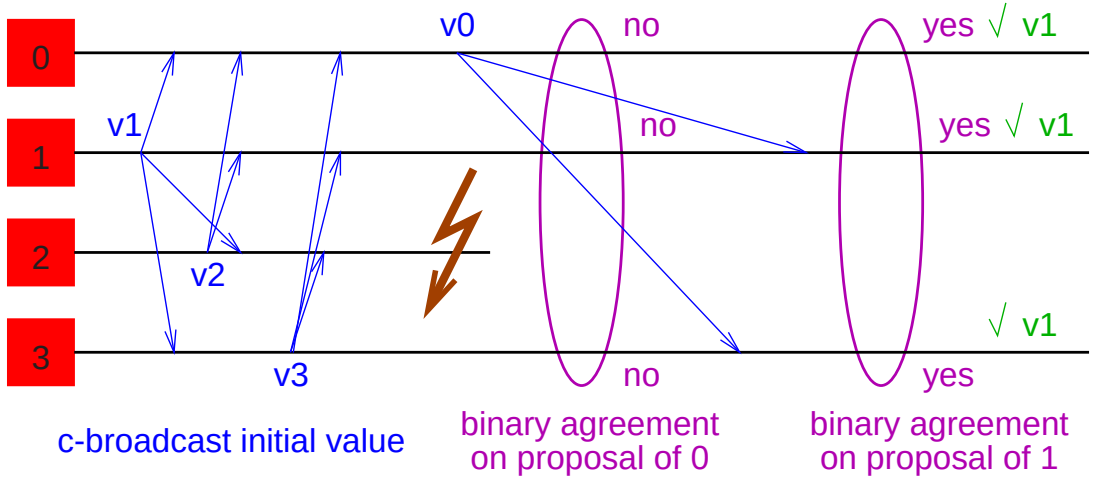
- efficient non-interactive threshold-cryptographic coin implementations exist

Multi-valued Byzantine agreement

- agreement over arbitrary domain requires non-trivial extension of binary case (validity no longer determined by simple majority)
- previous approaches —
  - decide for default value when conflicting values are found
  - "strong consensus" [N94], where domain size $\propto n/t$
- instead [CKPS01] —

  External validity:
  - assume a global predicate Q known to all servers, which determines if value is valid
  - every honest server decides for b such that $Q(b)$ holds

Protocol for multi-valued Byzantine agreement
[CKPS01]



- every server proposes its initial value using consistent broadcast
- receive $n - t$ proposals
- run a sequence $i = 0, 1, \ldots, n - 1$ of binary Byzantine agreements to decide if server i has sent a valid proposal and stop if yes
- decide for proposal of server i

Atomic broadcast

- every server may a-broadcast an unbounded number of messages m
- conditions —

  Validity: some honest server a-broadcasts m $\Rightarrow$ some honest server eventually a-delivers m

  Agreement: some honest server a-delivers m $\Rightarrow$ all honest servers eventually a-deliver m

  Total order: some honest server a-delivers m before m' $\Rightarrow$ any distinct honest server a-delivers m before m'

  Termination: the protocol *eventually makes progress*

- main abstraction for deterministic state machine replication

Protocol for atomic broadcast
[CKPS01]

- uses multi-valued agreement and a signature scheme
- protocol idea —

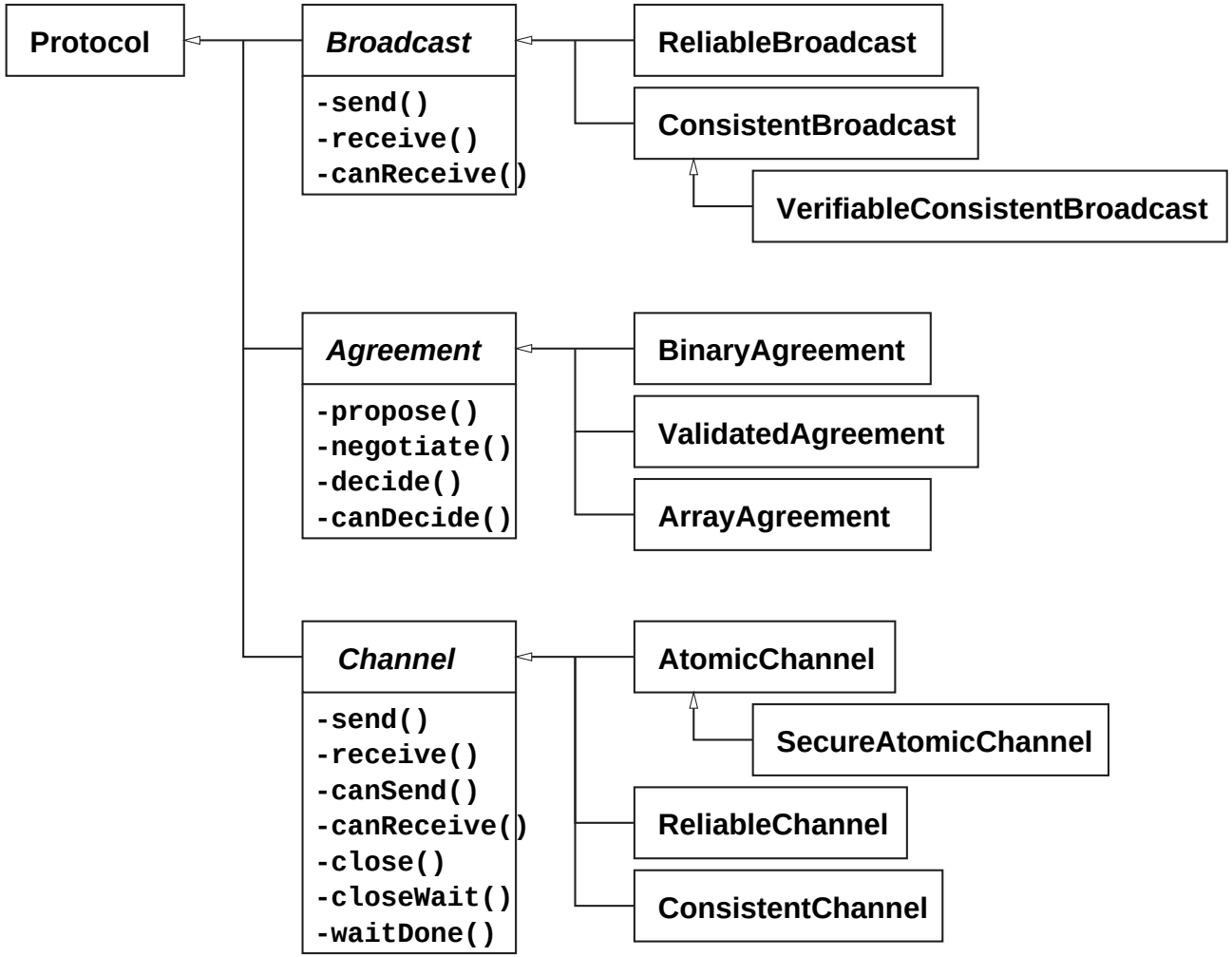
  $M \leftarrow$ FIFO queue of messages to broadcast;
  // messages are *appended* to M upon a-broadcast in another thread
  $r \leftarrow 0$;
  while channel not closed **do**
    $m \leftarrow$ *first*(M);
    $\sigma \leftarrow$ sign(r, m); send (σ, m) to others;
    receive valid (σ, m) for round r from $n - t$ servers; collect them in W;
    propose W in multi-valued agreement of round r; let it decide on V;
    a-deliver $m \in V$ in fixed order;
    remove $m \in V$ from M;
    $r \leftarrow r + 1$;

Secure causal atomic broadcast

- in atomic broadcast, corrupted server may violate causal order
- must maintain "input causality" [RB94]: message secrecy, message integrity, and message consistency
- protocol using non-interactive $(n, t+1, t)$ threshold public-key encryption scheme:

  start an atomic broadcast channel
  // to s-broadcast a message m, compute $c \leftarrow$ *encrypt*(m) and a-broadcast c
  while channel not closed **do**
  receive a-delivered ciphertext c;
  compute decryption share δ for c;
  send δ to others;
  receive $t+1$ decryption shares;
  $m \leftarrow$ *decrypt*(c);
  s-deliver m;

Replication protocols structure



Asynchronous replication prototype details

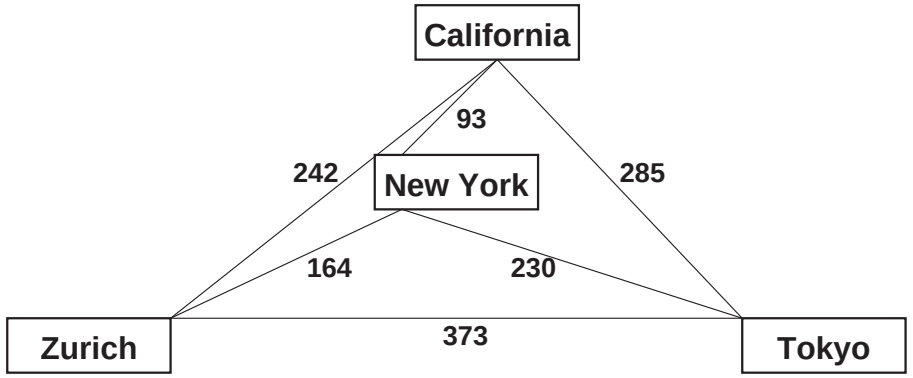
- asynchronous replication protocols in Java
- API example —

```
class Channel extends Protocol {
  void send(byte[] message);       // multiple times, by any server
  byte[] receive();                // multiple times
  boolean canSend();
  boolean canReceive();
  void closeWait();                // waits until closed
  boolean isClosed();
}
```

- multi-threaded architecture, TCP for reliable point-to-point links
- standard Java 1.2 cryptography (JCA/JCE 1.2)
  - link authentication using HMAC-SHA1 (160-bit output)
  - bulk encryption using AES/MARS (128-bit keys)
  - digital signatures using hash&sign RSA (1024-bit keys)

Experiments I

- three setups: LAN ($n = 4$), Internet ($n = 4$) and LAN+I'net ($n = 7$)
- each group with standard desktops/PCs: Pentium II/III-class CPUs; Linux, Windows & AIX operating systems
- Internet setup with average packet round-trip times in ms:



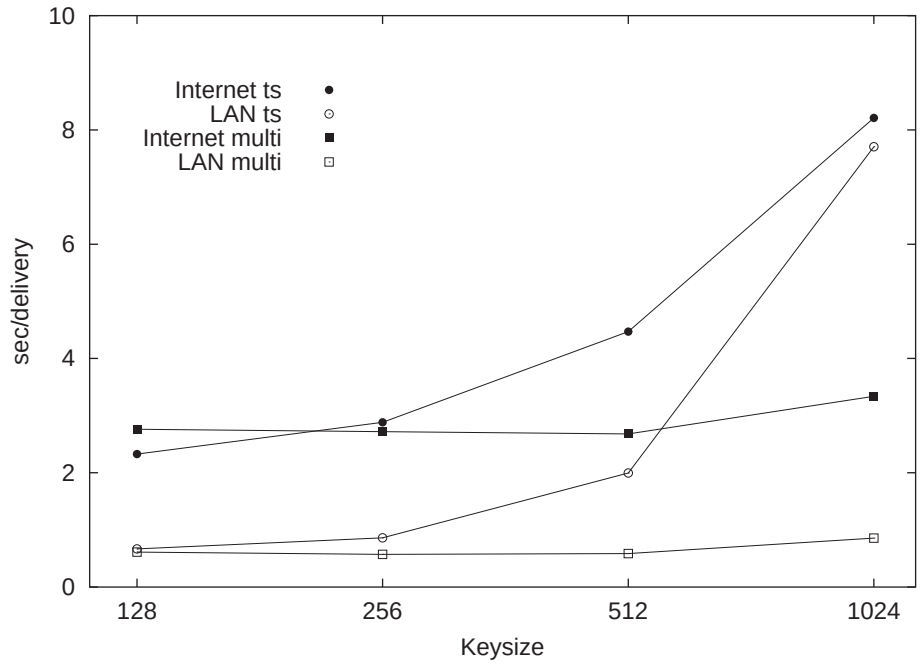
- comparing non-interactive threshold signatures using RSA-threshold signatures and multi-signature vector

Experiments II

- latency (s) with vector multi-signatures and 1024-bit keys:

Setup	consistent	reliable	atomic	secure
LAN	0.05	0.03	0.54	0.76
Internet	0.83	0.72	2.95	3.61
LAN+I'net	0.64	0.60	2.74	3.79

- varying public-key size from artificially small values to 1024 bits:



Related work

- View-synchronous group communication (ISIS–Ensemble, Cornell)
  – tolerate only crash failures
- Rampart (Reiter, AT&T Research)
  – failure detector for liveness and safety
- BFT (Castro & Liskov, MIT)
  – failure detector for liveness
- COCA (Cornell On-line Certification Authority, Zhou, Schneider & van Renesse)
  – distributed certification authority (special-purpose protocols)

Related work

- View-synchronous group communication (ISIS–Ensemble, Cornell)
  - tolerate only crash failures
- Rampart (Reiter, AT&T Research)
  - failure detector for liveness and safety
- BFT (Castro & Liskov, MIT)
  - failure detector for liveness
- COCA (Cornell On-line Certification Authority, Zhou, Schneider & van Renesse)
  - distributed certification authority (special-purpose protocols)
- SINTRA/MAFTIA
  - randomized agreement and atomic broadcast using cryptography
  - failure detector only for efficiency (with optimistic protocol [KS01])

References

[CKS00] Cachin, Kursawe, Shoup. *Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography*. PODC 2000.

[CKPS01] Cachin, Kursawe, Petzold, Shoup. *Secure and efficient asynchronous broadcast protocols*. CRYPTO 2001 and eprint.iacr.org/2001/006.

[C01] Cachin. *Distributing trust on the Internet*. DSN-2001.

[CP02] Cachin, Poritz. *Secure intrusion-tolerant replication on the Internet*. DSN-2002.

[CKLS02] Cachin, Kursawe, Lysyanskaya, Strobl. *Asynchronous verifiable secret sharing and proactive cryptosystems*. ACM CCS 2002.