BTI 4202: Security and Trust in Distributed Systems

Christian Grothoff

Berner Fachhochschule

28.5.2021

Risk Analysis: Operating a Tor Hidden Service

Learning objectives

- Fallacies of distributed computing
- Boyd's theorem
- CAP Theorem
- Zooko's Triangle
- Self stabilization
- Attacks and defenses
- Distributed Hash Tables
  - CAN
  - Chord
  - Kademlia
- Secure Multiparty Computation
- Secure Multiparty Computation example: Fog of Trust

Part I: Security in Distributed Systems

The 8 Fallacies of Distributed Computing[1]

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology does not change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

[1] According to Peter Deutsch and James Gosling

Limits on authentication

Theorem (Boyd's Theorem I)

"Suppose that a user has either a confidentiality channel to her, or an authentication channel from her, at some state of the system. Then in the previous state of the system such a channel must also exist. By an inductive argument, such a channel exists at all previous states."

Theorem (Boyd's Theorem II)

"Secure communication between any two users may be established by a sequence of secure key transfers if there is a trusted chain from each one to the other."

Solution space: Zfone Authentication (ZRTP) [5]

Idea: combine human interaction proof and baby duck approach:

- A and B perform Diffie-Hellman exchange
- Keying material from previous sessions is used (duckling)
- Short Authentication String (SAS) is generated (hash of DH numbers)
- Both users read the SAS to each other, recognize voice

$\Rightarrow$ ZRTP foils standard man-in-the-middle attack.

CAP Theorem [3]

No distributed system can be *consistent*, *available* and *partition tolerant* at the same time.

- Consistency: A *read* sees the changes made by all previous *writes*
- Availability: *Reads* and *writes* always succeed
- Partition tolerance: The system operates even when network connectivity between components is broken

Blockchain Trilemma

Blockchains claim to achieve three properties:

- Decentralization: there are many participants, and each participant only needs to have a small amount of resources, say $O(c)$
- Scalability: the system scales to $O(n) > O(c)$ transactions
- Security: the system is secure against attackers with $O(n)$ resources

The Blockchain trilemma is that one can only have two of the three.

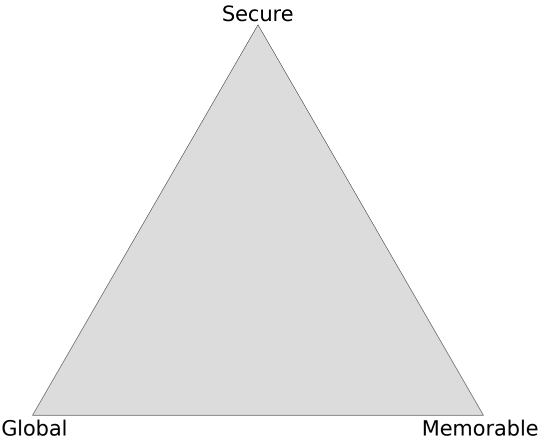
Ryge's Triangle

Ryge's Triangle postulates three key management goals for a system associating cryptographic keys with addresses or names:

- Non-interactive: the system should require no user interface
- Flexible: addresses/names can be re-used by other participants
- Secure: the system is secure against active attackers

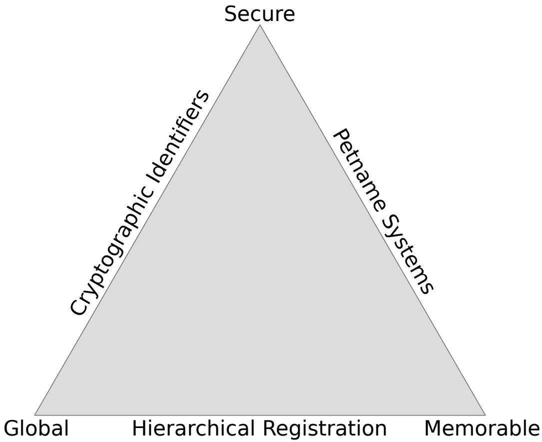
Ryge's triangle says that one can only have two of the three.

Zooko's Triangle



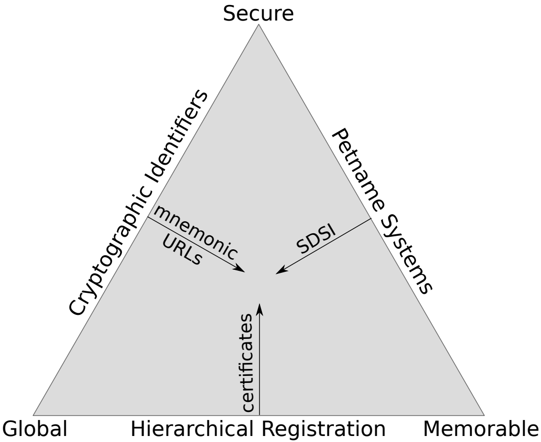
A name system can only fulfill **two**!

Zooko's Triangle



DNS, ".onion" IDs and `/etc/hosts/` are representative designs.

Zooko's Triangle



DNSSEC security is limited (adversary model!)

Self stabilization (Dijkstra 1974)

- A system is self-stabilizing, if starting from any state, it is guaranteed that the system will eventually reach a correct state (convergence).
- Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens (closure).
- Self-stabilization enables a distributed algorithm to recover from a transient fault **regardless of its nature**.

Example: Spanning-tree Protocol from Networking!

Sybil Attack

Background:

- Ancient Greece: Sybils were prophetesses that prophesized under the devine influence of a deity. Note: At the time of prophecy not the person but a god was speaking through the lips of the sybil.
- 1973: Flora Rheta Schreiber published a book “Sybil” about a woman with 16 separate personalities.

Sybil Attack

Background:

- Ancient Greece: Sybils were prophetesses that prophesized under the devine influence of a deity. Note: At the time of prophecy not the person but a god was speaking through the lips of the sybil.
- 1973: Flora Rheta Schreiber published a book "Sybil" about a woman with 16 separate personalities.

The Sybil Attack [2]:

- Insert a node multiple times into a network, each time with a different identity
- Position a node for next step on attack:
  - Attack connectivity of the network
  - Attack replica set
  - In case of majority votes, be the majority.

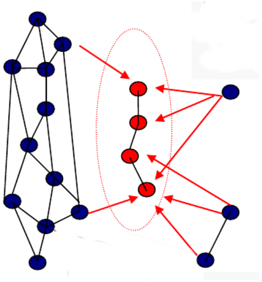
Defenses against Sybil Attacks

- Use authentication with trusted party that limits identity creation
- Use "external" identities (IP address, MAC, e-mail)
- Use "expensive" identities (solve computational puzzles, require payment)

Douceur: Without trusted authority to certify identities, no realistic approach exists to completely stop the Sybil attack.

Eclipse Attack: Goal

- Separate a node or group of nodes from the rest of the network
- isolate peers (DoS, surveillance) or isolate data (censorship)



Eclipse Attack: Techniques

- Use Sybil attack to increase number of malicious nodes
- Take over routing tables, peer discovery

$\Rightarrow$ Details depend on overlay structure

Eclipse Attack: Defenses

- Large number of connections
- Replication
- Diverse neighbour selection (different IP subnets, geographic locations)
- Aggressive discovery ("continuous" bootstrap)
- Audit neighbour behaviour (if possible)
- Prefer long-lived connections / old peers

Poisoning Attacks

Nodes provide false information:

- wrong routing tables
- wrong meta data
- wrong performance measurements

Timing Attacks [4]

Nodes can:

- measure latency to determine origin of data
- delay messages
- send messages using particular timing patterns to aid correlation
- include wrong timestamps (or just have the wrong time set...)

Break

Part II: Distributed Hash Tables

Distributed Hash Tables (DHTs)

- Distributed **index**
- GET and PUT operations like a hash table
- JOIN and LEAVE operations (internal)
- Trade-off between JOIN/LEAVE and GET/PUT costs
- Typically use exact match on cryptographic hash for lookup
- Typically require overlay to establish particular connections

DHTs: Key Properties

To know a DHT, you must know (at least) its:

- routing table structure
- lookup procedure
- join operation process
- leave operation process

... including expected costs (complexity) for each of these operations.

A trivial DHTs: The Clique

- routing table: hash map of all peers
- lookup: forward to closest peer in routing table
- join: ask initial contact for routing table, copy table, introduce us to all other peers, migrate data we're closest to to us
- leave: send local data to remaining closest peer, disconnect from all peers to remove us from their routing tables

Complexity?

A trivial DHTs: The Circle

- routing table: left and right neighbour in cyclic identifier space
- lookup: forward to closest peer (left or right)
- join: lookup own peer identity to find join position, transfer data from neighbour for keys we are closer to
- leave: ask left and rigt neighbor connect directly, transfer data to respective neighbour

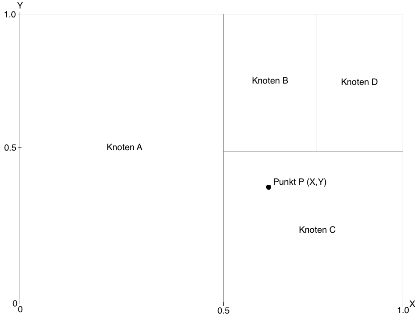
Complexity?

Additional Questions to ask

- Security against Eclipse attack?
- Survivability of DoS attack?
- Maintenance operation cost & required frequency?
- Latency? ($\neq$ number of hops!)
- Data persistence?

Content Addressable Network: CAN

- routing table: neighbours in d-dimensional torus space
- lookup: forward to closest peer
- join: lookup own peer identity to find join position, split quadrant (data areas) with existing peer
- leave: assign quadrant space to neighbour (s)

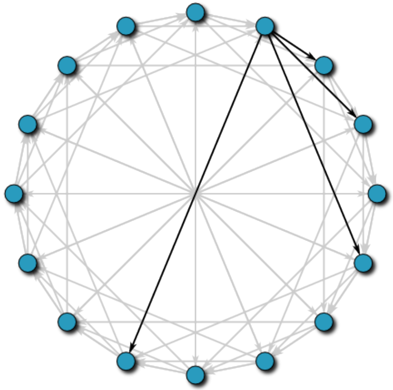


Interesting CAN properties

- CAN can do range queries along $\leq n$ dimensions
- CAN's peers have $2d$ connections (independent of network size)
- CAN routes in $O(d\sqrt[d]{n})$

Chord

- routing table: predecessor in circle and at distance 2^i, plus r successors
- lookup: forward to closest peer (peer ID after key ID)
- join: lookup own peer identity to find join position, use neighbor to establish finger table, migrate data from respective neighbour
- leave: join predecessor with successor, migrate data to respective neighbour, periodic stabilization protocol takes care of finger updates

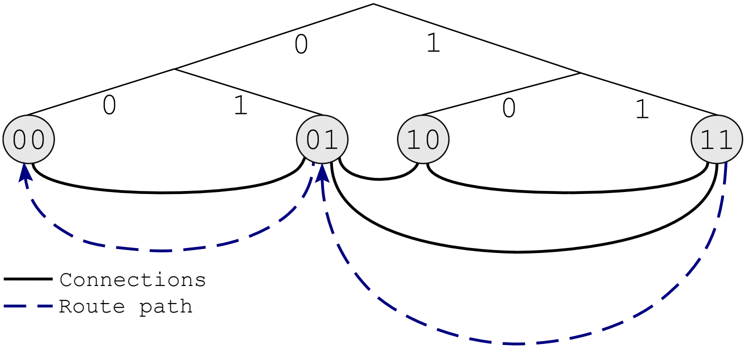


Interesting Chord properties

- Simple design
- $\log_2 n$ routing table size
- $\log_2 n$ lookup cost
- Asymmetric, inflexible routing tables

Kademlia

- routing table: 2^{160} buckets with k peers at XOR distance 2^i
- lookup: iteratively forward to α peers from the "best" bucket, selected by latency
- join: lookup own peer identity, populate table with peers from iteration
- maintenance: when interacting with a peer, add to bucket if not full; if bucket full, check if longest-not-seen peer is live first
- leave: just drop out



Interesting Kademlia properties

- XOR is a symmetric metric: connections are used in both directions
- α replication helps with malicious peers and churn
- Iterative lookup gives initiator much control,
- Lookup helps with routing table maintenance
- Bucket size trade-off between routing speed and table size
- Iterative lookup is a trade-off:
  - good UDP (no connect cost, initiator in control)
  - bad with TCP (very large number of connections)

Break

Part III: Secure Multiparty Computation

Secure Multiparty Computation (SMC)

- Alice und Bob haben private Daten a_i and b_i.
- Alice und Bob führen ein Protokoll aus und berechnen gemeinsam $f(a_i, b_i)$.
- Nur einer von beiden lernt das Ergebnis (i.d.R.)

Adversary models

Honest but curious

Dishonest and curious

Pairing-based cryptography

Let G_1, G_2 be two additive cyclic groups of prime order q, and G_T another cyclic group of order q (written multiplicatively). A pairing is an efficiently computable map e:

$$e : G_1 \times G_2 \rightarrow G_T \quad (1)$$

which satisfies $e \neq 1$ and bilinearity:

$$\forall_{a,b \in F_q^*},\ \forall_{P \in G_1, Q \in G_2} :\ e\left(aP, bQ\right) = e\left(P, Q\right)^{ab} \quad (2)$$

Examples: Weil pairing, Tate pairing.

Hardness assumption

Computational Diffie Hellman:

$$g, g^x, g^y \Rightarrow g^{xy} \qquad (3)$$

remains hard on G even given e.

Boneh-Lynn-Sacham (BLS) signatures [1]

Key generation:

Pick random $x \in \mathbb{Z}_q$

Signing:

$\sigma := h^x$ where $h := H(m)$

Verification:

Given public key g^x:

$$e(\sigma, g) = e(h, g^x) \quad (4)$$

Boneh-Lynn-Sacham (BLS) signatures [1]

Key generation:

Pick random $x \in \mathbb{Z}_q$

Signing:

$\sigma := h^x$ where $h := H(m)$

Verification:

Given public key g^x:

$$e(\sigma, g) = e(h, g^x) \tag{4}$$

Why:

$$e(\sigma, g) = e(h, g)^x = e(h, g^x) \tag{5}$$

due to bilinearity.

Fun with BLS

Given signature $\langle \sigma, g^x \rangle$ on message h, we can *blind* the signature and public key g^x:

$$e(\sigma^b, g) = e(h, g)^{xb} = e(h, g^{xb}) \tag{6}$$

Thus σ^b is a valid signature for the *derived* public key $(g^x)^b$ with blinding value $b \in \mathbb{Z}_q$.

Part IV: Fog of Trust

The Fog of Trust

Problem:

- Publishing who certified whom exposes the social graph.
- The “NSA kills based on meta data”.

The Fog of Trust

Problem:

- Publishing who certified whom exposes the social graph.
- The "NSA kills based on meta data".

Solution:

- Do not publish the graph.
- Have Alice and Bob collect their certificates locally.
- Use SMC protocol for

  private set intersection cardinality with signatures!

We will only consider paths with **one** intermediary.

Straw-man version of protocol 1

Problem: Alice wants to compute $n := |\mathcal{L}_A \cap \mathcal{L}_B|$

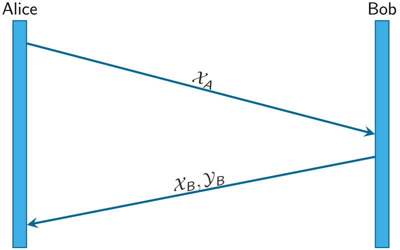
Suppose each user has a private key c_i and the corresponding public key is $C_i := g^{c_i}$ where g is the generator

The setup is as follows:

- $\mathcal{L}_A$: set of public keys representing Alice trusted verifiers
- $\mathcal{L}_B$: set of public keys representing Bob's signers
- Alice picks an ephemeral private scalar $t_A \in \mathbb{F}_p$
- Bob picks an ephemeral private scalar $t_B \in \mathbb{F}_p$

Straw-man version of protocol 1

$$\mathcal{X}_A := \left\{\, C^{t_A} \;\middle|\; C \in \mathcal{L}_A \,\right\}$$



$$\mathcal{X}_B := \left\{\, C^{t_B} \;\middle|\; C \in \mathcal{L}_B \,\right\}$$

$$\begin{aligned}\mathcal{Y}_B &:= \left\{\, \overline{C}^{t_B} \;\middle|\; \overline{C} \in \mathcal{X}_A \,\right\} \\ &= \left\{\, C^{t_B \cdot t_A} \;\middle|\; C \in \mathcal{L}_B \,\right\}\end{aligned}$$

$$\begin{aligned}\mathcal{Y}_A &:= \left\{\, \hat{C}^{t_A} \;\middle|\; \hat{C} \in \mathcal{X}_B \,\right\} \\ &= \left\{\, C^{t_A \cdot t_B} \;\middle|\; C \in \mathcal{L}_A \,\right\}\end{aligned}$$

Alice can get $|\mathcal{Y}_A \cap \mathcal{Y}_B|$ at linear cost.

Attack against the Straw-man

If Bob controls two trusted verifiers $C_1, C_2 \in \mathcal{L}_A$, he can:

- Detect relationship between $C_1^{t_A}$ and $C_2^{t_A}$
- Choose $K \subset \mathbb{F}_p$ and substitute with fakes:

$$\mathcal{X}_B := \bigcup_{k \in K} \left\{ C_1^k \right\}$$

$$\mathcal{Y}_B := \bigcup_{k \in K} \left\{ (C_1^{t_A})^k \right\}$$

so that Alice computes $n = |K|$.

Cut & choose version of protocol 1: Preliminaries

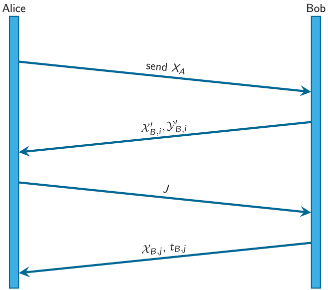
Assume a fixed system security parameter $\kappa \geq 1$.

Let Bob use secrets $t_{B,i}$ for $i \in \{1, \dots, \kappa\}$, and let $\mathcal{X}_{B,i}$ and $\mathcal{Y}_{B,i}$ be blinded sets over the different $t_{B,i}$ as in the straw-man version.

For any list or set Z, define

$$Z' := \{h(x) | x \in Z\} \tag{7}$$

Cut & choose version of protocol 1



Protocol messages:

1. Alice sends:
   $$\mathcal{X}_A := \texttt{sort}\,[\, C^{t_A} \mid C \in \mathcal{A}\,]$$
2. Bob responds with commitments:
   $$\mathcal{X}'_{B,i}, \mathcal{Y}'_{B,i} \quad \textit{for} \;\; i \in 1, \ldots, \kappa$$
3. Alice picks a non-empty random subset $J \subseteq \{1, \ldots, \kappa\}$ and sends it to Bob.
4. Bob replies with $\mathcal{X}_{B,j}$ for $j \in J$, and $t_{B,j}$ for $j \notin J$.

Cut & choose version of protocol 1: Verification

For $j \notin J$, Alice checks the $t_{B,j}$ matches the commitment $\mathcal{Y}'_{B,j}$.

For $j \in J$, she verifies the commitment to $\mathcal{X}_{B,j}$ and computes:

$$\mathcal{Y}_{A,j} := \left\{ \hat{C}^{t_A} \,\middle|\, \hat{C} \in \mathcal{X}_{B,j} \right\} \tag{8}$$

To get the result, Alice computes:

$$n = |\mathcal{Y}'_{A,j} \cap \mathcal{Y}'_{B,j}| \tag{9}$$

Alice checks that the n values for all $j \in J$ agree.

Protocol 2: Private Set Intersection with Subscriber Signatures

- Naturally, signers are willing to *sign* that Bob's key is Bob's key.
- We still want the identities of the signers to be private!
- BLS (Boneh et. al) signatures are compatible with our blinding.

$\Rightarrow$ Integrate them with our cut & choose version of the protocol.

Costs are linear in set size. Unlike prior work this needs no CA.

References I

Dan Boneh, Ben Lynn, and Hovav Shacham.
Short signatures from the weil pairing.
In *Advances in Cryptology – ASIACRYPT '01, LNCS*, pages 514–532. Springer, 2001.

John Douceur.
The Sybil Attack.
In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, March 2002.

Seth Gilbert and Nancy Lynch.
Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.
SIGACT News, 33(2):51–59, June 2002.

References II

Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright.
Timing attacks in low-latency mix-based systems.
In *Proceedings of Financial Cryptography (FC '04)*, pages 251–265, February 2004.

Laurianne McLaughlin.
Philip zimmermann on what's next after pgp.
IEEE Security & Privacy, 4(1):10–13, 2006.