The GNUnet

Decentralizing Privacy-Preserving Network Applications

Christian Grothoff, PhD (UCLA)

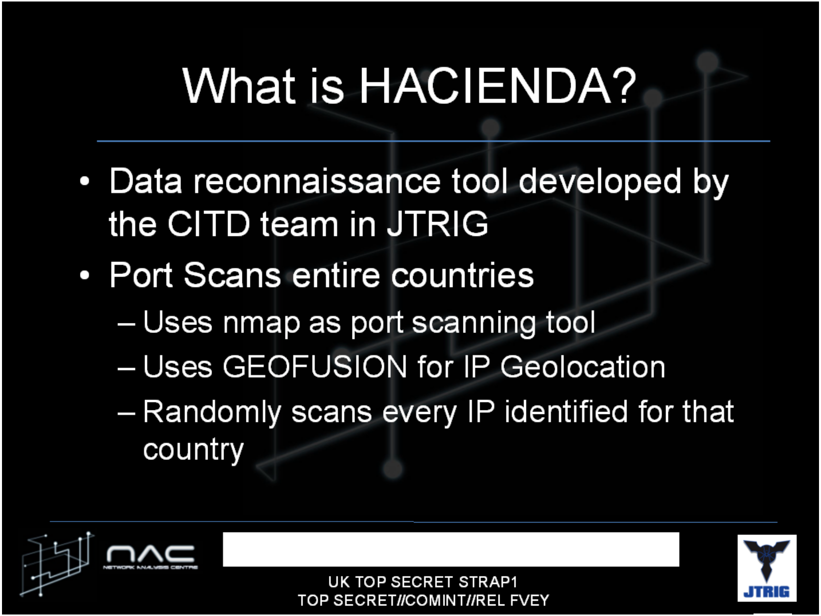
Inria Rennes Bretagne Atlantique

18.10.2017

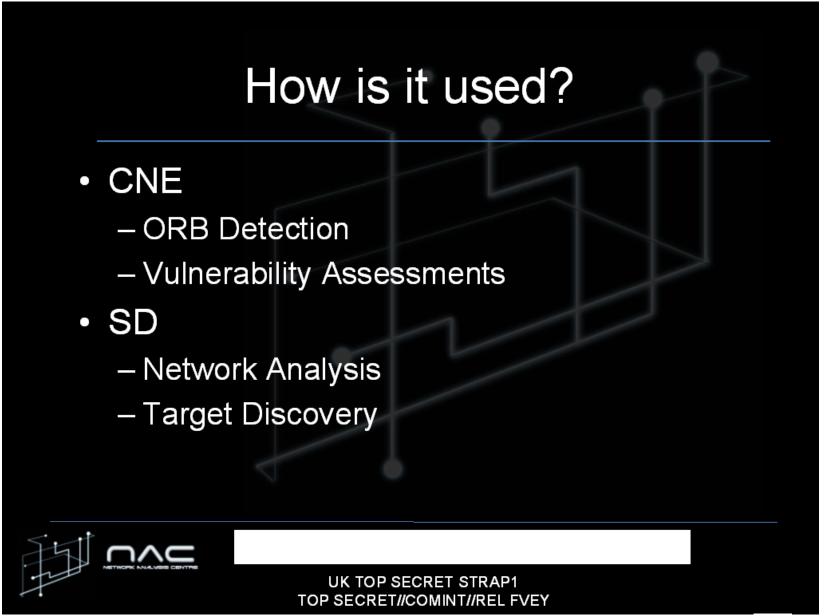
Part I: Motivation

"Never doubt your ability to change the world." –Glenn Greenwald

Example: Information Leak



Example: Information Leak



Example: Collateral Damage

TOP SECRET//COMINT

Communications Security Establishment — Centre de la sécurité des télécommunications

LANDMARK

- CSEC's Operational Relay Box (ORB) covert infrastructure used to provide an additional level of non-attribution; subsequently used for exploits and exfiltration
- 2-3 times/year, 1 day focused effort to acquire as many new ORBs as possible in as many non 5-Eyes countries as possible

Canada

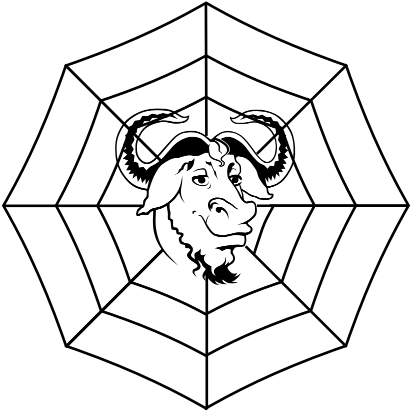
3

They Made Bad Design Choices!

Internet Design Goals (David Clark, 1988)

1. **Internet communication must continue despite loss of networks or gateways.**
2. The Internet must support multiple types of communications service.
3. The Internet architecture must accommodate a variety of networks.
4. The Internet architecture must permit *distributed management* of its resources.
5. The Internet architecture must be cost effective.
6. The Internet architecture must permit host attachment with a low level of effort.
7. **The resources used in the internet architecture must be accountable.**

Let's do something about it!



Design Choices for a Civil Network!

Internet Design Goals (David Clark, 1988)

1. **Internet communication must continue despite loss of networks or gateways.**
2. The Internet must support multiple types of communications service.
3. The Internet architecture must accommodate a variety of networks.
4. The Internet architecture must permit *distributed management* of its resources.
5. The Internet architecture must be cost effective.
6. The Internet architecture must permit host attachment with a low level of effort.
7. **The resources used in the internet architecture must be accountable.**

GNUnet Design Goals

1. GNUnet must be implemented as free software.
2. **The GNUnet must only disclose the minimal amount of information necessary.**
3. **The GNUnet must be decentralised and survive Byzantine failures in any position in the network.**
4. **The GNUnet must make it explicit to the user which entities must be trustworthy when establishing secured communications.**
5. **The GNUnet must use compartmentalization to protect sensitive information.**
6. The GNUnet must be open and permit new peers to join.
7. **The GNUnet must be self-organizing and not depend on administrators.**
8. The GNUnet must support a diverse range of applications and devices.
9. The GNUnet architecture must be cost effective.
10. **The GNUnet must provide incentives for peers to contribute more resources than they consume.**

Let's Implement It!

Internet

Google
DNS/X.509
TCP/UDP
IP/BGP
Ethernet
Phys. Layer

Let's Implement It!

Internet

Internet	
Google	
DNS/X.509	
TCP/UDP	
IP/BGP	
Ethernet	
Phys. Layer	HTTPS/TCP/WLAN/...

Let's Implement It!

Internet	
Google	
DNS/X.509	
TCP/UDP	
IP/BGP	
Ethernet	CORE (OTR)
Phys. Layer	HTTPS/TCP/WLAN/...

Let's Implement It!

Internet	
Google	
DNS/X.509	
TCP/UDP	
IP/BGP	R^5N DHT
Ethernet	CORE (OTR)
Phys. Layer	HTTPS/TCP/WLAN/...

Let's Implement It!

Internet
Google
DNS/X.509
TCP/UDP
IP/BGP
Ethernet
Phys. Layer

CADET (Axolotl+SCTP)
R^5N DHT
CORE (OTR)
HTTPS/TCP/WLAN/...

Let's Implement It!

Internet
Google
DNS/X.509
TCP/UDP
IP/BGP
Ethernet
Phys. Layer

GNU Name System
CADET (Axolotl+SCTP)
R^5N DHT
CORE (OTR)
HTTPS/TCP/WLAN/...

Let's Implement It!

Internet

Google
DNS/X.509
TCP/UDP
IP/BGP
Ethernet
Phys. Layer

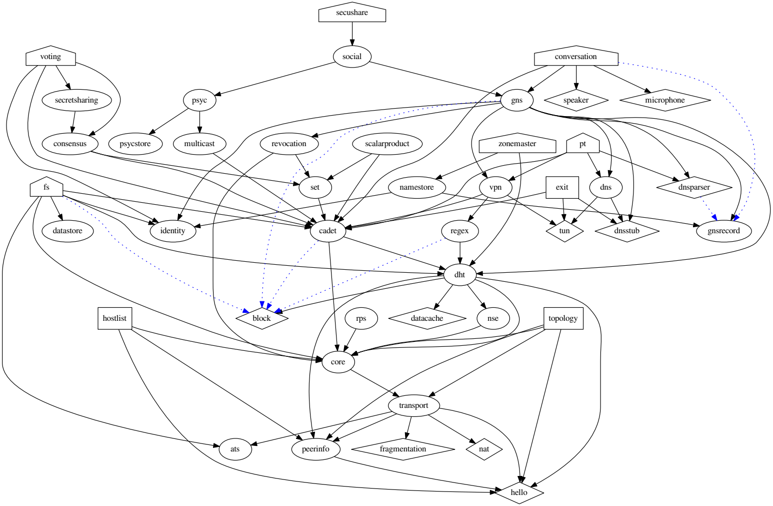
Applications
GNU Name System
CADET (Axolotl+SCTP)
R^5N DHT
CORE (OTR)
HTTPS/TCP/WLAN/...

Let's Implement It!

Internet
Google
DNS/X.509
TCP/UDP
IP/BGP
Ethernet
Phys. Layer

GNUnet
Applications
GNU Name System
CADET (Axolotl+SCTP)
R^5N DHT
CORE (OTR)
HTTPS/TCP/WLAN/...

A real peer: Dependencies



Applications (being) built using GNUnet

- Anonymous and non-anonymous file-sharing
- IPv6–IPv4 protocol translator and tunnel
- GNU Name System: censorship-resistant replacement for DNS
- Conversation: secure, decentralised VoIP
- SecuShare, a social networking application
- GNU Taler: privacy-preserving payments
- ...

Summary

- This is **not** about the NSA
- Chinese, French, German, Russian agencies do the same
- This is about design goals

GNUnet is about designing network protocols to serve civil society.

Part II: The GNU Name System[1]

"The Domain Name System is the Achilles heel of the Web." –Tim Berners-Lee

[1] Joint work with Martin Schanzenbach and Matthias Wachs

The GNU Name System (GNS)

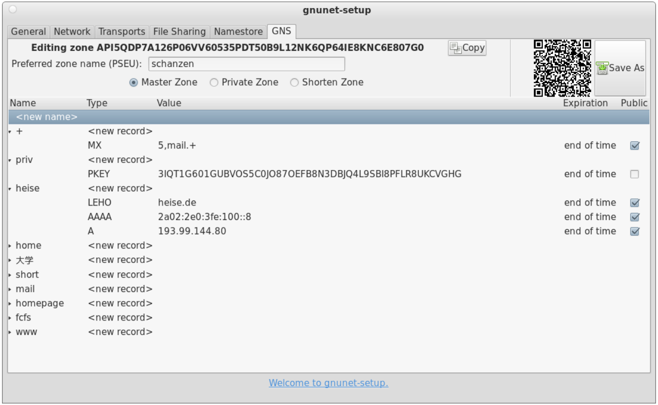
Properties of GNS

- Decentralized name system with secure memorable names
- Delegation used to achieve transitivity
- Also supports globally unique, secure identifiers
- Achieves query and response privacy
- Provides alternative public key infrastructure
- Interoperable with DNS

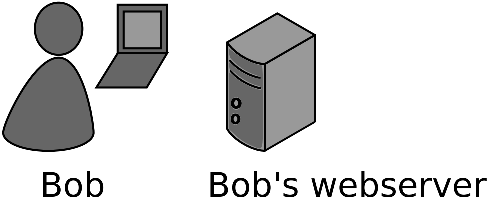
Uses for GNS in GNUnet

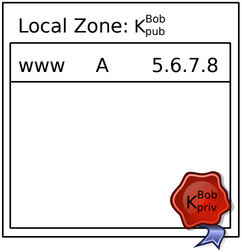
- Identify IP services hosted in the P2P network
- Identities in social networking applications

Zone management: like in DNS



Name resolution in GNS



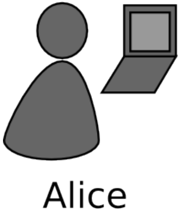


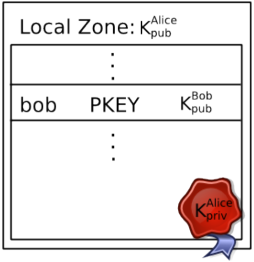
- Bob can locally reach his webserver via **www.gnu**

Secure introduction

- Bob gives his public key to his **friends**, possibly via QR code

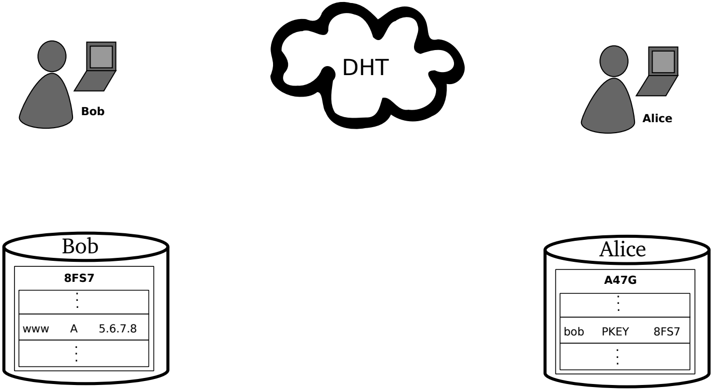
Delegation



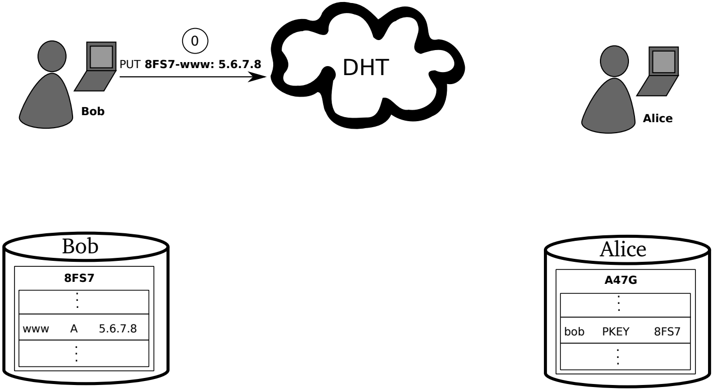


- Alice learns Bob's public key
- Alice creates delegation to zone K_{pub}^{Bob} under label **bob**
- Alice can reach Bob's webserver via **www.bob.gnu**

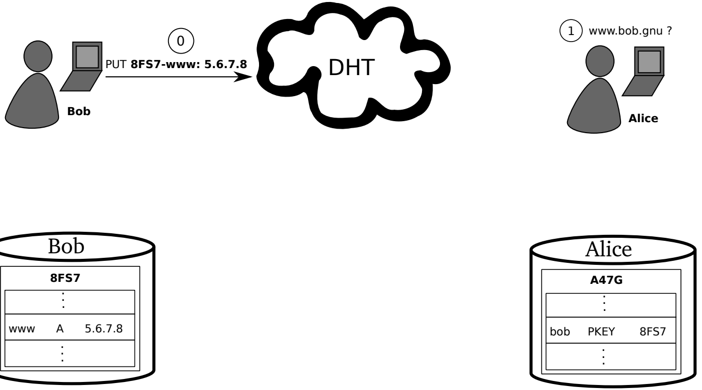
Name resolution



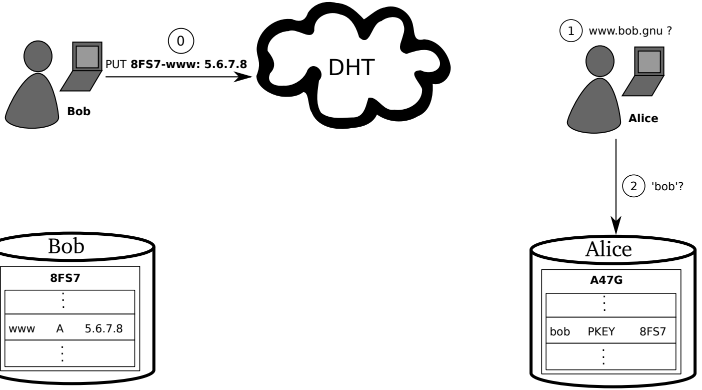
Name resolution



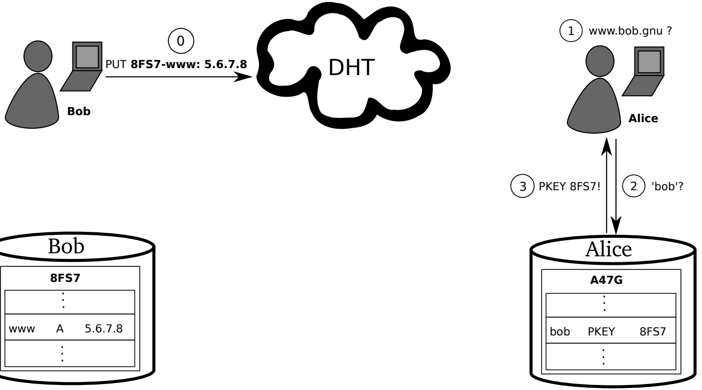
Name resolution



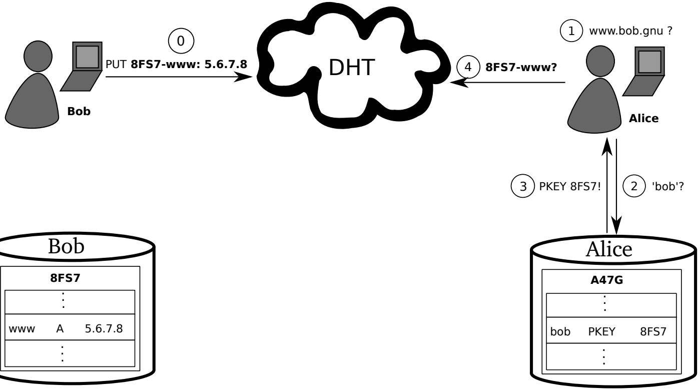
Name resolution



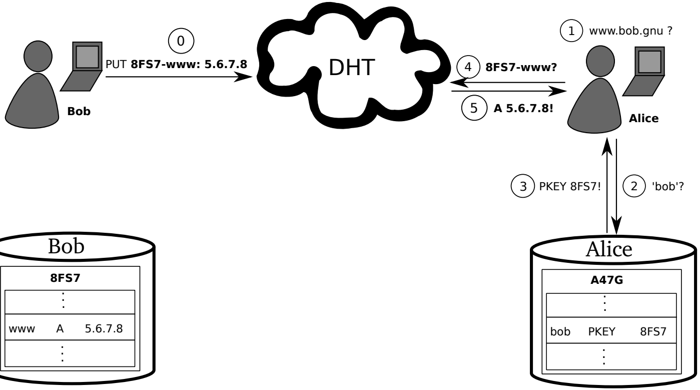
Name resolution



Name resolution

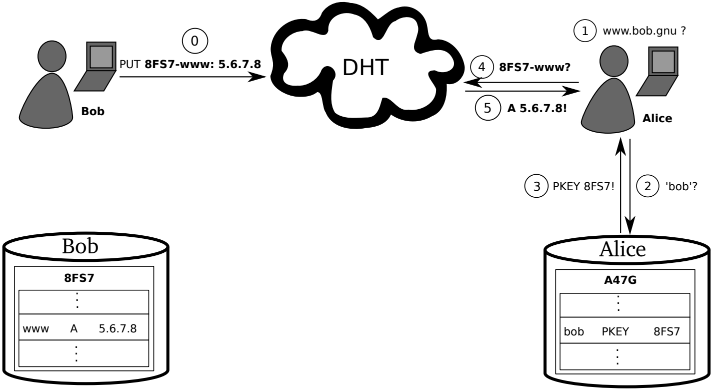


Name resolution



GNS as PKI (via DANE/TLSA)

Privacy issue: DHT



Query privacy: terminology

G	generator in ECC curve, a point
n	size of ECC group, $n := \|G\|$, n prime
x	private ECC key of zone ($x \in \mathbb{Z}_n$)
P	public key of zone, a point $P := xG$
l	label for record in a zone ($l \in \mathbb{Z}_n$)
$R_{P,l}$	set of records for label l in zone P
$q_{P,l}$	query hash (hash code for DHT lookup)
$B_{P,l}$	block with encrypted information for label l in zone P published in the DHT under $q_{P,l}$

Query privacy: cryptography

Publishing records $R_{P,l}$ as $B_{P,l}$ under key $q_{P,l}$

$$
\begin{aligned}
h &:= H(l, P) && (1)\\
d &:= h \cdot x \mod n && (2)\\
B_{P,l} &:= S_d(E_{HKDF(l,P)}(R_{P,l})), dG && (3)\\
q_{P,l} &:= H(dG) && (4)
\end{aligned}
$$

Query privacy: cryptography

Publishing records $R_{P,l}$ as $B_{P,l}$ under key $q_{P,l}$

$$h := H(l, P) \tag{1}$$

$$d := h \cdot x \mod n \tag{2}$$

$$B_{P,l} := S_d(E_{HKDF(l,P)}(R_{P,l})), dG \tag{3}$$

$$q_{P,l} := H(dG) \tag{4}$$

Searching for records under label l in zone P

$$h := H(l, P) \tag{5}$$

$$q_{P,l} := H(hP) = H(hxG) = H(dG) \Rightarrow \texttt{obtain } B_{P,l} \tag{6}$$

$$R_{P,l} = D_{HKDF(l,P)}(B_{P,l}) \tag{7}$$

Key revocation

- Revocation message signed with private key (ECDSA)
- Flooded on all links in P2P overlay, stored forever
- Efficient set reconciliation used when peers connect
- Expensive proof-of-work used to limit DoS-potential
- Proof-of-work can be calculated ahead of time
- Revocation messages can be stored off-line if desired

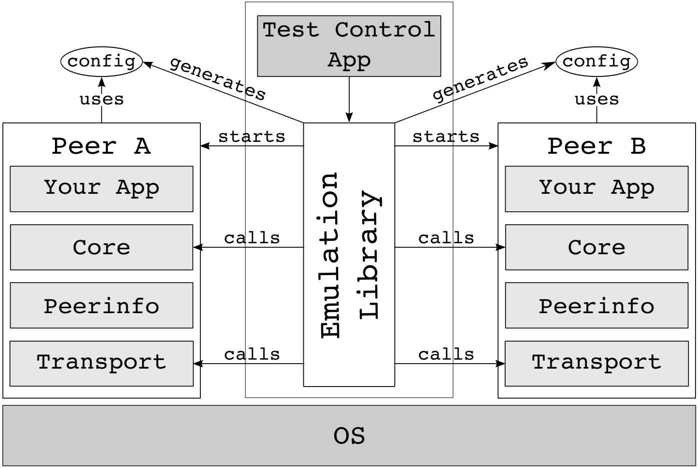
Summary

- Interoperable with DNS
- Delegation allows using zones of other users
- Trust paths explicit, trust agility
- Simplified key exchange compared to Web-of-Trust
- Privacy-enhanced queries, censorship-resistant
- Reliable revocation

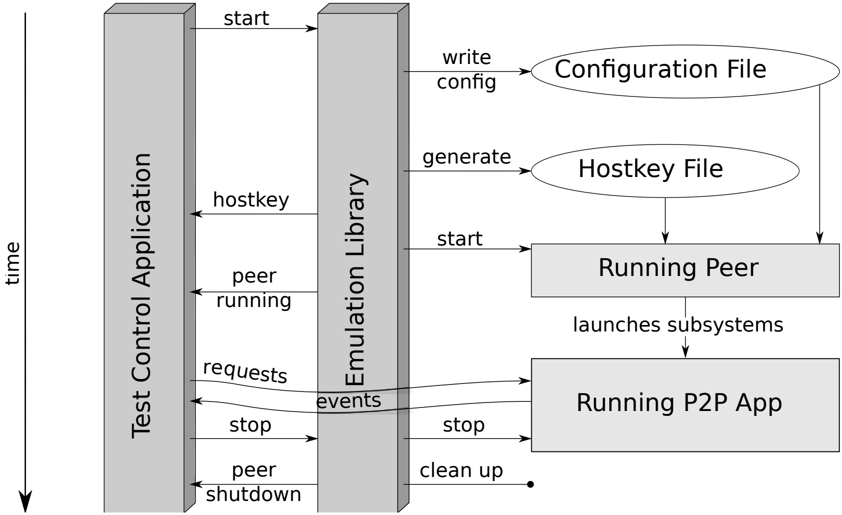
Part III: The GNUnet Testbed[2]

[2] Joint work with Sree Harsha Totakura

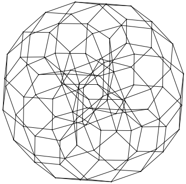
Our Emulation Approach



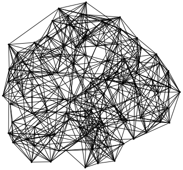
Low Level Emulation Library Usage



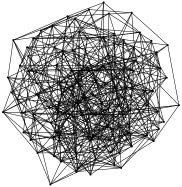
Network Topologies



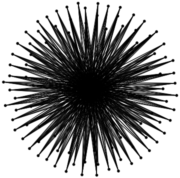
(a) 2d-grid



(b) Small-World



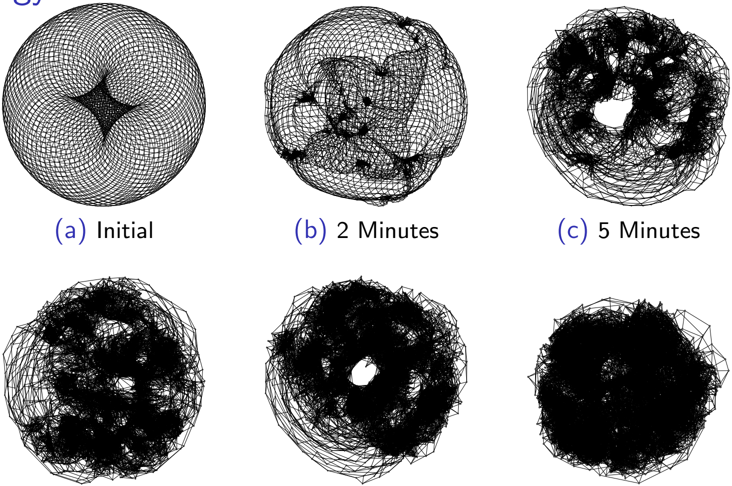
(c) Erdos-Renyi



(d) InterNAT

Figure: Some of the supported topologies.

Topology Generation and Evolution



(a) Initial (b) 2 Minutes (c) 5 Minutes

(d) 10 minutes (e) 15 minutes (f) 30 minutes

Figure: Watching the network evolve.

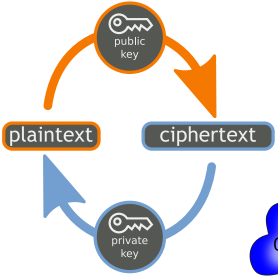
Limitations of Emulation

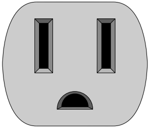
- Timing accuracy
  - Network latency
  - Throughput
- Underlying OS interference
  - CPU scheduling
  - Disk access
  - Memory usage

- Speed
- Shared IP/hostnames
- Peer diversity
- GNUnet

Important Lessons Learned

- Cryptography
- Execution time
- Latency
- Sockets
- Memory



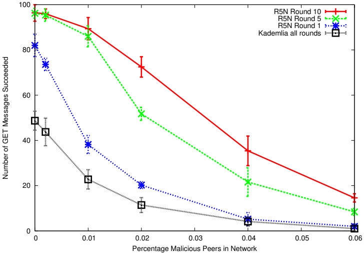


Peer and Emulation Performance

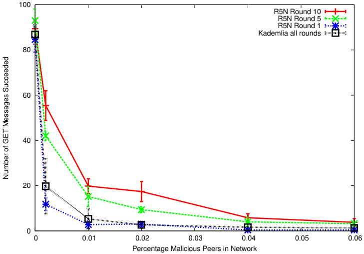
Service	Non-shared	Heap	Shared
supervisor	228 KB	32 KB	2,364 KB
transport	359 KB	99 KB	2,888 KB
core	300 KB	84 KB	2,428 KB
dht	536 KB	240 KB	3,684 KB
total	1,424 KB	456 KB	11,364 KB

Architecture	# Hosts (Total)	Cores (Total)	Memory (Total)	# Peers	Connections per second	Time to start peer
Cortex-A8	1	1	512 MB	100	$\sim$ 1	$\sim$ 206 ms
Xeon W3505	1	2	12 GB	2,025	$\sim$ 60	$\sim$ 12 ms
Xeon W3520	1	8	12 GB	2,025	$\sim$ 188	$\sim$ 5 ms
Opteron 8222	1	16	64 GB	10,000	$\sim$ 327	$\sim$ 27 ms
Opteron 850	31	124	217 GB	80,000	$\sim$ 559	$\sim$ 1 ms

Comparison of DHT Performance



(a) Ideal Small-World topology



(b) 75% NAT Topology

Figure: Example of results obtained from the GNUnet emulation framework.

Part IV: Network Size Estimation[3]

"How many people run GNUnet?" *—Frequently Asked Question*

[3] Joint work with Nathan Evans and Bart Polot

Mental Coin Flips

- Think of **head** or **tail** each round
- Remember the number of the first **mismatch** with the slides!

Flip!

Coin 0

head

Flip!

Coin 1

head

Flip!

Coin 2

tail

Flip!

Coin 3

head

Flip!

Coin 4

head

Flip!

Coin 5

head

Flip!

Coin 6

head

Flip!

Coin 7

tail

Flip!

Coin 8

head

Shout!

Shout *your* number once it appears on the slide!

Number 8

8

Number 7

7

Number 6

6

Number 5

5

Number 4

4

Number 3

3

Number 2

2

Result

Number	Group Size
3	6.35
4	12.70
5	25.41
6	50.82
7	101.63
8	203.27

Thank you for playing

Want to play again?

Motivation

Purpose of Network Size Estimation

- Human curiosity
- Detection of unusual events
- Value of a botnet
- Tuning parameter

Functional Goals

Functional Goals

- All peers obtain the network size estimate
- Supports churn
- Fully decentralized
- Efficient, secure
- Operates in unstructured topologies

Intuitive Idea

- Set of elements distributed in a space
- Pick a random spot
- Measure distance to nearest element
- More elements $\Rightarrow$ smaller distance, more *overlapping*

Intuitive Idea

Intuitive Idea

Intuitive Idea

Intuitive Idea

Intuitive Idea - Applied to networks

- Space: all possible IDs
- Population: randomly distributed peer IDs
- Overlap: number of leading bits in common with a random ID

Theorem

Let $\overline{p}$ be the expected maximum number of leading overlapping bits between all n random node identifiers in the network and a random key. Then the network size n is approximately

$$2^{\overline{p}}$$

- $1 \Rightarrow 2$
- $6 \Rightarrow 64$
- $22 \Rightarrow 4$ M

Theorem

Let $\overline{p}$ be the expected maximum number of leading overlapping bits between all n random node identifiers in the network and a random key. Then the network size n is

$$2^{\overline{p}-0.332747}$$

- $1 \Rightarrow$ 1-2
- $6 \Rightarrow 50$
- $22 \Rightarrow$ 3.3 M

Our Approach: Key Points

- Use the current time to generate a random number
- More overlapping bits $\Rightarrow$ gossip earlier
- Also delay gossip randomly to avoid traffic spikes
- Proof-of-Work to make Sybil attacks harder
- Implemented! (1500 lines C code - GNUnet)

Security

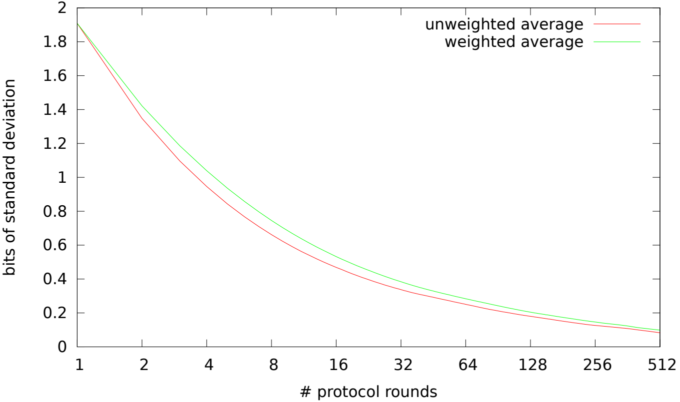
Attacker Model

- Freely participate
- Multiple identities
- May alter, drop, send/receive data
- Same resources as “normal” peers

Security Properties

- Resistant to malicious participants (DoS, Manipulation)
- No trusted third parties
- Reliable

Precision vs. Rounds of Measurement



Compare

	S. & Coll.	Gossip	H. Sampling	This work
MEM	$O(\sqrt{N})$	$O(1)$	$O(\|N\|)$	$O(1)$
CPU	$O(\sqrt{N})$	$O(\|N\|)$	$O(\|E\|)$	$O(\|E\|/\|N\|)$
NET	$O(\|N\|\sqrt{N})$	$O(\|N\|^2)$	$O(\|N\| \cdot \|E\|)$	$O(\|E\|)$
SEC	DoS, BE	DoS, BE	DoS, BE	Pr.-of-Work
IMP	Simulation	Simulation	Simulation	Yes

(BE = Bad Estimates)

Part V: Fog-of-Trust[4]

"PGP assumes keys are too big and complicated to be managed by mortals, but then in practice it practically begs users to handle them anyway."

—Matthew Green

[4] Joint work with Jeffrey Burdges

Motivation

For email: differences of p≡p to other OpenPGP mail clients

- Keyservers are never used by default to prevent leakage of a peer's social graph (by signings and queries) and MITM attacks (re-encyption).
- The sender's public key is attached by default.
- The subject field gets encrypted by default (by moving it into the body).
- Instead of fingerprints, *Trustwords* (16-bit mappings of 4-digit hexablocks to words) are used.
- p≡p has a rating system and communicates (graphically) a *Privacy Status* with traffic lights semantics to the user.

The Web of Trust

Problem:

- Alice has certified many of her contacts and *flagged* some as *trusted* to check keys well.
- Bob has been certified by many of his contacts.
- Alice has **not** yet certified Bob, but wants to securely communicate with him.

The Web of Trust

Problem:

- Alice has certified many of her contacts and *flagged* some as *trusted* to check keys well.
- Bob has been certified by many of his contacts.
- Alice has **not** yet certified Bob, but wants to securely communicate with him.

Solution:

- Find paths in the certification graph from Alice to Bob.
- If sufficient number of short paths exist certifying the same key, trust it.

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

The Web of Trust

Problem:

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

The Web 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!

Straw-man version of protocol 1

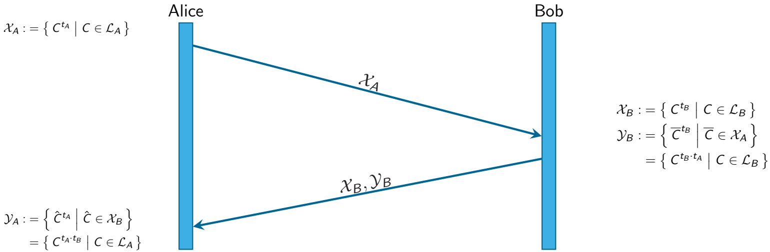
Problem: Alice wants to compute $n := |\mathcal{L}_A \cap \mathcal{L}_B|$

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's subscriptions
- $\mathcal{L}_B$: set of public keys representing Bob's subscriptions
- 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



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

Attack against the Straw-man

If Bob controls two subscribers $C_1, C_2 \in \mathcal{L}_A$, 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

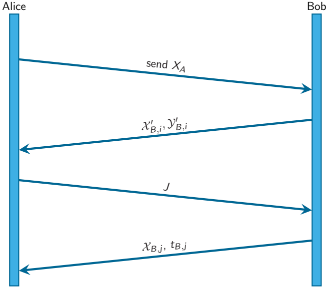
Assume a fixed system security parameter $\kappa \geq 1$.

Let Bob use secrets $t_{B,i}$ for $i \in \{1, \ldots, \kappa\}$, 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

$$Z' := \{h(x) | x \in Z\} \tag{8}$$

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:

$$\mathcal{Y}_{A,j} := \left\{ \hat{C}^{t_A} \;\middle|\; \hat{C} \in \mathcal{X}_{B,j} \right\} \tag{9}$$

To get the result, Alice computes:

$$n = |\mathcal{Y}'_{A,j} \cap \mathcal{Y}'_{B,j}| \tag{10}$$

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

Protocol 2: Private Set Intersection with Subscriber Signatures

- Suppose subscribers are willing to *sign* that they are subscribed.
- We still want the subscriptions to be private!
- BLS (Boneh et. al) signatures are compatible with our blinding.

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

Detailed protocol is in the paper.

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

Part VI: GNU Taler[5]

"I think one of the big things that we need to do, is we need to get a way from true-name payments on the Internet. The credit card payment system is one of the worst things that happened for the user, in terms of being able to divorce their access from their identity."
–Edward Snowden, IETF 93 (2015)

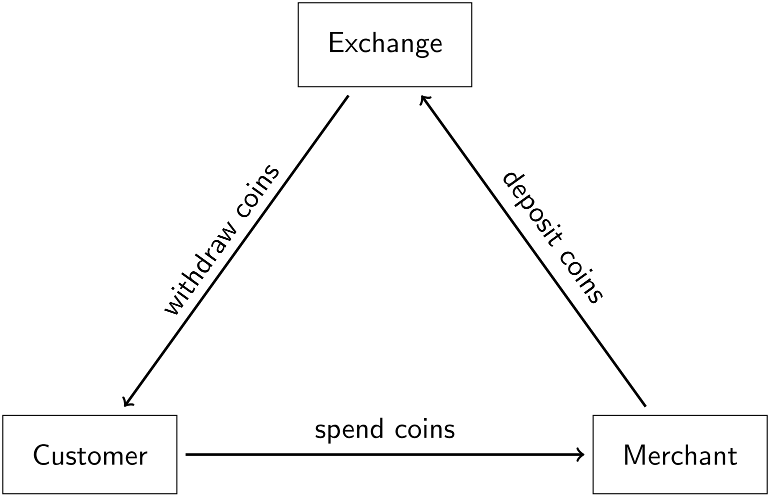
[5] Joint work with Florian Dold, Jeffrey Burdges and others

What is Taler?

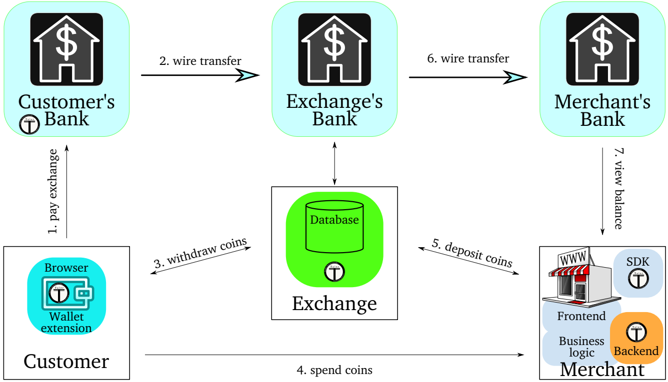
Taler is an electronic instant payment system.

- Uses electronic coins stored in **wallets** on customer's device
- Like **cash**
- Pay in **existing currencies** (i.e. EUR, USD, BTC),
  or use it to create new **regional currencies**

Taler Overview

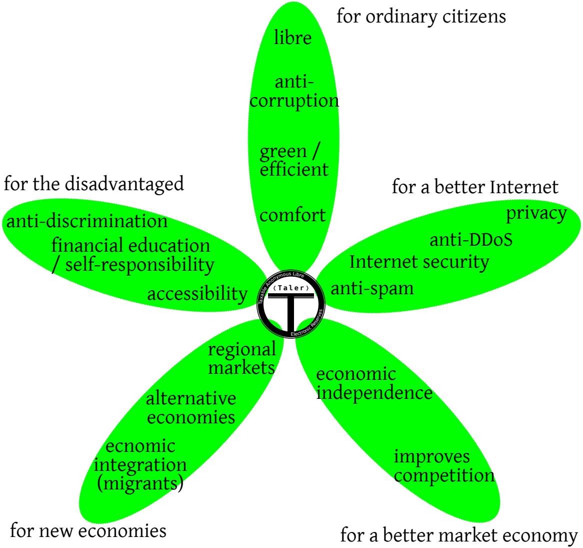


Architecture of Taler



⇒ Convenient, taxable, privacy-enhancing, & resource friendly!

Social Impact of Taler



Taxability

We say Taler is taxable because:

- Merchant's income is visible from deposits.
- Hash of contract is part of deposit data.
- State can trace income and enforce taxation.

Taxability

We say Taler is taxable because:

- Merchant's income is visible from deposits.
- Hash of contract is part of deposit data.
- State can trace income and enforce taxation.

Limitations:

- withdraw loophole
- *sharing* coins among family and friends

How does it work?

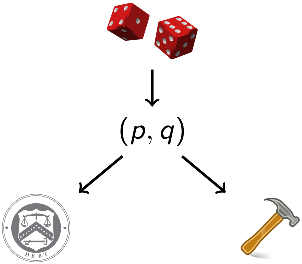
We use a few ancient constructions:

- Cryptographic hash function (1989)
- Blind signature (1983)
- Schnorr signature (1989)
- Diffie-Hellman key exchange (1976)
- Cut-and-choose zero-knowledge proof (1985)

But of course we use modern instantiations.

Exchange setup: Create a denomination key (RSA)

1. Pick random primes p, q.
2. Compute $n := pq$,
   $\phi(n) = (p-1)(q-1)$
3. Pick small $e < \phi(n)$ such that
   $d := e^{-1} \mod \phi(n)$ exists.
4. Publish public key (e, n).



Merchant: Create a signing key (EdDSA)

- pick random $m \mod o$ as private key
- $M = mG$ public key

$\downarrow$

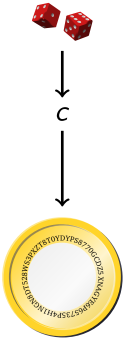
m

$\downarrow$

M

Capability: $m \Rightarrow$ 

Customer: Create a planchet (EdDSA)

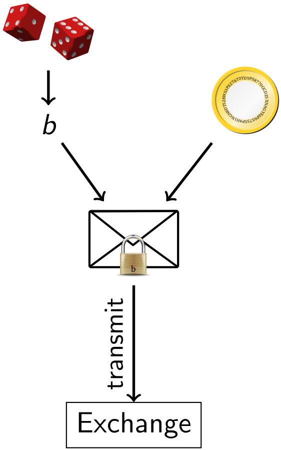


- Pick random $c \mod o$ private key
- $C = cG$ public key

Capability: $c \Rightarrow$ 

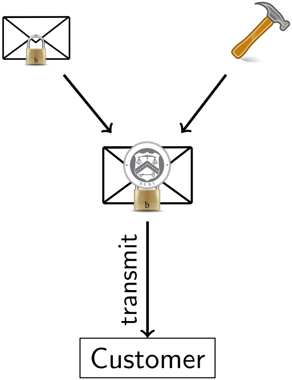
Customer: Blind planchet (RSA)

1. Obtain public key (e, n)
2. Compute $f := FDH(C)$, $f < n$.
3. Pick blinding factor $b \in \mathbb{Z}_n$
4. Transmit $f' := fb^e \mod n$



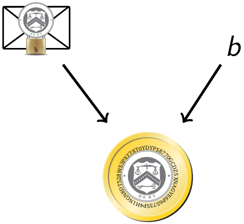
Exchange: Blind sign (RSA)

1. Receive f'.
2. Compute $s' := f'^d \mod n$.
3. Send signature s'.

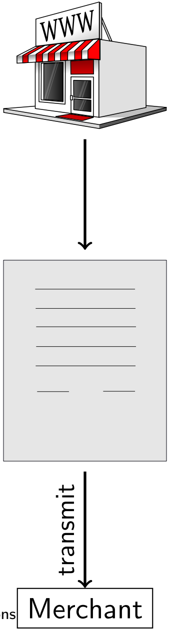


Customer: Unblind coin (RSA)

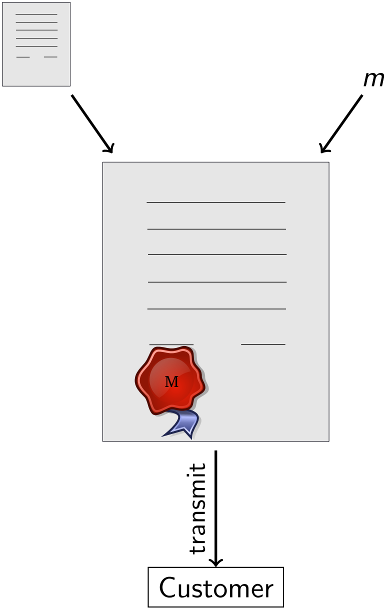
1. Receive s'.
2. Compute $s := s'b^{-1} \mod n$



Customer: Build shopping cart



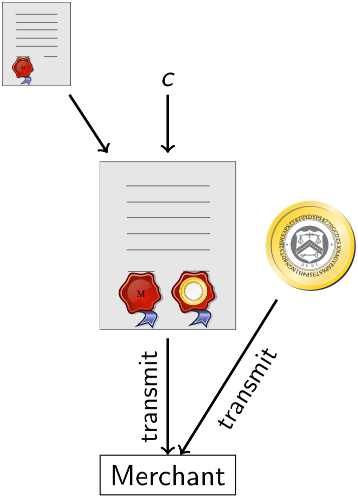
Merchant: Propose contract (EdDSA)



1. Complete proposal D.
2. Send D, $EdDSA_m(D)$

Customer: Spend coin (EdDSA)

1. Receive proposal D, $EdDSA_m(D)$.
2. Send s, C, $EdDSA_c(D)$



Merchant and Exchange: Verify coin (RSA)

$$s^e \stackrel{?}{\equiv} m \mod n$$

Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Key goals:

- maintain unlinkability
- maintain taxability of transactions

Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Key goals:

- maintain unlinkability
- maintain taxability of transactions

Method:

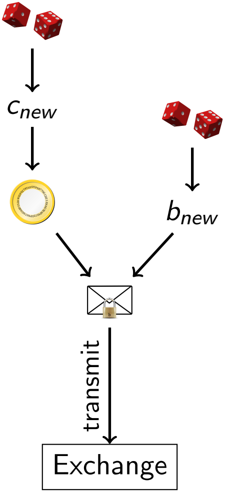
- Contract can specify to only pay *partial value* of a coin.
- Exchange allows wallet to obtain *unlinkable change* for remaining coin value.

Strawman solution

Given partially spent private coin key c_{old}:

1. Pick random $c_{new} \mod o$ private key
2. $C_{new} = c_{new} G$ public key
3. Pick random b_{new}
4. Compute $f_{new} := FDH(C_{new})$, $m < n$.
5. Transmit $f'_{new} := f_{new} b^e_{new} \mod n$

... and sign request for change with c_{old}.

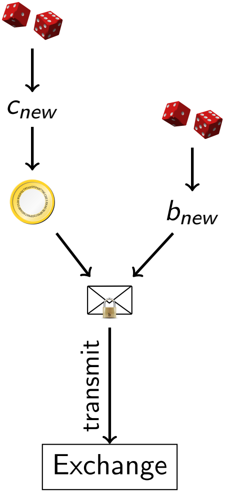


Strawman solution

Given partially spent private coin key c_{old}:

1. Pick random $c_{new} \mod o$ private key
2. $C_{new} = c_{new} G$ public key
3. Pick random b_{new}
4. Compute $f_{new} := FDH(C_{new})$, $m < n$.
5. Transmit $f'_{new} := f_{new} b^e_{new} \mod n$

... and sign request for change with c_{old}.

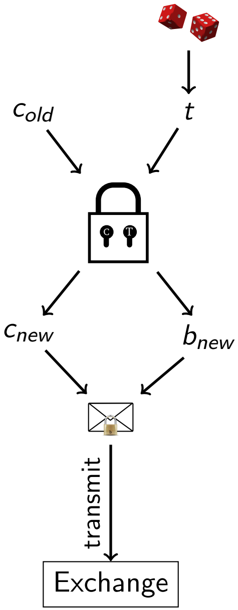


Problem: Owner of c_{new} may differ from owner of c_{old}!

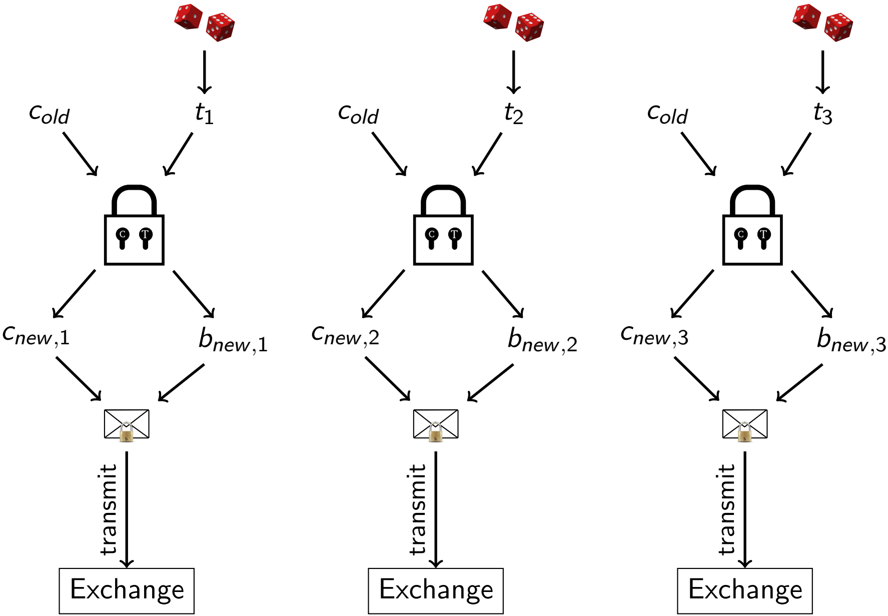
Customer: Transfer key setup (ECDH)

Given partially spent private coin key c_{old}:

1. Let $C_{old} := c_{old} G$ (as before)
2. Create random private transfer key $t \mod o$
3. Compute $T := tG$
4. Compute $X := c_{old}(tG) = t(c_{old} G) = tC_{old}$
5. Derive c_{new} and b_{new} from X
6. Compute $C_{new} := c_{new} G$
7. Compute $f_{new} := FDH(C_{new})$
8. Transmit $f'_{new} := f_{new} b^e_{new}$



Cut-and-Choose



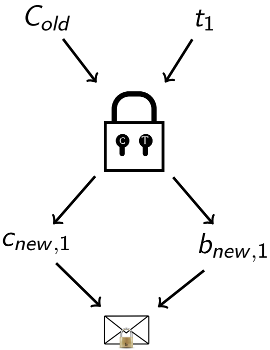
Exchange: Choose!

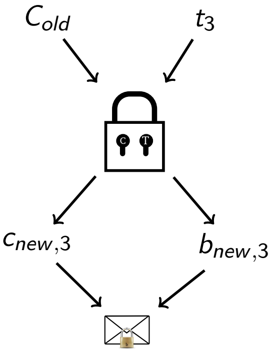
Exchange sends back random $\gamma \in \{1, 2, 3\}$ to the customer.

Customer: Reveal

1. If $\gamma = 1$, send t_2, t_3 to exchange
2. If $\gamma = 2$, send t_1, t_3 to exchange
3. If $\gamma = 3$, send t_1, t_2 to exchange

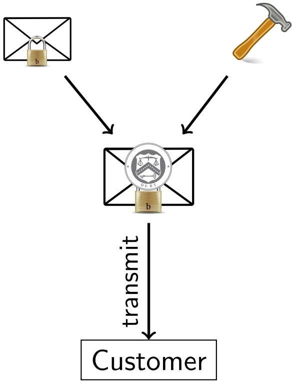
Exchange: Verify ($\gamma = 2$)





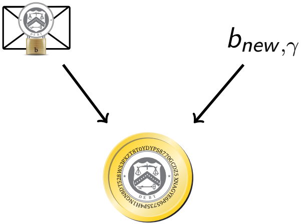
Exchange: Blind sign change (RSA)

1. Take $f'_{new,\gamma}$.
2. Compute $s' := f'^{d}_{new,\gamma} \mod n$.
3. Send signature s'.



Customer: Unblind change (RSA)

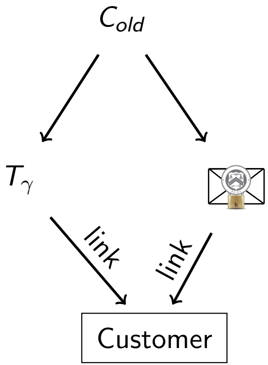
1. Receive s'.
2. Compute $s := s' b_{new,\gamma}^{-1} \mod n$.



Exchange: Allow linking change

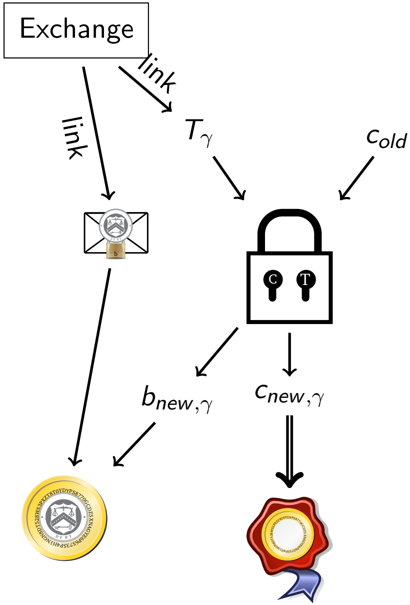
Given C_{old}

return T_γ, $s := s' b_{new,\gamma}^{-1} \mod n$.



Customer: Link (threat!)

1. Have c_{old}.
2. Obtain T_γ, s from exchange
3. Compute $X_\gamma = c_{old} T_\gamma$
4. Derive $c_{new,\gamma}$ and $b_{new,\gamma}$ from X_γ
5. Unblind $s := s' b_{new,\gamma}^{-1} \mod n$



Refresh protocol summary

- Customer asks exchange to convert old coin to new coin
- Protocol ensures new coins can be recovered from old coin

$\Rightarrow$ New coins are owned by the same entity!

Thus, the refresh protocol allows:

- To give unlinkable change.
- To give refunds to an anonymous customer.
- To expire old keys and migrate coins to new ones.
- To handle protocol aborts.

Transactions via refresh are equivalent to sharing a wallet.

Competitor comparison

	Cash	Bitcoin	Zerocoin	Creditcard	GNU Taler
Online	−−−	++	++	+	+++
Offline	+++	−−	−−	+	−−
Trans. cost	+	−−−	−−−	−	++
Speed	+	−−−	−−−	o	++
Taxation	−	−−	−−−	+++	+++
Payer-anon	++	o	++	−−−	+++
Payee-anon	++	o	++	−−−	−−−
Security	−	o	o	−−	++
Conversion	+++	−−−	−−−	+++	+++
Libre	−	+++	+++	− − −	+++

Part VII: Summary

"You Broke the Internet! Let's build ourselves a GNU one!" –Carlo von Lynx/Klaus Schleisiek

Scientific Contributions

- New design goals for a global network
- Specific alternative network architecture
- Many new interesting network protocols
- Concrete implementation artifact
- Various experimental evaluations
- Spin-off projects (libmicrohttpd, libextractor) in production use, or in preparation (GNU Taler)

FAQ

- What is the adversary model for GNUnet?
- When will you release GNUnet 1.0?
- How scalable is the DHT?
- How do you plan for people to install GNUnet?
- Does Tor not solve the problem?
- How does GNUnet compare to Freenet/I2P/RetroShare/net2o?
- Isn't all this crypto going to make it too slow?
- Will terrorists use GNUnet?
- What if your host computer is compromised?
- What is your favourite NSA slide?

Conclusion

There is hope!

Further reading

1. Christian Grothoff, Bart Polot and Carlo von Loesch. *The Internet is broken: Idealistic Ideas for Building a GNU Network*. **W3C/IAB Workshop on Strengthening the Internet Against Pervasive Monitoring (STRINT)**, 2014.
2. Nathan Evans and Christian Grothoff. *R^5N. Randomized Recursive Routing for Restricted-Route Networks*. **5th International Conference on Network and System Security**, 2011.
3. Matthias Wachs, Martin Schanzenbach and Christian Grothoff. *A Censorship-Resistant, Privacy-Enhancing and Fully Decentralized Name System*. **13th International Conference on Cryptology and Network Security**, 2014.
4. M. Schanzenbach *Design and Implementation of a Censorship Resistant and Fully Decentralized Name System*. **Master's Thesis (TUM)**, 2012.
5. Álvaro García-Recuero, Jeffrey Burdges and Christian Grothoff. *Privacy-Preserving Abuse Detection in Future Decentralised Online Social Networks.* **Data Privacy Management (DPM)**, pages 78–93, 2016.

Example: Trusting Trust

[edit] (S//NF) Strawhorse: Attacking the MacOS and iOS Software Development Kit

(S) Presenter: [redacted], Sandia National Laboratories

(S//NF) Ken Thompson's gcc attack (described in his 1984 Turing award acceptance speech) motivates the StrawMan work: what can be done of benefit to the US Intelligence Community (IC) if one can make an arbitrary modification to a system compiler or Software Development Kit (SDK)? A (whacked) SDK can provide a subtle injection vector onto standalone developer networks, or it can modify any binary compiled by that SDK. In the past, we have watermarked binaries for attribution, used binaries as an exfiltration mechanism, and inserted Trojans into compiled binaries.

(S//NF) In this talk, we discuss our explorations of the Xcode (4.1) SDK. Xcode is used to compile MacOS X applications and kernel extensions as well as iOS applications. We describe how we use (our whacked) Xcode to do the following things: -Entice all MacOS applications to create a remote backdoor on execution -Modify a dynamic dependency of securityd to load our own library - which rewrites securityd so that no prompt appears when exporting a developer's private key -Embed the developer's private key in all iOS applications -Force all iOS applications to send embedded data to a listening post -Convince all (new) kernel extensions to disable ASLR

(S//NF) We also describe how we modified both the MacOS X updater to install an extra kernel extension (a keylogger) and the Xcode installer to include our SDK whacks.

Example: Pwning your Enemies