Anon-Pass: Practical Anonymous Subscriptions

Michael Z. Lee†, Alan M. Dunn†, Jonathan Katz*, Brent Waters†, Emmett Witchel†

† University of Texas at Austin
* University of Maryland

May 20, 2013
Let’s build a service

Sharing Resistance
(admission control)
They are collecting information about you.
Anonymous Media

Accesses can’t be correlated

Song 1

1234...

time

Unlinkability

Song 2

8720...
Linked accesses could deanonymize users

Access patterns for enough time could help deanonymize clients

The Netflix Prize dataset
[Narayanan, Shmatikov 2008]

Social networks
[Narayanan, Shmatikov 2009]
But even if tokens are unlinkable...

We assume clients are using a network anonymity service
Anonymous
Music Service
Straw Man

♫♪♩♬
♫♪♩♬
♬♪♫♩
♫♪♩♬
♫♪♬♩

1234...
8720...
7964...
8739...
1910...
2372...

Unlinkability
but not sharing resistance
How do we get both?

Unlinkable Serial Transactions [Syverson et al. 1997]
Sharing resistance, unlinkability
but needs unbounded storage

Anonymous Blacklisting Systems [Tsang et al. 2008]
Sharing resistance, unlinkability
but computationally expensive
And also be practical?

Unlinkable Serial Transactions [Syverson et al. 1997]
- Sharing resistance, unlinkability
- but needs unbounded storage

Anonymous Blacklisting Systems [Tsang et al. 2008]
- Sharing resistance, unlinkability
- but computationally expensive

Anon-Pass
- Sharing resistance, unlinkability, and efficiency
Example: over 12,000 concurrent clients
How?

How is Anon-Pass built?

How is Anon-Pass used?

How does Anon-Pass perform?
How is it built?

Split up time into **epochs**
Each user has a **unique** token for an epoch

Each epoch allows a new, **unpredictable** token
How is it built?

Split up time into **epochs**
Each user has a **unique** token for an epoch

Each epoch allows a new, **unpredictable** token
Use a **pseudorandom function** (PRF)
High Level Protocols

Register
Get a blinded signature on a secret

Login
Prove the token used the signed secret (in zero knowledge)
Anonymous
Music Service

Song 1

$\text{PRF}_{(t)}$

1234...

time

$t-1$  $t$  $t+1$  $t+2$
Anonymous
Music Service

---

Song 1

Song 2

PRF\(_{(t)}\)

PRF\(_{(t+2)}\)
Anonymous
Music Service

But songs don’t always fit in one epoch

PRF \((t)\)

1234...
Anonymous
Music Service

But songs don’t always fit in one epoch

PRF_{(t+1)}[1234...5629...]

song song
Anonymous Music Service

But songs don’t always fit in one epoch

\[ PRF(t+2) \]
Anonymous Music Service

But songs don’t always fit in one epoch
And these accesses are implicitly linked

Conditional Linkability
Accesses can be implicitly linked

The service knows when the same song is repeatedly accessed

Client is implicitly linked while accessing the same media

And unlinkability costs the service provider (and therefore harms the system)
Re-Up

Our way of getting conditional linkability

Prove the current token and the next token are linked

**Trades** unlinkability for efficiency

But the client already lost unlinkability while accessing the same media
Re-Up is more efficient

Login proves you should be allowed access
Re-Up proves you logged in before

Login takes 10 expensive operations
Re-Up takes only 2
Using Login and Re-Up

A client must Login to start a new song
And Re-Up to continue playing the same song

To be unlinkable again, the client must wait until the next epoch
Epoch Lengths: Long vs. Short

A **short** epoch means less time to be unlinkable
And less delay between client actions
Happy Clients

A **long** epoch means fewer client requests
And lower server load
Happy Server

Choosing an epoch length depends on the service
(e.g., 15 seconds for music, 5 minutes for movies)
Re-Up helps balance this tension

Short epochs means less wait between unlinkable actions

Re-Up instead of Login reduces server load
And Anon-Pass is formally proven

Formal proof of security holds under the DDHI assumption

Formal proof of soundness holds under the LRSW assumption

Stated and proved in the paper
How is Anon-Pass built?

How is Anon-Pass used?

How does Anon-Pass perform?
How could it be used?

Anonymous Music Streaming
  Music download over normal HTTP
  15 second epoch
Unlimited-use Subway Pass
  NYC’s “unlimited” pass
  6 minute epoch
Account Proxy
  Multiplex accounts to news sites
  1 minute epoch
System Architecture

my laptop

Client Application

subscription service

Application Server
System Architecture

my laptop

subscription service
System Architecture

my laptop

Authentication Server

User Agent

subscription service
System Architecture

my laptop

User Agent

Gateway

Authentication Server

subscription service

3rd party service
User Agent

Purpose: minimize changes to client applications

Job: Create Login and Re-Up requests
Keep the user secret secure

Modified VLC to anonymously stream (54 LoC)
No modifications to support browsers
Authentication Server

Purpose: enforce sharing resistance

Job: Verify tokens and token uniqueness
    Record active tokens

Runs on the service or as a 3rd party
Gateway

Purpose: enforce access control with minimal change to existing services

Job: Prevent unauthorized access and responses
Remove verification from the critical path

Runs on the service as a front end server
How?

How is Anon-Pass built?

How is Anon-Pass used?

How does Anon-Pass perform?
Evaluation Environment

quad-core 2.66 GHz Intel Core 2 CPU
8GB RAM
1 Gbps network

10 client machine to evaluate the streaming music service

An HTC Evo 3D to evaluate the anonymous subway pass
Crypto Cost

Graph showing:
- Login: 8 milliseconds
- Re-up: 1.25 milliseconds

7.8x Faster

Color codes:
- Blue: Other
- Orange: Verify
Music Service Scaling

HTTP server to stream music
15 second epoch

Add clients until we run out of resources

Used 10 client machines
Music Service Scaling

Login Only vs. Anon-Pass

- 8,000 Clients
- 12,000 Clients

% CPU

Time

Login Only
Anon-Pass
Problem: Need to rate limit between swipes
A long epoch can simulate that timeout
But sharing is still possible…
Anonymous Subway Pass

Solution: Login and Re-Up at the same time
Accesses during later epochs are linkable
Anonymous Subway Pass

Implemented as an Android application

Clients Login and Re-Up twice (18 minute NYC policy)

Takes only 0.2 seconds (on an HTC Evo 3D)
Anon-Pass

Practical – efficient enough to scale

Flexible – works with different services

Deployable – minimizes service changes