Anon-Pass: Practical Anonymous Subscriptions

Michael Z. Lee and Alan M. Dunn | University of Texas at Austin
Jonathan Katz | University of Maryland
Brent Waters and Emmett Witchel | University of Texas at Austin

Anon-Pass, a protocol and system for subscription services, lets users authenticate anonymously while preventing mass sharing of credentials. Service providers can't correlate users' actions but are guaranteed that each account is in use at most once at a given time.

Electronic subscriptions are widespread and quickly becoming the dominant mode of access for services like music and video streaming, news, and academic articles. Although electronic subscriptions are convenient for users, they reveal a lot of information, ranging from personal preferences to geographic movements. Many users want electronic services, but they also want privacy. Simply anonymizing data doesn't always protect users' privacy. Indeed, multiple anonymized datasets released for research purposes, including the AOL search dataset[1] and the Netflix Prize dataset,[2] have been partially deanonymized through correlation or by understanding of the semantics of the released data.

It's difficult to create systems that protect user privacy and simultaneously control admission—that is, keeping out users who haven't paid. Foregoing one of these two goals makes achieving the other considerably easier. If users are required to log in to an account, foregoing anonymity, a service can enforce that no user is logged in twice simultaneously. On the other hand, a subscription system using a single shared identity for all users prevents user identification (although traffic anonymization via a system like Tor[3] might be required to mask network-level identifiers). However, subscribing users could share the secret with nonsubscribers.

Ideally, we would have an anonymous subscription system that protects the interests of both the service and the users. At a high level, such a service needs two operations: registration and login. Registration lets users sign up for the service, at which point they might need to provide identifying details, such as a credit card number or public key. Login lets registered users access protected resources via their subscription. Informally, an anonymous subscription service ensures that users' logins aren't linked to the information they provided at registration and login sessions aren't linkable with one another.

Although cryptographic protocols for providing anonymous credentials services already exist,[4] there are problems with putting these protocols into practice (for more information, see the "Related Work in Anonymous Credentials" sidebar). Many of these protocols are designed for thousands of concurrent users; however, Netflix streamed 1 billion hours of content in July 2012 and has millions of subscribers.[5] At this scale, some proposed cryptographic operations require

Related Work in Anonymous Credentials

Our work continues research into anonymous credentials, which allow admission control while maintaining anonymity.[1]

Handling credential abuse has been a central theme of much anonymous credentials work; however, credential abuse takes on a different meaning in many of the different systems. Early work focused on e-cash, where credentials represented units of currency.[2,3] The key task was to prevent double-spending. However, currency-based systems are use limited and don't translate to unlimited subscription services.

One of the earliest proposals for anonymous subscription services was unlinkable serial transactions.[4] The system ensures a user can have only one valid credential at a time by recording every previously seen credential and issuing a new anonymous credential at the end of each transaction. Instead of requiring the potentially unbounded storage cost of unlinkable serial transactions, Anon-Pass has users periodically contact the service.

More recent work focused on anonymous blacklisting systems wherein services can blacklist users.[5] Anonymous blacklisting systems usually let services reveal some form of linking information to prevent future access. Anon-Pass links a user only when the user explicitly tells the service to, reducing the cryptographic cost.

One way to implement an anonymous subscription service is by blacklisting users at login and removing them at logout. However, many of these schemes suffer from poor scalability. Indeed, authentication for BLAC has overhead proportional to the number of blacklisted users.[6] A more recent system, BLACR, measures scalability in terms of authentications per minute using 5,000 concurrent users. In contrast, Anon-Pass can sustain almost 500 login operations per second and scales to 12,000 clients concurrently streaming music.

References

1. D. Chaum, "Blind Signatures for Untraceable Payments," *Advances in Cryptology* (CRYPTO 82), 1982, pp. 199–203.
2. I. Damgård, "Payment Systems and Credential Mechanisms with Provable Security against Abuse by Individuals," *Advances in Cryptology* (CRYPTO 88), 1988, pp. 328–335.
3. D. Chaum, "Security without Identification: Transaction Systems to Make Big Brother Obsolete," *Comm. ACM*, 1985, pp. 1030–1044.
4. S.G. Stubblebine, P.F. Syverson, and D.M. Goldschlag, "Unlinkable Serial Transactions: Protocols and Applications," *ACM Trans. Information and System Security*, 1999, pp. 354–389.
5. R. Henry and I. Goldberg, "Formalizing Anonymous Blacklisting Systems," *Proc. 32nd IEEE Symp. Security and Privacy*, 2011, pp. 81–95.
6. P.P. Tsang et al., "Blacklistable Anonymous Credentials: Blocking Misbehaving Users without TTPs," *Proc. 14th ACM Conf. Computer and Communications Security*, 2007, pp. 72–81.

too much computation. Existing work doesn't focus on realistic evaluation scenarios, making it difficult to understand what performance issues would arise in a deployed system. Anon-Pass, a new protocol for anonymous subscription services, achieves significant improvements in efficiency over prior protocols.

Anonymous Subscriptions with Conditional Linkage

How do we achieve both anonymity and admission control? Ideally, we want each new client operation to appear to be from a new user, unrelated to previous users. (Note that, in this article, *user* denotes the person, and *client* denotes the program or machine performing actions.) However, when two operations are authorized with the same client secret at the same time, it must be clear they're from the same user. To keep login credentials verifiable but also make them changeable, we divide time into equal-length intervals or *epochs*, agreed on by both clients and servers. Clients use the epoch as input to a pseudorandom function (PRF), which allows them to change the login credential for each epoch. Changing login credentials means each client appears to be a new user in every new epoch, but each client secret can create only one unique login credential per epoch, preventing multiple simultaneous logins with the same credentials. A login credential provides access only for the duration of an epoch.

There's a tension between a service provider's desire for a long epoch (to reduce server load) and users' desire for a short epoch (to improve anonymity). The service needs to perform cryptographic checks during login, making login a computationally expensive operation. Consequently, the service provider wants to maximize epoch length. However, users are unlinked from previous activity only once an epoch boundary has passed and hence prefer a shorter epoch. For example, when listening to a music streaming service, users probably don't want to wait five minutes—or even one minute—for the next track to play.

We believe that users want unlinkability across accesses to distinct pieces of content, such as a movie, song, and news article, but not all accesses to protected resources need to be unlinkable. For instance, whereas users want to decorrelate their watching *The Godfather* from listening to "Teenage Dream," they don't need to decorrelate watching the first minute of *The Godfather* from watching the second minute. Therefore, we

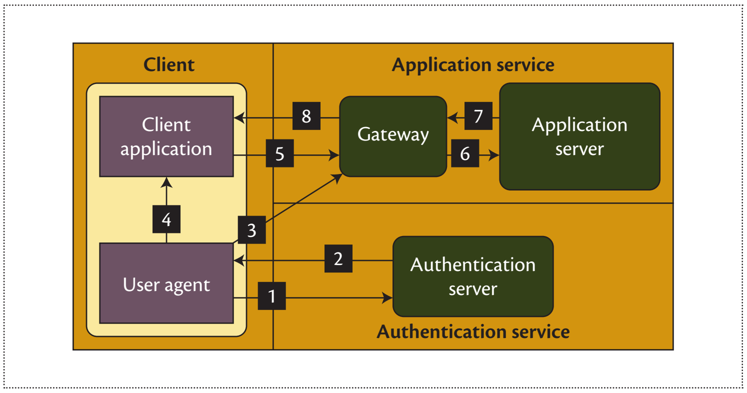

Figure 1. The communication between the authentication server, resource gateway, and client user agent with respect to the client and the service. (1) The user agent initiates communication. (2) The authentication server verifies the credentials and returns a sign-in token to the user agent. (3) The user agent communicates this sign-in token to the resource gateway, then (4) passes this information to the client application. (5) The client application includes the token as a cookie along with its normal request. (6) The gateway checks that the sign-in token hasn't already been used in the current epoch and then proxies the connection to the application server. (7) The application server returns the requested content, and (8) the gateway verifies that the connection is still valid before returning the response to the client.

designed Anon-Pass with conditional linkage, wherein some accesses to protected resources are linked to reduce computational cost when privacy is less important. We provide short epochs, giving users the ability to reanonymize quickly if they so choose, while also providing an efficient method for those who don't need unlinkability to cheaply reauthenticate for the next epoch.

Syntax and Intended Usage

An anonymous subscription scheme with conditional linkage consists of two algorithms—Setup and EndEpoch—and three protocols—Reg, Login, and Re-up. Setup and EndEpoch are used for bookkeeping of internal service state and let us clearly define service provider operations in our proofs. The three protocols comprise the scheme's primary functionality: registering new users in the system (Reg) and authenticating clients to the system (Login and Re-up). Re-up differs from normal authentication because it requires that the client is already logged in to the system. The protocol implements our scheme's conditional linkage aspect by extending the current user session into the next epoch.

System initialization begins with the server running Setup(n) (where n is the security parameter) to generate the service public and secret keys. Following setup, clients can register new users at any time; we denote the secret key of client i as sk_i. Independent of user registrations (which don't affect the server's state and can be performed at any time), there's a sequence of Login, Re-up, and EndEpoch executions. The time period between two executions of EndEpoch, or between Setup and the first execution of EndEpoch, is an epoch. Login$_i$ (respectively, Re-up$_i$) denotes an execution of Login (respectively, Re-up) between the ith client and the server, with both parties using their prescribed inputs.

During an epoch, we (recursively) define that client i is logged in if either Login$_i$ was previously run during that epoch or, at some point in the previous epoch, client i was logged in and Re-up$_i$ was run. Client i is linked if at some previous point during that epoch, client i was logged in and Re-up$_i$ was run.

Security

We define two notions of security: *soundness*, which ensures that malicious users can't generate more active logins than the number of times they've registered, and *anonymity*, which guarantees unlinkability for clients that authenticate using the Login protocol. However, clients that reauthenticate using the Re-Up protocol will be linked to their actions in the previous epoch.

A scheme is sound if, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following experiment is negligible: $\mathcal{A}$ is a malicious user trying to authenticate without using a valid client secret it knows. $\mathcal{A}$ can perform several actions against the service, including registering polynomially many malicious users (and hence learning the associated secret), forcing honest clients to log in and re-up, and globally incrementing the current epoch. $\mathcal{A}$ succeeds if, at any point, the number of logged-in clients is greater than the number of $\mathcal{A}$'s users plus the number of honest clients currently logged in.

A scheme is anonymous if, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following experiment is negligibly close to 1/2: $\mathcal{A}$ is a malicious service trying to link users' access patterns. During setup, a random bit c is chosen, $\mathcal{A}$ sets the service public key, and two clients—U_0 and U_1—are registered. $\mathcal{A}$ then proceeds in three phases where, in each phase, $\mathcal{A}$ may take any number of actions among those it's allowed (while still remaining polynomial time):

- *Phase one.* $\mathcal{A}$ may increment the epoch; query the oracle Login(b), which performs a login for U_b; and query the oracle Re-up(b), which performs a re-up for U_b. In essence, $\mathcal{A}$ has full knowledge and control of the access pattern for U_0 and U_1. If at some point both U_0 and U_1 aren't currently logged in, then $\mathcal{A}$ can proceed to phase two.
- *Phase two.* $\mathcal{A}$ may perform the same operations, but queries ChallengeLogin(b) and ChallengeRe-up(b)

instead. ChallengeLogin(b) is the same as Login$(b \oplus c)$, and ChallengeRe-up(b) is the same as Re-up$(b \oplus c)$. The second phase ends when neither client is logged in at the beginning of an epoch.

- *Phase three.* $\mathcal{A}$ may perform the same operations as during phase one: increment the epoch, query the oracle Login(b), and query the oracle Re-up(b).

At any point, $\mathcal{A}$ can output a bit c'. $\mathcal{A}$ succeeds if $c = c'$.

Design

Anon-Pass is intended to instantiate our protocol in a way that's practical for deployment. We present a conceptual framework for the system in which the various system functionalities are separated.

There are three major pieces of Anon-Pass functionality: the *client user agent*, the *authentication server*, and the *resource gateway*. The client user agent and the authentication server correspond to the client and server in the cryptographic protocol. The resource gateway enforces admission to the underlying service, denying access to users who aren't properly authenticated. An Anon-Pass *session* is a sequence of epochs beginning when a user logs in and ending when the user stops re-upping.

Figure 1 shows the Anon-Pass system's major components. We depict the most distributed setting, wherein each of the three functions is implemented separately from existing services. However, a deployment might merge functionality—for example, the resource gateway might be folded into an already existing component for session management.

Our system supports internal and external authentication servers. An internal authentication server corresponds to a service provider offering anonymous access, for instance, *The New York Times* website offering anonymous access at a premium. An external authentication server corresponds to an entity providing anonymous access to already existing Web services, for example, a commercial Web proxy, such as proxify.com or zend2.com, offering anonymous services.

Although not depicted in Figure 1, our system implements registration. We don't discuss the registration protocol's payment portion. Anonymous payment is a separate and orthogonal problem; possible solutions include paying with e-cash or Bitcoins.

We want services to use our authentication scheme without much modification, so we provide a simple interface: during a certain time period, authorized clients can contact the service and are cut off as soon as the session is no longer valid. Services might have to accommodate Anon-Pass's access control limitations. For example, a media-streaming service might want to limit how much data any client can buffer in a given epoch. The service provider loses the ability to enforce any access control for buffered data.

Timing

Anon-Pass requires time synchronization between clients and servers because both must agree on epoch boundaries. To support a 15-second epoch, clients and servers should be synchronized within one second. The Network Time Protocol (NTP) is sufficient, available, and scalable for this task. The `pool.ntp.org` organization (www.ntp.org/ntpfaq/NTP-s-algo.htm) runs a pool of NTP servers that keep the clocks of 5 to 15 million machines on the Internet synchronized to within approximately 100 ms.

The server's response to a login request includes a time stamp. Clients verify that they agree with the server on the current epoch. Client anonymity could be violated if the epoch number decreases, so clients must track the latest time stamp from each server they use and refuse to authenticate to a server that returns a time stamp that's earlier than a prior time stamp from that server. This ensures that, regardless of any time difference between server and client, anonymity is preserved.

Clients who re-up choose a random time during the epoch to send the re-up request to prevent repetitive behavior that becomes identifying. Randomizing the re-up request time also has the benefit of spreading the computational load of re-ups on the server across the entire epoch.

Client User Agent

The client user agent is responsible for establishing user secrets, communicating with the authentication server, and maintaining client sessions. Separating it from the client application achieves two goals: it minimizes the amount of code that users must trust to handle their secrets, and it lowers the amount of modification necessary to support new client applications.

Once the user agent establishes a connection with the authentication server, it runs our Login protocol and receives a sign-in token in the form of a standard, public-key signature on the user's PRF value and the current epoch. The user agent sends this token to the resource gateway as proof that it's authenticated for the current epoch. The resource gateway uses the epoch to ensure timely use, and the signature to determine token validity. The user agent can't use this token in a later epoch.

When the user agent and authentication server run our Re-up protocol, the user agent receives a token that includes the current and next epoch as well as the two corresponding PRF values. These additional values allow the resource gateway to link the re-up operations to the session's initial login request.

Authentication Server

We separated the authentication server from the service to offer service providers greater flexibility. The server's primary task is to run the authentication protocols and ensure that clients aren't authenticating more than once per epoch. Because the protocol's cryptographic operations use a lot of computational resources, we designed Anon-Pass so that an authentication service provider can distribute the work among multiple machines. The only information that needs to be shared between processes is the current epoch and PRF values of all logged-in users, for example, using a distributed hash table. Storing only information about currently authenticated users relieves service providers of having to store all spent tokens, which requires unbounded storage.

Resource Gateway

The resource gateway performs a lightweight access check before sending data back to a client. A client can receive data during an epoch only if it's authenticated for that epoch. Therefore, the epoch length bounds how much data can go to a client before the client must reauthenticate (log in or re-up).

Construction

Here, we provide an overview of a cryptographic construction for a secure anonymous subscription scheme with conditional linkage that allows us to formalize and prove Anon-Pass properties (see "Anon-Pass: Practical Anonymous Subscriptions" for details[6]). Our construction uses several primitives—bilinear groups, zero-knowledge proofs of knowledge, a PRF family, and cryptographic assumptions from prior work. In our formal model (unlike the implementation), we assume protocols aren't executed concurrently, so there's a well-defined ordering among those events.

Similar to Jan Camenisch and his colleagues' scheme,[7] our construction works by associating a unique token $Y_d(t)$ with each client secret d in each epoch t. To register, a client obtains a blind signature from the service on a secret of its choosing. To log in, the client sends a token and proves in zero knowledge that it knows the service's signature on the secret and that this secret was used to compute to the token that was sent. The token is used to determine admission to the service; the service accepts the token only if it hasn't been presented before in that epoch. Intuitively, the difficulty of generating signatures ensures soundness, and the tokens' pseudorandomness ensures anonymity.

We use the Dodis-Yampolskiy PRF[8] and an adapted version of CL signatures proposed by Camenisch and Anna Lysyanskaya.[9] These building blocks are efficient and enable efficient zero-knowledge proofs as needed for our construction.

A client can authenticate during epoch t by sending the token $Y_d(t)$ and proving in zero knowledge that the token is correct. However, if the client is already logged in during epoch $t - 1$, it can authenticate by sending $Y_d(t)$ and proving that $Y_d(t - 1)$ uses the same client secret. This method is much more efficient, with the tradeoff that the two user sessions are now explicitly linked. In an epoch in which the client isn't logged in, it can perform a fresh login to reanonymize itself.

Implementation

We implemented the cryptographic protocol in the `libanonpass` library using the Pairing Based Cryptographic Library,[10] PolarSSL (https://polarssl.org) for clients, and OpenSSL (www.openssl.org) for the server. To show our protocol's flexibility, we implemented several usage scenarios including a music-streaming service and an anonymous unlimited-use public transit pass. These applications were large enough to highlight implementation issues specific to each context.

We implemented the authentication server and resource gateway as two separate Nginx modules (www.nginx.org). The design didn't need to share much state—the only state Anon-Pass needed to track was the current set of active login tokens. Both the authentication server and the resource gateway needed to track this information; however, this was consolidated to a single distributed hash table as both were run by the same service. The authentication server performed the cryptographic operations to try to keep expensive computations out of users' critical data path. Instead, the resource gateway only needed to verify a standard Elliptic Curve Digital Standard Algorithm (ECDSA) signature and verify and update the table of active sessions.

The basic client was a wrapper around `libanonpass` and PolarSSL, which provided an encrypted connection. Protocol messages were sent using cookies to simplify server-side parsing and minimize client application modifications.

Music-Streaming Service

We implemented a music-streaming service over HTTPS by exposing media from Web-accessible URIs. The service directly implemented our anonymous credential scheme and let users choose an anonymous session's granularity as either a full playlist or an individual song. We modified VLC (www.videolan.org/vlc), a popular media player, to communicate with our user agent and pass our session tokens as cookies to the resource gateway.

Our music service let users download songs, but we rate-limited playback. Rate limiting reduces network bandwidth usage, which allowed our service to support more clients with jitter-free service. Rate limiting also reduces the amount of data a client can buffer during an epoch. If a client loses its anonymous service in the next epoch, it would have only a small amount of buffered data. Our music service couldn't enforce access control for that buffered data.

Public Transit Pass

We implemented a public transit pass as an Android application. Currently, public transit providers that issue month- or weeklong unlimited-access passes limit user access to prevent cheating. Without safeguards, users could hand off their pass to their friends. Anonymous subscriptions can provide these safeguards without revealing users' identities (so their movements can't be tracked).

We used the Java Native Interface to call into `libanonpass` from an Android application. The Android application had a simple interface with a single button to generate a login and two additional re-up tokens. It then displayed this data as a QR code for a physical scanner to read. The login and re-up tokens prevent use of the client for the remainder of the current epoch and the next two epochs. If the transit provider chose a six-minute epoch, this would create a 12- to 18-minute period in which a login attempt from the same phone would fail. Although this guarantee isn't precisely the same guarantee an unlimited-ride transit pass currently provides, it does present an alternative that gives riders anonymity while still enforcing a lockout period.

Other anonymous subscription systems, such as unlinkable serial transactions (UST),[11] and anonymous blacklisting systems, such as Nymble[12] and BLAC,[13] require network connectivity when clients use authentication tokens. When using a blacklisting system, users want to proactively fetch the blacklist to ensure that they aren't on the list prior to contacting a server; otherwise, they could be deanonymized. Blacklists can grow quickly; for example, BLAC adds 0.27 Kbytes of overhead per blacklist entry. When using a UST-like system, users must receive the next token when a prior token is used up (but not before). Anon-Pass is ideal for subway systems where network phone coverage is spotty at best, because it needs to communicate only in one direction at the entry gate.

However, there is a caveat. Although Anon-Pass might be able to simulate the lockout period, a transit provider must implement an unlimited-use pass, which doesn't map directly to the model enforced today. This is because, by definition, Anon-Pass can't prevent timesharing over a longer period of time. Once the 12- to 18-minute period is up, another user could use a duplicated pass, but the original rider wouldn't have necessarily exited the system. The requirement of one physical device allowing access to one person is broken because the credential could be copied in an untraceable manner.

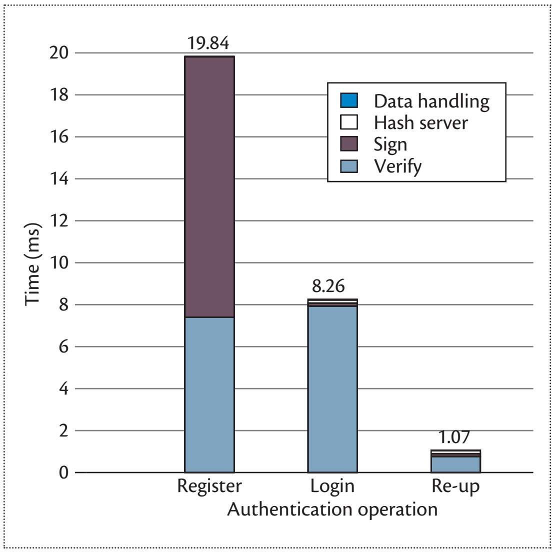

Figure 2. The average cost of different requests on an unsaturated server. The bulk of the time is spent on signature verification.

Evaluation

We evaluated several Anon-Pass applications using a series of microbenchmarks. Further results, including a theoretical cost comparison to prior work and an additional example service, can be found in "Anon-Pass: Practical Anonymous Subscriptions."[6]

Measured Operation Costs

There are overheads when integrating the protocols into a full system. Figure 2 shows a breakdown of each authentication operation and how time was spent on the server. For Reg, the signature operation was our modified CL signature on the blinded client secret, whereas Login and Re-up used standard ECDSA signatures. The majority of the work for the ECDSA signature could be precomputed and hence took almost no time to compute. Re-up was 7.7 times faster than login.

Music-Streaming Service

We built an example music-streaming service; however, we lack datacenter-level resources, so we adapted the benchmark to run on our local cluster of machines. Our cluster's main constraints were the limited network bandwidth (1 Gbps) and memory available to run clients. Each client randomly chose a song and fetched it using `pyCurl` rather than a more memory-intensive media player like VLC. Avoiding VLC let us scale to a greater number of clients for our testbed.

We served a media library consisting of 406 MP3 files, drawn from the most popular 500 songs on the Grooveshark music service, eliminating duplicates and songs longer than 11 minutes. The average song length

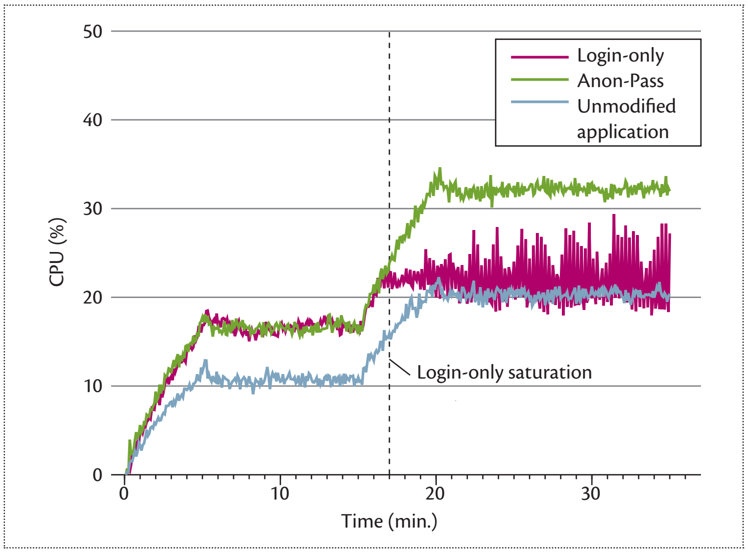

Figure 3. CPU usage on the application server, measured every five seconds. CPU usage with login-only follows the Anon-Pass behavior until the authentication server reaches saturation. The music-streaming clients time out, and the application server has an overall drop in CPU utilization due to the lower number of clients successfully completing requests.

was 4:05 ± 64.38 seconds. We represented the music files using white noise encoded at 32 Kbps. The system dynamics were independent of the music content; 32 Kbps let our server saturate its CPU before saturating its outbound network bandwidth.

We simulated three different scenarios: a baseline system without any authentication, a login-only system in which users couldn't reauthenticate cheaply, and the full Anon-Pass system including re-up. When authentication was involved, we used an epoch length of 15 seconds. The scenario began with 6,000 clients gradually logging in over a period of five minutes. After a song finished, the client unlinked itself and chose a new song to stream. After 10 more minutes, we had an additional 6,000 clients log in, also over a five-minute period. We couldn't scale the experiment further because we exhausted the resources that could be devoted to additional clients.

Figure 3 shows the CPU utilization on the application server sampled once every five seconds. Anon-Pass used more CPU resources than the baseline application because the service had to perform an additional ECDSA verification once per epoch. Until approximately 17 minutes into the experiment, the login-only service was very similar to Anon-Pass. However, at 17 minutes, the login-only service's utilization graph drops and is much more variable.

To see why this happens, we look at a graph of the CPU utilization for authentication, which shows the limited capacity of the login-only service server (see Figure 4). At 6,000 clients, the login-only service could keep up with authentication requests. However, the steady-state average CPU utilization was already 77.9 percent. At the CPU saturation point, 8,100 clients were attempting to connect to the service. When a user couldn't reauthenticate, the music playback was cut off, and the client was forced to retry. With 12,000 clients attempting to stream music concurrently, the login-only configuration had a client failure rate of 34 percent, compared to only 0.02 percent when using Re-up.

Public Transit Pass

To evaluate the public transit pass scenario, we used the Android application and computed the time it took to generate the login QR code on a commodity phone. The login QR code consisted of a normal client login and two additional Re-up tokens. The time to generate a login QR code on an HTC Evo 3D was 222 ± 24 ms. Power usage on this platform was minimal because the application didn't need access to any radios on the phone.

On our server, the combined login and token verification cost 8.4 ms of CPU time, most of which was the cost of verifying the login. Putting this into perspective, in 2013, Bay Area Rapid Transit had approximately 385,600 riders per day on weekdays.[14] We don't have data on traffic peaks; however, Anon-Pass can easily handle the total load. One modern CPU core on our server can perform the approximately 400,000 verifications in just under an hour. These operations are trivially parallelizable across multiple cores and machines.

Anon-Pass is a building block for anonymous authentication and can be used in a range of applications. We've made our source code publicly available at https://github.com/ut-osa/anon-pass in the hopes of spurring further developments in this field. Anon-Pass demonstrates that it's possible to balance the tension between client flexibility and service load through the use of a lighter-weight reauthentication operation for some usage models. However, work remains to convince services to provide unlinkability for their users. ■

Acknowledgments

We thank Sangman Kim and Lara Schmidt for their kind help. We also thank our shepherd, Paul Syverson, and the anonymous reviewers for their useful feedback. This research was supported by funding from National Science Foundation grants IIS-0964541, CNS-0905602, CNS-1223623, and CNS-1228843 as well as National Institutes of Health grant LM011028-01.

References

1. S. Hansell, "AOL Removes Search Data on Vast Group of Web Users," *The New York Times*, 8 Aug. 2006.
2. A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets," *Proc. 29th IEEE Symp. Security and Privacy*, 2008, pp. 111–125.
3. R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," *Proc. 13th Conf. Usenix Security Symp.*, 2004, pp. 303–320.
4. R. Henry and I. Goldberg, "Formalizing Anonymous Blacklisting Systems," *Proc. 32nd IEEE Symp. Security and Privacy*, 2011, pp. 81–95.
5. M. Liedtke, "Netflix Users Watched a Billion Hours Last Month," *USA Today*, 4 July 2012.
6. M.Z. Lee et al., "Anon-Pass: Practical Anonymous Subscriptions," 2013; http://z.cs.utexas.edu/users/osa/anon-pass.
7. J. Camenisch et al., "How to Win the Clone Wars: Efficient Periodic n-Times Anonymous Authentication," *Proc. 13th ACM Conf. Computer and Communications Security*, 2006, pp. 201–210.
8. Y. Dodis and A. Yampolskiy, "A Verifiable Random Function with Short Proofs and Keys," *Proc. 8th Int'l Conf. Theory and Practice in Public Key Cryptography*, 2005, pp. 416–431.
9. J. Camenisch and A. Lysyanskaya, "Signature Schemes and Anonymous Credentials from Bilinear Maps," *Advances in Cryptology* (CRYPTO 04), 2004, pp. 56–72.
10. B. Lynn, "On the Implementation of Pairing-Based Cryptosystems," PhD thesis, Stanford Univ., 2007.
11. S.G. Stubblebine, P.F. Syverson, and D.M. Goldschlag, "Unlinkable Serial Transactions: Protocols and Applications," *ACM Trans. Information and System Security*, 1999, pp. 354–389.
12. P.C. Johnson," Nymble: Anonymous IP-Address Blocking," *Proc. 7th Int'l Conf. Privacy Enhancing Technologies*, 2007, pp. 113–133.
13. P.P. Tsang et al., "Blacklistable Anonymous Credentials: Blocking Misbehaving Users without TTPs," *Proc. 14th ACM Conf. Computer and Communications Security*, 2007, pp. 72–81.
14. "Ridership Reports | bart.gov," Bay Area Rapid Transit, 2013; http://www.bart.gov/about/reports/ridership.

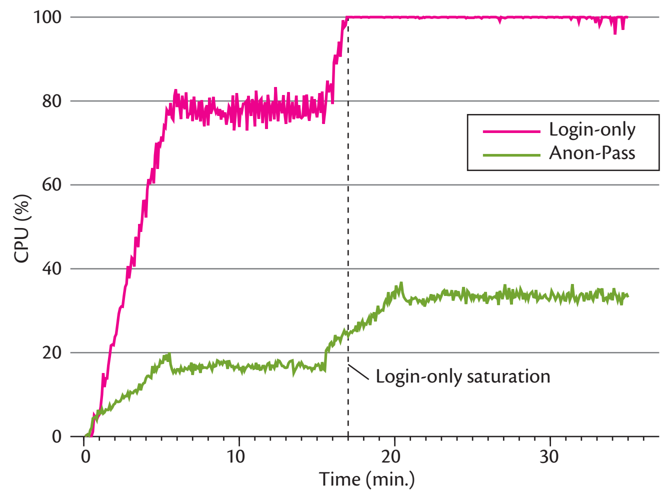

Figure 4. The authentication server CPU usage, measured every five seconds. The average CPU utilization for login-only at 6,000 clients (the first stable segment) is 77.9 percent (±2.42) and reaches saturation near the 17-minute mark, or approximately 8,100 clients. CPU utilization for Anon-Pass is 16.8 percent (±0.73) at 6,000 clients and 33.4 percent (±0.96) at 12,000 clients (the second stable segment).

Michael Z. Lee is pursuing a PhD in computer science at the University of Texas at Austin. His research interests include applied cryptography in anonymous systems and operating systems security. Lee received an MS in computer science from the University of Texas at Austin. Contact him at mzlee@cs.utexas.edu.

Alan M. Dunn is pursuing a PhD in computer science at the University of Texas at Austin. His research interests include system-level data privacy and using novel cryptographic constructions in realistic systems. Dunn received an MS in computer science from the University of Texas at Austin. Contact him at adunn@cs.utexas.edu.

Jonathan Katz is a professor in the University of Maryland's Department of Computer Science and director of the Maryland Cybersecurity Center. His research interests include cryptography, privacy/anonymity, and the "science of cybersecurity." Katz received a PhD in computer science from Columbia University. Contact him at jkatz@cs.umd.edu.

Brent Waters is an associate professor at the University of Texas at Austin. His research interests include cryptography and computer security. Waters received a PhD in computer science from Princeton University. He's a Microsoft Faculty Fellow. Contact him at bwaters@cs.utexas.edu.

Emmett Witchel is an associate professor in the University of Texas at Austin's Department of Computer Science. His research interests include operating systems, security, concurrency, and architecture. Witchel received a PhD in computer science and electrical engineering from the Massachusetts Institute of Technology. Contact him at witchel@cs.utexas.edu.