The atProtocol®

White Paper

Feb 2024
Version 2.0.1

Atsign Inc
1900 Camden Avenue, Suite 101
San Jose, CA 95124
atsign.com

Executive Overview

The atProtocol® is the underlying TCP/IP communication protocol used by the atPlatform. The atPlatform provides the building blocks to create apps that are privacy compliant and secure with edge-to-edge encryption. The atPlatform makes the network easier to manage because everything is directly addressable through unique identifiers called atSigns for people, entities, and things.

Each atSign creates its own public and private cryptographic key pairs. The private keys are kept private and public keys made available globally through the atProtocol. These key pairs are used to authenticate and negotiate symmetric keys for session and data encryption with other atSigns.

The atProtocol provides verbs for updating and the lookup of data with identifiers like location@alice.

The atProtocol supports both a public lookup, in which the querier of the data does not have to prove who they are, and a private lookup, which confirms the atSign of the entity asking for information and allows the owner of the data to decide what information to share. This means that it is possible to receive different answers for the same query if the requesting atSigns are different.

For example, suppose @alice sets the public lookup of location@alice to NYC and the private lookup to her mailing address. If anyone looks up location@alice without proving who they are, they will receive the response:

“NYC”

However, if @bob, her friend’s atSign, looks up location@alice, then the response will be:

“Conservatory Water at East 74th Street New York, NY”

The response to a query is completely under the control of the atSign’s owner, which allows for fine or coarse-grained customization of responses. The responses also include metadata that specify how long @bob can keep the data, or how often it should be refreshed.

The atProtocol also provides mechanisms for near real-time notifications that are not dependent on the services from the underlying operating system.

Technology Overview

Each participant in the atProtocol has a unique identifier known as an atSign. atSigns are centrally registered and the rest of the infrastructure is fully distributed.

Every atSign has a unique atServer microservice that is accessible on the Internet via a unique Fully Qualified Domain Name (FQDN) and TCP/IP port number and Secure Sockets Layer (SSL) certificate.

The atProtocol's objective is to provide end-to-end encrypted data transfer between two known atSigns, but also provides access to publicly available data that is cryptographically signed by the creating atSign. Data encryption itself is handled by the atSDK that uses the atProtocol, this separation of protocol and SDK allows extensible encryption without

changes in the protocol itself. Details on the data encryption and the atSDK are beyond the scope of this document.

The atProtocol defines a set of verbs that provide proof of the authenticity of the atSigns used and allows sharing of access to end-to-end encrypted data. Data ownership is held by the creating atSign and can be updated or rescinded at any time.

The atProtocol uses Transport Layer Security (TLS) over Transmission Control Protocol/Internet Protocol (TCP/IP). TLS is used to provide the first layer of authentication, using SSL client certificates as well as in-flight encryption of network traffic.

The atProtocol Design and Implementation

Functional Goals

- Easy to understand
- Intuitive to use
- Privacy by design

Technical Goals

- Dependable - Designed for global scale for at least 100 billion people, entities, and things
- Extensible - Protocol is simple enough for developers to build on it
- Near real-time - Data currency ensured

Architectural guidelines

- Distribute what you can, centralize what you must
- Always ask for permission
- Data is owned by the individual, persons, entities, or things

atSign characters

The atSign namespace, unlike DNS, extends to Unicode (specifically UTF-8), which allows characters beyond the Latin script. Emojis and other Unicode character sets are allowed thus expanding the namespace[1].

There are however some rules when choosing data identifiers, like location@alice, it cannot include UTF-8 white space, invisible characters, or control characters. atSigns are also Latin case insensitive, ensuring that @alice and @Alice refer to the same atSign.

[1] Currently only the Latin character set is open for registration, along with a limited set of emojis.

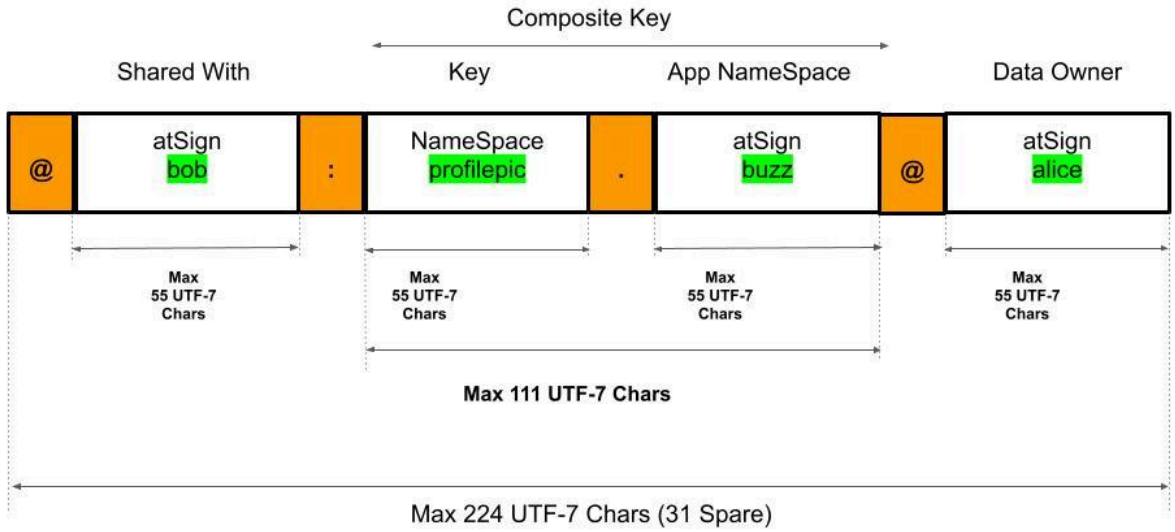
Following conventions set out in DNS and the URI internet RFCs, the following characters are reserved and cannot be used as part of an atSign.

!	*	'	(	)	;	:	@	&	=	+	$	,	/	?	#	[	]	{	}

atSign namespace

While the atProtocol itself has no real limits on the namespace being used, the atSign namespace has been constrained because of reasonable limits for the underlying keypair databases used in the clients of the atProtocol. If needed in the future, this can be opened up further with no changes needed to the protocol, but it will need changes to client databases.

The atSign itself is unique, so it's also usable as a unique identifier for the application namespace that makes up a composite key. For example, to ensure the application has no namespace clash, if an application was called atBuzz then, the application owner should own the @buzz atSign. In effect, if you own the @alice you also own alice@ in the namespace for every atSign.



While this may look limiting at first glance, the namespace is actually immense.

Conceptual Architecture

Being dependable at scale is the biggest challenge to building new infrastructure at Internet scale. The other part of being dependable are the three security guidelines of confidentiality, integrity, and availability.

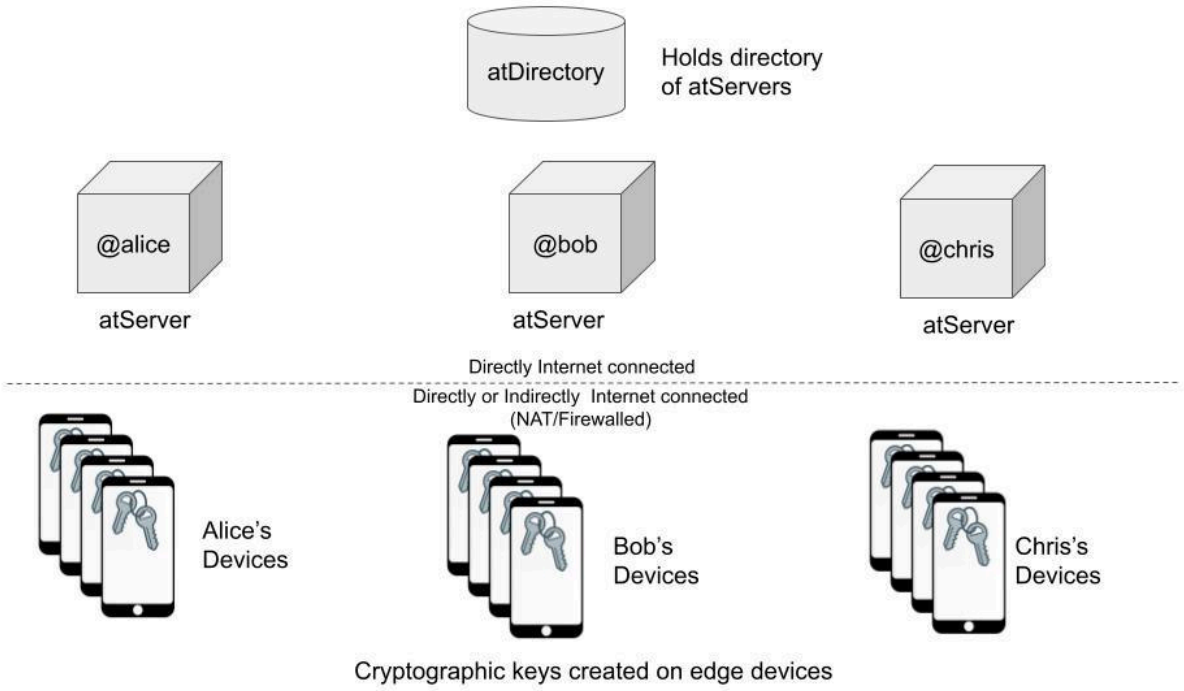
The architecture has to be simple to understand and the codebase must be tightly written. This allows it to be code reviewed and security tested extensively.

Two Tiers

The atProtocol has two tiers of servers: the atDirectory, which provides a global centralized namespace, and the distributed atServers, which provide an always-on cache for the data owned by an atSign.

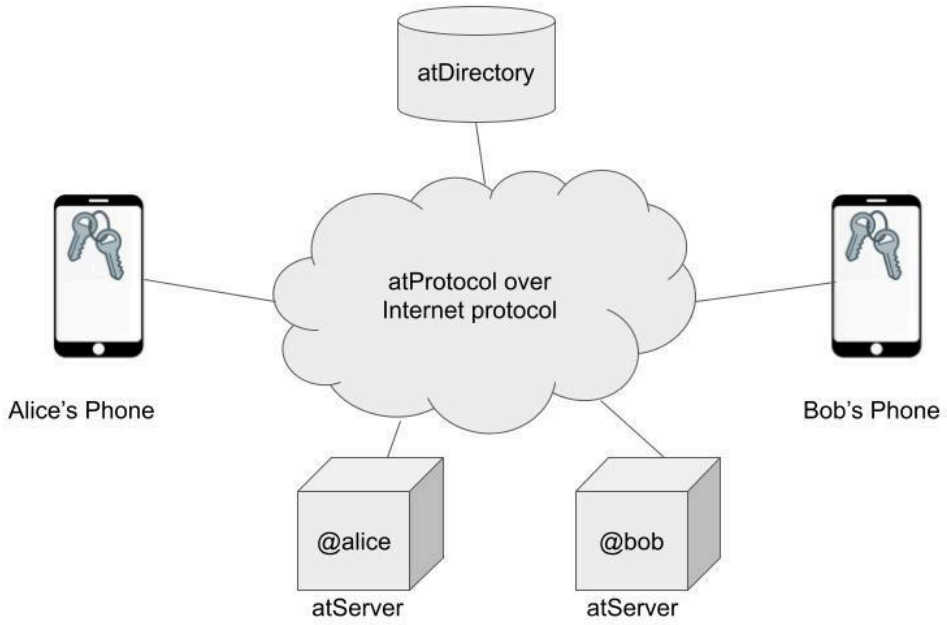
The atDirectory are the only centralized part of the atProtocol and are centralized to provide a single namespace and a globally dependable platform. No data beyond the atSign and responding authoritative atServer is held on the atDirectory. This information is considered public, and no authentication is required to look up the atServer for a particular atSign.

atServers provide the second tier of the atProtocol architecture, and are responsible for answering lookups for specific atSigns. atServers are generally deployed via orchestrators such as Docker Swarm or Kubernetes, but can also run as standalone executables. atServers have to be uniquely Internet addressable through use of an FQDN & Port pair that can be translated via DNS to a unique IP & Port.



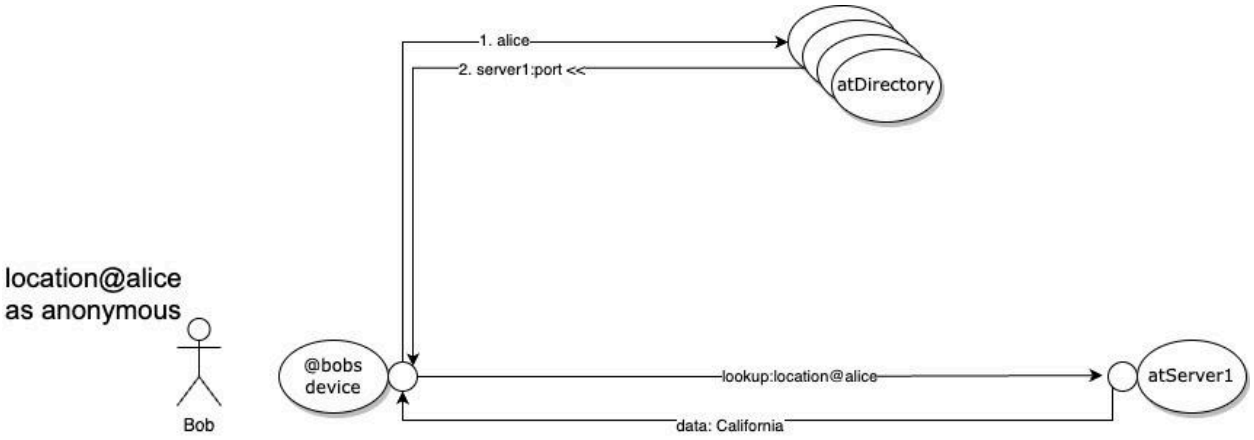
It is important to note that devices and clients do not have to be addressable on the Internet, as the atServer becomes the Internet point of presence for the atSign it serves. This means that **devices can be behind firewalls** or Network Address Translation (NAT) and still have full atProtocol functionality.

All components of the architecture speak the atProtocol and the atProtocol itself leverages existing protocols such as TCP/IP, DNS, and TLS. All network traffic is encrypted and authenticated with public/private key pairs that are generated on edge devices.



All atProtocol components are peers within the protocol but only atServers can connect and authenticate with each other.

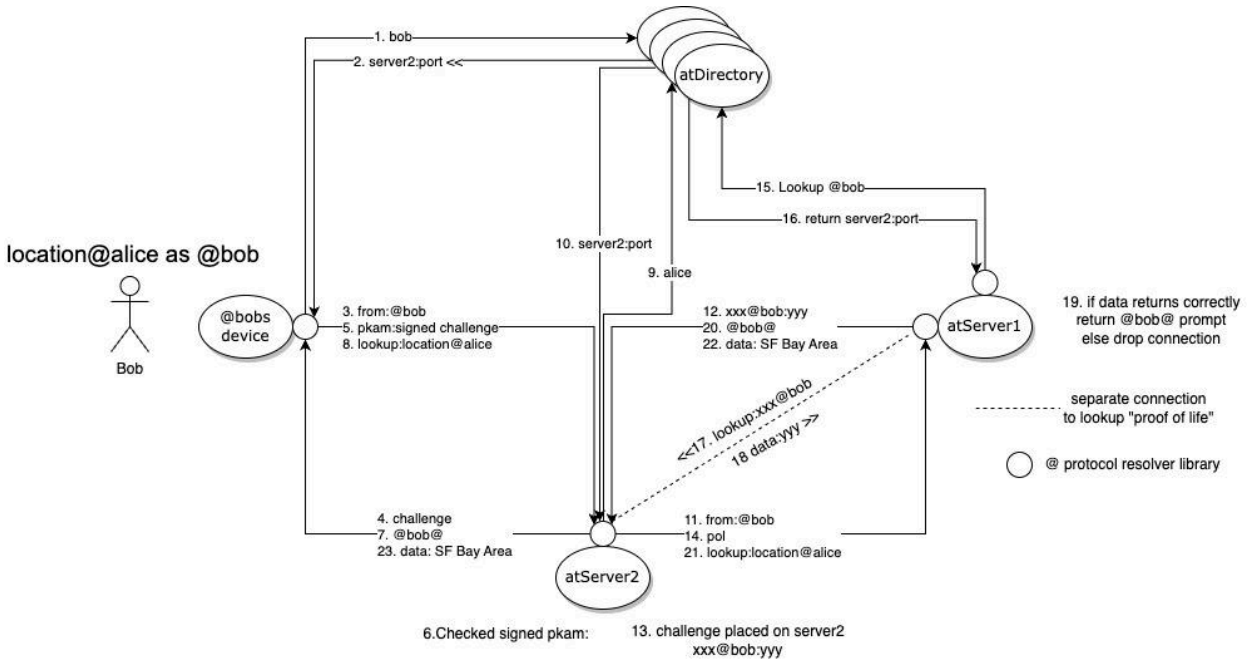
Public Lookup



When a public lookup of an atSign is performed, no authentication is required. Any information can be made public, however, in most cases the information will be private and require the requesting atSign to prove they are who they say they are. This also prevents the atDirectory administrators from knowing who is looking up any atSign apart from the IP used to connect and the atSign being queried.

The proof is the ability to place a cookie at a specified place in the atSign's name space. For this, the requesting atSign has to authenticate to the atServer serving the requester's atSign so the requesting atSign can perform the placement of a challenge. The challenge is to place a specific value in a specific location determined by the serving atServer. Once the challenge is completed, the resolver will tell the atServer it's ready for the lookup of the challenge. If that lookup is successful, then the resolver has successfully proved that they are who they say they are. At that point data lookups can be requested as the requesting, verified atSign. In addition the certificate of each atServer is used to mutually authenticate the TLS connection, when private data is being looked up.

Private Lookup



Although the diagram looks complicated, the process is a simple series of four verbs that are used in the TCP/TLS connections between the atServers and the resolver code. The resolver code can live as a single executable (at_cli) onLinux/OSX/Windows, as a library or daemon, or as an app on mobile platforms such as iOS or Android.

The atServers can run virtually anywhere, from a cloud server to dedicated hardware or running directly on Internet of Things (IoT) devices or mobile phones.

The only requirement is that the IP address of the FQDN and the port number given by the atDirectory is an **internet routable address and port**. The resolver, in some implementations, can use the atServer for resolving if it itself does not have full internet access. This enables the resolver to be on a private network but still be able to resolve

information. Again, this is similar to the way DNS resolvers can point to another DNS server for resolution.

atDirectory and Considerations for Technologists

The atProtocol pulls some of its high level architecture from DNS, a design that has stood the test of time. As with DNS, the atProtocol has atDirectory that provides redirection to the atServer for a particular atSign. Unlike DNS there are no top level domains (TLDs) or subdomains. Data for any atSign is served only by the atSign's atServer but to find that server, the atDirectory must first be queried. This information is considered public.

The atDirectory has been designed to scale to billions of atSigns and handle the requests for atSign lookups at near real-time, globally. To achieve this, in-memory databases are utilized and only the absolute minimum of data is stored. Just two things can be determined by communicating with the atDirectory: does the atSign exist, and if it does, how to reach it.

The atDirectory's sole role is to centralize the namespace and offer the root service dependably. When asking an atDirectory for the lookup of a particular atSign, the atDirectory will respond with a null if the name does not exist and with the FQDN of the atServer and the IP port number for that atSign if it does.

The format is <FQDN>:<PORT>.

The IP addresses of the atDirectory are held in DNS at root.atsign.org and the atDirectory itself runs on port 64 (which happens to be ASCII for @). Connecting to the atDirectory provides just an @ prompt, and sending a name followed by a CR will return the <FQDN>:<port>. The command to exit is @exit. Any future verbs will start with the @, but at present the only verb for the atDirectory is @exit.

atDirectory Infrastructure

The atDirectory is not a single machine, or even a single anything. The atDirectory service is run as a set of microservices and backended by in-memory, read-only databases across multiple datacenters.

Private and Hybrid Network deployments

The atPlatform can be deployed totally independently from the Internet, with a private atDirectory and atServers. The only requirements being that IP connectivity is available between atServers and that atClients have IP connectivity to an atDirectory and its own

atServer. To communicate using TLS each atServer needs to have an SSL certificate that is acceptable to both clients and all atServers. These certificates could be private or public on a private atPlatform.

On a hybrid network where a private atDirectory is used it can also forward unknown atSigns to the public atDirectory. Much like split horizon DNS, split horizon atDirectories, can be used to prevent/allow public and private atServers to communicate with each other. In this configuration private atServers would need to use public certificates.

atDirectory Verbs

The atDirectory only has one verb @exit, all other inputs are considered to be lookup requests.

(Commands entered are bolded).

```
cconstab@cally:~$ openssl s_client -ign_eof -brief root.atsign.org:64
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: ECDHE-RSA-AES256-GCM-SHA384
Peer certificate: CN = root.atsign.org
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Supported Elliptic Curve Point Formats: uncompressed
Server Temp Key: X25519, 253 bits
@spiderdeveloped
98545c6e-4ba2-52e0-9abc-6df80621ea3d.hornet.atsign.zone:2490
@isolatedcabaret
8abfc843-6407-5c92-822e-43cc1883656c.hornet.atsign.zone:2491
@realisticforeign
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
@colouredtropical
613120ec-e54a-5c9b-9bbc-e3fd454fc93f.hornet.atsign.zone:2488
@notanatsign
null
@@exit
CONNECTION CLOSED BY SERVER
cconstab@cally:~$
```

Note: The atProtocol uses TLS, so to connect to the atServers you need to speak TLS too. This is easily achieved using the OpenSSL command line tools with the arguments shown. It is important to use the "-ign_eof" option in OpenSSL to make sure your interaction is with the atServer, not the OpenSSL command line.

atServers and Considerations for Technologists

atServers provide the lookup service for a particular atSign. This forces the atServer to not mix an atSign's data with any other atSign's data, unlike web servers that can provide service to multiple websites at the same time.

atScheme

The atProtocol defines a secure URI (Universal Resource Identifier) for any data stored across the atPlatform (for example phone@alice) with one important difference: the value returned for an identifier is polymorphic, i.e. it depends on who is accessing the resource. In addition, the @ scheme, <atsign://>, creates a URL (Universal Resource Locator) that can be securely shared and interpreted. For example, atsign://phone@alice can be identified and used to locate phone@alice with all the security and permissions features applied to that resource. This convention is comfortable and particularly useful for storing reference values to be returned by atProtocol requests.

atServer Verbs

The verbs listed below will give you a high level understanding of the atProtocol itself, but most of the verbs also have extensive arguments for additional functionality.

The full protocol specification can be found on GitHub.

<lookup:>

The lookup verb allows for the lookup of a particular data identifier in the atSign's namespace. Shown below is a connection made to the **@realisticforeign**'s atServer and the lookup command is used to resolve **location.wavi@realisticforeign**. If a lookup is valid, the resulting information is returned with the data: header, newline, and a @ prompt ready for further commands.

```
cconstab@cally:~$ openssl s_client -ign_eof -brief
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
```

```
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@lookup:location.wavi@realisticforeign
data:{"location":"Paris france","radius":"2 mi"}
@lookup:somethingelse@realisticforeign
data:null
@^C
cconstab@cally:~$
```

If the lookup is not valid, then a null is returned again with the data: header, as we see above with the lookup of **somethingelse@realisticforeign**. The “lookup” verb used here provides public lookups. The same verb is used to resolve once authenticated as a particular atSign using the “from:” and “pol” verbs.

<from:>

The “from” verb is used to tell the atServer what atSign you claim to be, and the atServer will respond with a challenge. The challenge will be in the form of a full data identifier and a cookie to place at that address.

There is however a check of the TLS Clients SSL certificate before the from:atSign gives the challenge. The client SSL certificate has to match the FQDN list in the atDirectory for that atSign in either the CN or SAN fields in the certificate. If there is not a match then the atServer will drop the connection and report the error, as seen below.

```
cconstab@cally:~$ openssl s_client -ign_eof -brief
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@from:@colin
error:AT0401-Client authentication failed : Certificate Verification Failed
@CONNECTION CLOSED BY SERVER
cconstab@cally:~$
```

Here we see the connection being rejected as the client either did not provide a certificate or provided an incorrect one. In effect, this means that the atServer for @colin in the above example has to be used to connect and uses the from:@colin verb. Each atServer has to have an SSL certificate not just to act as a server but also to act as a client to other atServers.

If the certificate does match then a challenge is provided, as below. Note that the “openssl” command needs to provide not only the fullchain and the key, but also the caroot file that it provides, which in this case is fullchain.pem.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief -cert ./cert.pem -key
./privkey.pem -CAfile ./fullchain.pem
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
depth=2 C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN =
USERTrust RSA Certification Authority
verify error:num=2:unable to get issuer certificate
issuer= C = GB, ST = Greater Manchester, L = Salford, O = Comodo CA Limited, CN = AAA
Certificate Services
issuer= C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN =
USERTrust RSA Certification Authority
issuer= C = AT, O = ZeroSSL, CN = ZeroSSL RSA Domain Secure Site CA
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification error: unable to get issuer certificate
Server Temp Key: X25519, 253 bits
@from:@colin
data:proof:_4828b6c6-17cd-4d2e-91a3-d9eab3ebd591@colin:038d5c0e-34de-4227-9a86-da6
cefe83f73
@
```

In this case the atServer is asking for the challenge “038d5c0e-34de-4227-9a86-da6cefe83f73” to be placed at the location “_4828b6c6-17cd-4d2e-91a3-d9eab3ebd591@colin” with the header of “data:proof:”. The

challenge being that, to actually place that cookie on the @colin, atServer requires access to the @colin atServer which is only achievable for the person who has the @colin keys. Once in place, the challenge response needs to be public and cryptographically signed by the @colin atServer.

Public access is important as the @realisticforeign atServer will want to lookup that challenge to prove the request is actually from @colin. The type of challenge and the location is in the format of a UUID v4, ensuring that the likelihood of a namespace clash is mathematically unlikely, especially when coupled with the timeout of any challenges placed within a few minutes or the fact that they are cleared once used.

It is also worth noting that locations that are prepended with _ do not show up when using the scan verb, which is explained later.

<pol>

Once the cookie is placed, the proof of life (pol) verb is used to signal to the @realisticforeign atServer to check for the cookie on the @colin atServer. The atServer responds to the “pol” verb by looking up the location of the “from:” atSign using the atDirectorys, then connecting to the atServer for that atSign and using the “lookup” verb to look for the cookie at the challenge location. If the cookie is found then the prompt is changed to the atSign in the from:atSign, confirming that the “pol” verb was successful. If the cookie was not found for any reason, the @ prompt is returned.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief -cert ./cert.pem -key
./privkey.pem -CAfile ./fullchain.pem
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
depth=2 C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN =
USERTrust RSA Certification Authority
verify error:num=2:unable to get issuer certificate
issuer= C = GB, ST = Greater Manchester, L = Salford, O = Comodo CA Limited, CN = AAA
Certificate Services
issuer= C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN =
USERTrust RSA Certification Authority
issuer= C = AT, O = ZeroSSL, CN = ZeroSSL RSA Domain Secure Site CA
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification error: unable to get issuer certificate
```

```
Server Temp Key: X25519, 253 bits
@from:@colin
data:proof:_4828b6c6-17cd-4d2e-91a3-d9eab3ebd591@colin:038d5c0e-34de-4227-9a86-da6cefe83f73
@pol
@colin@
```

<cram:>

To place the cookie to prove ownership and access to the from:atSign, the resolver needs to authenticate to the atServer. The first method of authenticating is via a shared secret which is, by convention, a 512 bit random string. The same “from” verb is used to create the challenge and the secret is added to the challenge. Then a SHA512 digest is sent with the “cram:” verb to authenticate. The atServer does the same calculation and, if it matches what was sent, the authentication is agreed and the prompt updated to the atSign of the atServer. Once authenticated, the “update” verb can be used to place the cookie at the requested place, or any other information placed in the atSigns namespace.

The use of a shared secret is only used as a bootstrap to using the “pkam:” verb, which uses public/private key authentication.

The “cram:” verb is useful to start up the atServer with a known secret, and then to share it with the atSign owner via perhaps a QRcode to authenticate. Next, the atSign owner's device can cut and provide pkam keys, then it can remove the shared secret.

By using this method, no data would reside on the atServer until the pkam authentication is in place.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief
76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone:2499
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = 76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
```

```
Server Temp Key: X25519, 253 bits
@from:@tigerequivalent
data:_2ea52658-cfdf-4541-801e-8fee7633bc76@tigerequivalent:1b79a4cd-c84b-466d-927d-0824cb1fde1d
@cram:1ed369697e1f0f53be745dc02a6aa0efd390ed3dd9d500f98b23a7a9ec6e6d9eae38eae67021579592ca2a464f5cfaafc9888d3b3aac70078554f0592022197b
data:success
@tigerequivalent@
```

To generate the digest, the at_cram command line tool can be used in another Unix shell, then cut and pasted into the "cram: " verb. Here we have successfully authenticated to @tigerequivalent's atServer.

```
cconstab@Liberator:$ dart at_cram.dart ./tigerequivalent _2ea52658-cfdf-4541-801e-8fee7633bc76@tigerequivalent:1b79a4cd-c84b-466d-927d-0824cb1fde1d
1ed369697e1f0f53be745dc02a6aa0efd390ed3dd9d500f98b23a7a9ec6e6d9eae38eae67021579592ca2a464f5cfaafc9888d3b3aac70078554f0592022197b
cconstab@Liberator:$
```

<pkam:>

Whilst shared secrets offer solid security, the secret is shared with both the client and the atServer, so if you do not trust the administrator of the secondary with a shared secret, you might consider this a security risk. The solution to this issue is the use of a public/private key pair. You create a cryptographic pair of keys, one that you keep for yourself (private) and the other you give to the administrator of the atServer (public). The challenge from the "from:" verb is given in the same way but instead of using the "cram:" verb to authenticate you use the "pkam:" verb. The "pkam:" verb used to send a cryptographically signed version of the challenge, which can then be validated by the atServer with the public key.

This allows only the holder of the private key to authenticate without any shared secrets on the atServer. In fact, any shared secrets can and should be safely deleted using the "delete:" verb once "pkam:" has been successful.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief 76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone:2499
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
```

```
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = 76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@from:@tigerequivalent
data:_99af7adb-3149-4ac1-bd69-518a2087b9fa@tigerequivalent:f90555dd-e704-474a-84b2-c
1d27c4d7a32
@pkam:ViL6vTa+5GU/jC5L24NQnVAO/3NmnjQ+y5LQYLP+pNQg623AQ7RP//FbNgoPYDmVP
tI0s692vE0cSeApRJKmCZ+N9LZL0EIekpoBjSMo67GRECwFxdw0F3EACr/0MFgXYlZ2dDzNq
qqMffPPHylhepzHUi6sssljZUrZ1KZKDUWVaWeytALLf7kymDd6bZj3xdfJHvK2/A8klOlFEbtnB
Qz2Th6fg0aTXdXxgR7I9uS0Vbu0bUX/k/1DrdGIAftY5MCO7t0/KdB6Sngn7Pnm+P48kSrmoT
cwncActiVWuSDelZoy4omNou0fthdooYGcyGMXzlscO/coj7ec8UN0CA==
data:success
@tigerequivalent@
```

To sign the challenge, the private key needs to be used together with the [at_pkam command line tool] in another shell or within an application.

```
cconstab@Liberator:$ dart main.dart -p ./@tigerequivalent_key.atKeys -r
_99af7adb-3149-4ac1-bd69-518a2087b9fa@tigerequivalent:f90555dd-e704-474a-84b2-c1
d27c4d7a32
ViL6vTa+5GU/jC5L24NQnVAO/3NmnjQ+y5LQYLP+pNQg623AQ7RP//FbNgoPYDmVPtI0s692v
E0cSeApRJKmCZ+N9LZL0EIekpoBjSMo67GRECwFxdw0F3EACr/0MFgXYlZ2dDzNqqqMffPP
HylhepzHUi6sssljZUrZ1KZKDUWVaWeytALLf7kymDd6bZj3xdfJHvK2/A8klOlFEbtnBQz2Th6fg
0aTXdXxgR7I9uS0Vbu0bUX/k/1DrdGIAftY5MCO7t0/KdB6Sngn7Pnm+P48kSrmoTcwncActiV
WuSDelZoy4omNou0fthdooYGcyGMXzlscO/coj7ec8UN0CA==
```

<update:>

The “update” verb is used to do just that, it allows an authenticated atSign, after using the “cram:” verb, to update the atSign's namespace entries. The format for those entries follows the following pattern:

<public/@share with sign>:key<@sharing sign> value

Using this format, both public responses to the “lookup” verb as well as specific responses for particular authenticated users after using the “pol” verb can be set, as in this example:

update:public:email@tigerequivalent tiger@example.com

This will create a public record for email@tigerequivalent that could be resolved by using the “lookup” verb.

update:@realisticforeign:email@tigerequivalent tiger@work.example.com

This will create a record that only @realisticforeign could resolve and then only after authentication via the “pol” verb.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief
76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone:2499
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = 76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@from:@tigerequivalent
data:_9d111144-0879-411a-bead-ca38c6a464be@tigerequivalent:c4631ff9-4cdc-4ff9-a274-1
a19c065e452
@pkam:RZBSLlEnRweNfgDDPvclATrCkyYaX7M1lGhQsVldv3m03VfNYHl46Bu01NXdaJAd3L
CYYWQbrvgkjBW/DLHAQbNr7ndwjsv69xzUZkxoldojEJ9ay2J3Z7UuYOceNjJfOqb9q5oKkFCP
T0fWxmnla4sITQ3rQIFhKs+/Kk9bj5Z0KCIUV+VIQQZAayWiW7oobqocxpo9awDmwTnj68BOX
ey88ZqxrCp9kN5bwoa3lkb+s+q7QZNcodc1wdFR4yoK9LuyR++fdogp89y9t7bc2oofujax5pOQ
Bn3nWqg9nYHptLyFeIBI0jBaGo6wPwtmVavj/SHSEK8KXUopPNDsAA==
data:success
@tigerequivalent@update:public:email@tigerequivalent tiger@example.com
data:18
@tigerequivalent@update:@realisticforeign:email@tigerequivalent tiger@work.example.com
data:19
@tigerequivalent@
```

<plookup:>

The “plookup:” verb provides a proxied public lookup for a resolver that perhaps is behind a firewall. This will allow a resolver to contact an atServer and have the atServer lookup public atSign’s information. This will be useful in large enterprise environments where they would want all lookups going through a single atServer for the entity or where a single port needs to be opened through a firewall to lookup atSigns.

Shown below is an example of “plookup:” showing the public value of email@tigerequivalent then the value for the same query but as @realisticforeign using the “lookup” verb.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@from:@realisticforeign
data:_99230666-2590-4d2f-a68e-819add5ecad5@realisticforeign:ef4293cd-350d-431e-8c9e-
24d2f78520aa
@pkam:HoHl+SDBmvLrqdyX4y0wsC1gbRVLrSVo2aKkCrLXlfyiFNIbHzFX7m+TRqonEZ+uX4c
2y9A6lsk1QY9rUsTKrJWmwFfZ3fpwpKkuYEwHmlMXC7gVHMi2CbLxesYCSX7XHd8ZEHIJx
DR2S200k3gGwkQHmgBwd0nXsdSQap6ZW8jmp6Fxa88qzOYMvlKX9uGmJQkQ+EPAyY+8W
OETFW3hf9sQ/EI4lXfh27atrnqljqqnklvWWulFnUop0oXTeiH8vXBC6wxZ1lJNo0+6pEXmoos0r
gLqcq+gbPMpxj9zl3bTCRqRTAmQ8+Wj4U3P4YlxvJuluOwjl9j2y0oHUhKLyA==
data:success
@realisticforeign@plookup:email@tigerequivalent data:tiger@example.com
@realisticforeign@lookup:email@tigerequivalent
data:tiger@work.example.com
@realisticforeign@
```

<scan>

The “scan” verb is used to scan the available data identifiers for you, either at the public level or once the pol process has been completed. This allows data identifiers to be

discovered and perhaps be collected. If a data identifier has a _ character as its first character, then it is omitted from the scan list, although it can still be looked up if known. The following example shows just that:

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@scan
data:["location.wavi@realisticforeign","locationnickname.wavi@realisticforeign","publickey@re
alisticforeign","signing_publickey@realisticforeign"]
@
```

The “scan” verb only scans for data identifiers that are available to you at your current authentication state. If unauthenticated, only public information is provided, once the “pkam”/”cram” or “pol” verb has been successfully then private data identifiers to the atSign that authenticated will be visible.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@scan
data:["location.wavi@realisticforeign","locationnickname.wavi@realisticforeign","publickey@re
```

```
alisticforeign","signing_publickey@realisticforeign"]
@from:@realisticforeign
data:_96dc3af8-1657-4fd6-86e3-53726876e87f@realisticforeign:989094d4-fa65-418a-878b-df72535adbb6
@pkam:BLweOba0B68Yx25EPoTCDFxL08HUslT3TTE51bTcMTljHRKacVPWeXaiUo8Zuvehv3XtD5eQbU3muWv1Md0xDSjIn7dnzNCYp99LW7I61cX2m3Aw48DTpH4w1xhBBGPII6riDl9eP4InIZQehMmqcpGZknTgXO+MTk8EFYwUr8s1opGCPD2DE4mGyCXAUN4lhXk1hSLoqDOUk5qWHR6sb/VO9hhlPZBoZ4hPjMaTCWxKIoGGnxdPsy4EntKURtzVXwUC7UiySy8WxVrZULTacDbFR8Y1Kbd4in4X5oZGWcLAxYlQxHEf5dpbafTUoI0MGF9wM1gSt5CLJzh/331Igw==
data:success
@realisticforeign@scan
data:["@realisticforeign:signing_privatekey@realisticforeign","cached:public:_notlisted@realisticforeign","cached:public:email@tigerequivalent","public:location.wavi@realisticforeign","public:locationnickname.wavi@realisticforeign","public:publickey@realisticforeign","public:signing_publickey@realisticforeign"]
@realisticforeign@
```

<llookup:>

As an authenticated user after the “cram:” verb, the “llookup:” verb can be used to locally lookup data at the data identifier location stored on the atServer.

```
cconstab@cally:~/@colin$ openssl s_client -ign_eof -brief ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms: ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@scan
data:["location.wavi@realisticforeign","locationnickname.wavi@realisticforeign","publickey@realisticforeign","signing_publickey@realisticforeign","weather@realisticforeign"]
@from:@realisticforeign
data:_e8defea5-d893-4a22-851f-db249e87a485@realisticforeign:110c8842-b061-43c7-9f94-8f91fc626391
@pkam:GkzZ8/z7H8NZdPIqQAVWf/o3fG+H3KnapXlQwe1IVSBXsoqIyWugSj0odOZfFIq84AD
```

OgZW9UDdmTOLvXXqjg4X4/WtDGZTzgMGmYl7NXNR6otkbWDoPW6e3vX/ly7tmh47MiuEQ
VScjgGRw0Dtum5geWjKGt1o2GVMqRWzZVF7BjUhKCwSn6vnq/ecmAy4d12W+XBV70p937E
ioZtUzEpn+l8p7sSWcvV5pwcWNHRCVAY1uz6DiKeik6Rc7OUJbUJSxMPhUMMwywVDitUXn
O/xw6mVCYkhHrHl13kDm6jSvMSEWTKvBw6zG044ZbsN71sV+0mMvBXbfxFw2l/SX7g==
data:success
@**realisticforeign@scan**
data:["@realisticforeign:signing_privatekey@realisticforeign","cached:public:_notlisted@realisticforeign","cached:public:email@tigerequivalent","public:location.wavi@realisticforeign","public:locationnickname.wavi@realisticforeign","public:publickey@realisticforeign","public:signing_publickey@realisticforeign","public:weather@realisticforeign"]
@**realisticforeign@llookup:public:publickey@realisticforeign**
data:MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjCkLI6RjrFMl3GmHoUg0DgHPa0vv4nZsWbKv8j/ae6C34Gc/dt7dC/2/bSc7TYD45QZtOPdKjCQoVAR5WbLCiWwt49Kyt0IeNT4zPftFvhYgFNIrYKPI6QMnzNJ0SKbYOnE0HhOjcIjIbMTsh4griOwYHBu/eXdcFU+DpGKbuX/mWxZqeqyNmd1vgJDQBJNaUXmzCyA7xovaJ+EnZVA3hddwUZhBkiUCm6msVj23nYNbQ+ddkS+wC1iTxVaPV13LkINkAHccCpX+Rv+hxBkus5ppa874xlKSm/+r/whG7gZ3oXgZTlrCvchmS2wStSKwupnZLBxYxZbBTyX678U5dQIDAQAB
@realisticforeign@

<delete:>

The “delete:” verb is used for just that: deleting data identifiers and the data they contain, but it can only be used by an authenticated user following the “cram:” verb. The “scan” verb is useful to find the correct data identifier to delete.

cconstab@cally:~/@colin$ **openssl s_client -ign_eof -brief**
ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@**from:@realisticforeign**
data:_30706078-63f0-428e-98ac-a7e60b2f81b3@realisticforeign:42a9b6f3-4c74-4f0d-918d-25bc46a2b604
@**pkam:BBNHiBgYervD1SbUOwHeyFl8bD802zKq+saAbhc1d4NtebqFGuYhALsjIqBWEkE5Ve**

```
xViN9fpG1S6ZTnrMsxzoG8abLvhk+t8SdfmoEJ/zwsbommqz3sw2UsYDJSj/qBSmuVfG86wa
ZEEvpbL4ytNmCyqmF299eH5X/CUexHvHin3uooxYjLjbRnPo1aUj3IqIaOq4pz7T4IaEZsaiVyIL
gH5h8YE/JXgFo6QhnEwDWVHB3w8uB4Pb6dIJeenHV/BULrL9S6hQXKlhOiOUAQXgJGiYi3Lb
V4xcIy6vX6tXjtwDxAFccWiV5s5B/tezNIx3VLtBfIA4BwjJi/NjnajQ==
data:success
@realisticforeign@update:public:test@realisticforeign test data
data:13
@realisticforeign@llookup:public:test@realisticforeign
data:test data
@realisticforeign@delete:public:test@realisticforeign
data:14
@realisticforeign@llookup:public:test@realisticforeign
data:null
@realisticforeign@
```

<notify:> and <monitor> The notification subsystem

The atProtocol also provides methods of push notifications that are independent of the operating system being used.

The “monitor:” verb is used to monitor either all or specific notification events that are sent using the “notify:” verb. Notifications are both queued and managed by the atServer, and the status of an individual notification can also be seen.

The notification subsystem works to get the notification to the atServer of the atSign being notified as efficiently as possible, and also handles retries in the event of things like network failures.

Sending a notification to @colin with the “notify” verb

```
cconstab@cally:~$ openssl s_client -brief
76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone:2499
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SH
A384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = 76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@from:@tigerequivalent
data:_4f78605b-2c35-4ed7-90f0-6e2d0f4d1637@tigerequivalent:749c97c9-a49a-4222-a20
5-87097f9fb325
```

@**pkam:M+ypyQsoexpnwzisyS9M+sN9xmDHXXbzerQvV13kghthGIt/GWTnJAb+x7mbohST3NHuqn2MQw0cAe5kMXu/OmEElAxl9/kFtns4qz16d1brDejL1iJSqJETeIN6A53isoNoLsuoEnKbYA6OBeFlMFw9eqYN5HC/AQaB58BBPnlMc9dk6UpiFW0KCLymltLl9UTc5U7n0n2daDinI6LPxO/wAU5fY8+VMM8sacT9i5MZyawtZvpSjELkPVgM+/iUY/e6EIHvuafcoMqrtaZMHNbp9M6B3WI6210pWECusUbTl00sHoCvMt4hybxuqf7gucLjBu0f9uY2BloLIhzrpw==**
data:success
@tigerequivalent@**notify:notifier:system:@colin:location.wavi@tigerequivalent**
data:680b8352-42c0-4e0b-a0bf-5b551c78512d
@tigerequivalent@

The “monitor” verb allows a client device to monitor notifications in near real-time. Notifications are sent to all sessions running the “monitor” verb.

Receiving a notification on @colin with the “monitor” verb.

cconstab@cally:~/@colin$ **openssl s_client -ign_eof -brief 79b6d83f-5026-5fda-8299-5a0704bd2416.hornet.atsign.zone:1029**
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms: ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = 79b6d83f-5026-5fda-8299-5a0704bd2416.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@**from:@colin**
data:_a101b1e1-7682-47ea-8524-9807511cfa11@colin:b75d1f96-d24a-4a4b-a263-9e82db64804e
@**pkam:eXSq+1TM30hecmqyw+xKrZOF0Q5I+zgSc98Lsr5fSB5HQtxGS00aYsdvg4IaI2RvcsJIg7KJIF0PtOfGnxN7Qq+xgsZGKN+0BN4xUcpa++H8MDpw2HXRmrzN18fRA1JmJzwdzJN95rgMHrq2OC+avt+gx3GoSPpkwXQLXvjcXoRwD/4/sJ8npKx6ONYU2AN+tXjhlJQl/vrl1dKaFFt0mgsijW8S8m0JC9EiSQsLKRXDNV0Kcux61QJ+5YVvFtbIplsH12sPTrLJjk9oWxE+RYSStSg20nP/X7q6OsmblBhojVH38YDx0dLyswLovjcOPiRGNn/LcCU7EKZGc5VRfw==**
data:success
@colin@**monitor**
notification:
{"id":"f8c8a763-d9f4-4660-8595-5bafa9ea06b4","from":"@tigerequivalent","to":"@colin","key":"@colin:location.wavi@tigerequivalent","value":null,"operation":"null","epochMillis":1626462703744}

<stats:>

The “stats<:>” verb provides an authenticated atSign with a number of stats. They can be listed with stats and detailed individually with “stats:<number>.”

cconstab@cally:~/@colin$ **openssl s_client -ign_eof -brief ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone:2489**
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms: ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = ba46657d-227b-5ecf-ad16-161f80249772.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@**from:realisticforeign**
data:_a3ac3fd9-d57e-4f7e-92f4-1bfebc1d5a8f@realisticforeign:11aebdd6-b0f7-44f0-8f9e-961546f70952
@**pkam:D2kdNJ9yzdpU6EZlbbcgv8DkzG/+g7/ecjkba5DVoCyK2lRGZDabnAM5D/ik+cbmm1Ix1fonHCNcCXHhfVLqFOjVC2lEu//oH7Hean3dmTtfva1jxuyAQcia0VWQWC0u4EFZeeVqREK7TChctB2VXOw4VBcHlS9ub2UumD9EEzzfsIFbmuUkJa8NemVHWwgBdhym1c2XFOB/x91WUORmFLBdX9QNCWp6L1tuQHEmdSOsQLu2nc8Fg0CtIC8Z8uQf8wHmHRAyOHhZq87b8hrVXYnMUdc+TWEC+w/NeENKHOmS/nhpXZEHXAMTCLY+iki7uHuqhZyvePaLzklt2whoEg==**
data:success
@realisticforeign@**stats:1**
data: [{"id":"1","name":"activeInboundConnections","value":"1"}]
@realisticforeign@**stats:5**
data: [{"id":"5","name":"topAtSigns","value":"{\"@tigerequivalent\":5}"}]
@realisticforeign@**stats**
data: [{"id":"1","name":"activeInboundConnections","value":"1"},
{"id":"2","name":"activeOutboundConnections","value":"0"},
{"id":"3","name":"lastCommitID","value":"14"},
{"id":"4","name":"secondaryStorageSize","value":14131},
{"id":"5","name":"topAtSigns","value":"{\"@tigerequivalent\":5}"},
{"id":"6","name":"topKeys","value":"{\"publickey@realisticforeign\":2,\"location.wavi@realisticforeign\":2,\"signing_publickey@realisticforeign\":1,\"public:_notlisted@realisticforeign\":1}"},
{"id":"7","name":"secondaryServerVersion","value":null}]
@realisticforeign@

atServer Synchronization Verbs

In order to synchronize the data between devices, such as mobile phones, and an atServer there are two verbs provided on the server: “stats:” and “sync:”. These two verbs provide enough data to see if the device is behind or ahead of the data stored on the atServer.

Each update of data to the atServer increments a counter (commitId) which is displayed after each update. The devices also keep track of their counters and then can determine if they are ahead or behind the counter on the atServer.

If the device is ahead of the atServer, then the device can use the “update:” verb to update the data. If it is behind the atServer, then it can use the “sync:” verb to pull the deltas.

This mechanism allows for multiple devices to sync to the same atServer. For example, a mobile phone, a tablet, and a PC all synching to a single atSign, but also multiple applications on those devices synching to a single atServer simultaneously. Each application on each device has a copy of the data and can update and sync it at any time.

Syncing is the most complex part of the protocol, and it also influences the way data is stored by applications. As all private data is stored in an encrypted format on the atServer by the atPlatform SDK, delta updates of data are not possible because the encryption (AES256) ensures that the encrypted data is different even if a single bit changes.

```
cconstab@cally:~$ openssl s_client -ign_eof -brief
76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone:2499
CONNECTION ESTABLISHED
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
Requested Signature Algorithms:
ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+S
HA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Peer certificate: CN = 76103630-0f85-5dec-8a91-364bf93b0e8d.hornet.atsign.zone
Hash used: SHA256
Signature type: RSA-PSS
Verification: OK
Server Temp Key: X25519, 253 bits
@from:@tigerequivalent
data:_ff6dc12b-8b11-4416-81fe-6c4d58da7581@tigerequivalent:57818857-fe08-4390-8731-a
c5d48c410ce
@pkam:Iw9eaZ0lIos1euiK+UA9ot/a9/644GvRsgDwRergCwXdzROHH1TMXwsht+WdDYktbA
bjOQ28vzf6Sh045aJhHakIUzr6MzLwBiMtEQIDD75DLftKa0eiJrS9Xp7u7546iWzuz9/UDzX02
CA5DkXnzwEMBgxKLwsNZOzEDVwG+4jj/IrAU+JPqF93OHJiTFJUgeKp+IEBz1ZWqtOBwP0iY
DtyxTD4gs2P9TAZdSfdhHTiFSTmPvPd/40cOwL9YZdNfsACU02ORd/2efmGmem6GcPdjVHb
```

1ve7K4WANaPityM4umEfQBjyW+kmlkmL3dEVUBbu3xZlie+g3pwSHSgdpw==
data:success
@tigerequivalent@**stats:3**
data: [{"id":"3","name":"lastCommitID","value":"26"}]
@tigerequivalent@**sync:24**
data:[{"atKey":"cached:public:weather@tigerequivalent","operation":"#","opTime":"2021-07-15 00:00:00.767668Z","commitId":25,"value":"wetter","metadata":{"createdAt":"2021-07-15 00:00:00.767302Z","updatedAt":"2021-07-15 00:00:00.767324Z"}},{"atKey":"@realisticforeign:notify@tigerequivalent","operation":"*","opTime":"2021-07-15 01:19:38.381118Z","commitId":26,"value":"hello","metadata":{"isBinary":"false","isEncrypted":"false","createdAt":"2021-07-15 01:19:38.380727Z","updatedAt":"2021-07-15 01:19:38.380727Z"}}]
@tigerequivalent@

The example above shows connecting to @tigerequivalent by using the “stats:3” verb to find the lastCommitID, then getting the delta values from 24 using the “sync:24” command.

Note these values are public and hence in clear text. With encrypted values, the values would appear in encrypted base64 encoded strings.

Metadata is also synchronized with these commands and typically this data is not encrypted. Metadata that needs to be protected can be encrypted at the application level.

Conclusion

The atProtocol provides a unique identifier that people, entities, and things can own independently. The owner of an atSign can provide public or uniquely end-to-end encrypted data for other atSigns and send notifications in near real-time.

The technologies used by the atProtocol such as DNS, SSL certificates,TLS, RSA, ECC, AES are all mature and widely understood. The architecture provides seamless network traffic navigation of firewalls and network address translation.

The causal effect of the atProtocol is far reaching, allowing self sovereign identities, end-to-end encryption for everything, and polymorphic data.

Whilst the atProtocol is as simple as possible, most developers will want to use an SDK that further abstracts the wire protocol to the application layer.

The atProtocol, together with the accompanying atPlatform with its SDK, provides new functionality for developers to create new experiences for people around the world.